CATCH: Context-based Meta Reinforcement Learning for Transferrable Architecture Search

Xin Chen⋆[1], Yawen Duan⋆[1], Zewei Chen[2], Hang Xu[2], Zihao Chen[2], Xiaodan Liang[3], Tong Zhang⋆⋆[4], Zhenguo Li[2]

[1] The University of Hong Kong
[2] Huawei Noah's Ark Lab
[3] Sun Yat-sen University
[4] The Hong Kong University of Science and Technology

Abstract. Neural Architecture Search (NAS) achieved many breakthroughs in recent years. In spite of its remarkable progress, many algorithms are restricted to particular search spaces. They also lack efficient mechanisms to reuse knowledge when confronting multiple tasks. These challenges preclude their applicability, and motivate our proposal of CATCH, a novel Context-bAsed meTa reinforcement learning (RL) algorithm for transferrable arChitecture searcH. The combination of meta-learning and RL allows CATCH to efficiently adapt to new tasks while being agnostic to search spaces. CATCH utilizes a probabilistic encoder to encode task properties into latent context variables, which then guide CATCH's controller to quickly "catch" top-performing networks. The contexts also assist a network evaluator in filtering inferior candidates and speed up learning. Extensive experiments demonstrate CATCH's universality and search efficiency over many other widely-recognized algorithms. It is also capable of handling cross-domain architecture search as competitive networks on ImageNet, COCO, and Cityscapes are identified. This is the first work to our knowledge that proposes an efficient transferrable NAS solution while maintaining robustness across various settings.

Keywords: Neural Architecture Search, Meta Reinforcement Learning

1 Introduction

The emergence of many high-performance neural networks has been one of the pivotal forces pushing forward the progress of deep learning research and production. Recently, many neural networks discovered by Neural Architecture Search (NAS) methods have surpassed manually designed ones on a variety of domains including image classification [47, 61], object detection [61], semantic segmentation [5], and recommendation systems [31]. Many potential applications of

⋆ Equal contribution.
⋆⋆ Correspondence to: tongzhang@tongzhang-ml.org

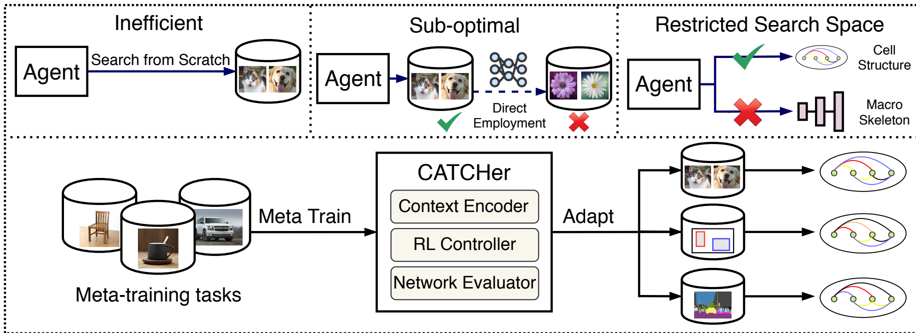


Fig. 1: Upper: drawbacks of current NAS schemes. Lower: the overall framework of CATCH. Our search agent, CATCHer, consists of three core components: context encoder, RL controller and network evaluator. CATCHer first goes through the meta-training phase to learn an initial search policy, then it adapts to target tasks efficiently.

practical interests are calling for solutions that can (1) efficiently handle a myriad of tasks, (2) be widely applicable to different search spaces, and (3) maintain their levels of competency across various settings. We believe these are important yet somewhat neglected aspects in the past research, and a transformative NAS algorithm should be able to respond to these needs to make a real influence.

Many algorithms [33, 37] have been proposed to improve the efficiency of NAS. However, they lack mechanisms to seek and preserve information that can be meaningfully reused. Hence, these algorithms can only repeatedly and inefficiently search from scratch when encountering new tasks. To tackle this problem, a rising direction of NAS attempts to create efficient transferrable algorithms. Several works [23, 36] try to search for architectures that perform well across tasks, but the solutions may not be optimal on the target tasks, especially when the target task distributions are distant from the training task distributions. Some recent works [28, 15] use meta-learning [16, 27] for one-shot NAS instead. With recent critiques [56, 26] pointing out some one-shot solutions' dependence on particular search spaces and sensitivity to hyperparameters, many concerns arise on the practicality of these meta NAS works based on one-shot methods. To avoid ambiguity, throughout this paper, *tasks* are defined as problems that share the same action space, but differ in reward functions. In NAS, the change of either the dataset or domain (e.g. from classification to detection) alters the underlying reward function, and thus can be treated as different tasks.

Striking a balance between universality and efficiency is hard. Solving the universality problem needs a policy to disentangle from specifics of search spaces, which uproots an important foundation of many efficient algorithms. The aim to improve efficiency on multiple tasks naturally links us to a transfer/meta learning paradigm. Meta Reinforcement Learning (RL) [38, 25] offers a solution to achieving both efficiency and universality, which largely inspired our proposal of CATCH, a novel context-guided meta reinforcement learning framework that is both search space-agnostic and swiftly adaptive to new tasks.

The search agent in our framework, namely CATCHer, acts as the decision-maker to quickly "catch" top-performing networks on a task. As is shown in Figure 1, it is first trained on a set of meta-training tasks then deployed to target tasks for fast adaptation. CATCHer leverages three core components: context encoder, RL controller, and network evaluator. The context encoder adopts an amortized variational inference approach [1, 38, 24] to encode task properties into latent context variables that guide the controller and evaluator. The RL controller makes sequential decisions to generate candidate networks in a stochastic manner. The network evaluator predicts the performance of the candidate networks and decides which nets are valuable for training. All three components are optimized in an end-to-end manner.

We test the method's universality and adaptation efficiency on two fundamentally different search spaces: cell-based search space [13] and Residual block-based [19, 57] search space. The former focuses on cell structure design, while the latter targets macro skeleton search. With NAS-Bench-201 [13], we can compare CATCH fairly with other algorithms by eliminating performance fluctuations rising from different search spaces and training settings. Our experiments demonstrate CATCH's superiority over various other works, including R-EA [40] and DARTS [33]. On Residual block-based search space, we use image classification tasks on sub-datasets of ImageNet [10] as meta-training tasks, and then adapt the CATCHer to target tasks, such as image classification on full ImageNet, object detection on COCO [30], and semantic segmentation on Cityscapes [9]. CATCH discovered networks on these tasks with competitive performance and inference latency. Our results demonstrated CATCH's robustness across various settings, easing previously raised concerns of NAS algorithms' sensitivity to search space, random seeds, and tendencies to overfit to only one or two reported tasks.

Our key contribution is the first attempt to design an efficient and universal transferrable NAS framework. It swiftly handles various tasks through fast adaptation, and robustly maintains competitive performance across different settings. Our work brings along new perspectives on solving NAS problems, including using amortized variational inference to generate task characteristics that inform network designs. It also demonstrates the possibility of creating efficient sample-based NAS solutions that are comparable with widely-recognized one-shot methods. With competitive networks identified across classification, detection, and segmentation domains, it further opens the investigation on the feasibility of cross-domain architecture search.

2 Related Work

NAS is an algorithmic approach to design neural networks through searching over candidate architectures. Many harness the power of Reinforcement Learning (RL) [60], Bayesian Optimization [3, 4], Evolutionary Algorithm [14, 39], and

Monte Carlo Tree Search [35, 52]. The field gradually gains its tractions with the emergence of highly-efficient algorithms [33, 37, 39] and architectures [40, 47] with remarkable performance.

Our method is inspired by PEARL [38], a recent work in context-based meta reinforcement learning, which captures knowledge about a task with probabilistic latent contexts. The knowledge is then leveraged for informed policy training. There are a few key challenges in efficiently applying it to NAS: (1) PEARL models the latent context embeddings of RL tasks as distributions over Markov Decision Processes (MDP), but it is less clear how a task in NAS can be meaningfully encoded. (2) RL is notoriously famous for its sample inefficiency, but it is extremely expensive to obtain reward signals on NAS. We address these challenges by (1) proposing the use of network-reward pairs to represent a task, (2) introducing meta-training tasks that can be cheaply evaluated to obtain more data for learning, and including a network evaluator that acts like Q-learning agents to speed up learning.

Previous works also explored the possibility of using meta-learning for NAS. Some [23, 36] aimed to identify a single architecture that simultaneously works well on all considered tasks. These solutions may not be scalable when confronting a large pool of target tasks. An early work [53] aimed to learn a general policy across tasks. However, it generates task embeddings from images, which may fail at datasets with the same images, and is unable to differentiate among classification, detection, and segmentation tasks on the same dataset. A few recent papers [28, 15] combined gradient-based meta-learning with DARTS, but the algorithms are only applicable to search spaces compatible with DARTS. Additionally, none of the above proposals reported their performance on large-scale tasks like ImageNet full dataset. This leaves questions on these proposals' generalizability and adaptation efficiency on more challenging datasets, where scientists expect meta-NAS algorithms should have an edge over typical NAS methods. CATCH is the first NAS algorithm to our knowledge that deploys meta-learning while maintaining universality, robustness across different search spaces, and capability to handle large-scale tasks.

3 CATCH Framework

In NAS, the change of dataset (e.g. CIFAR-10 vs. ImageNet) or domain (e.g. image classification vs. object detection) essentially indicates the shift of underlying reward distribution. The goal of a cross-task transfer algorithm is hence to quickly identify the best actions under the changed reward dynamics. To handle this challenge, the CATCH framework consists of two phases: a meta-training phase and an adaptation phase, as is presented in Algorithm 1. In the meta-training phase, we train the CATCHer on a pool of meta-training tasks that can be cheaply evaluated. A key goal of this phase is to present the context encoder with sufficiently diversified tasks, and encourage it to consistently encode meaningful information for different tasks. Meanwhile, both the controller and the evaluator may gain a good initialization for adaptation. In the adaptation

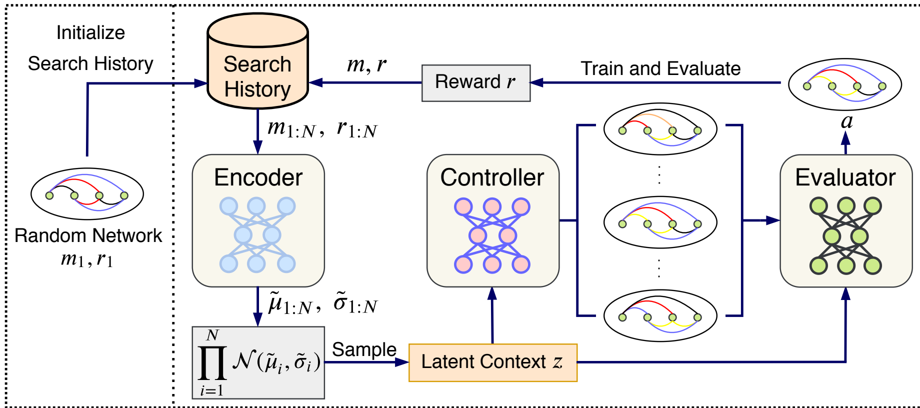


Fig. 2: The search procedure of CATCH on a given task. The procedure starts from initializing the search history by storing a randomly selected network m and its reward r. The encoder applies amortized variational inference approach to generate latent context encoding $\boldsymbol{z}$ by encoding network-reward pairs from the search history. The controller then generates candidate networks for the evaluator to choose the most promising ones to train and evaluate. Newly selected networks and their rewards will be stored in the search history. The loop continues after the three components are optimized.

phase, the meta-trained CATCHer then learns to find networks on the target task efficiently through the guidance of the latent context encoding.

We show the search procedure on any single task in Figure 2, which corresponds to line 3-13 of Algorithm 1.

3.1 Context Encoding

The use of latent context encoding is a crucial part of CATCH. The question is what information about the task is reliable to construct such latent contexts. Directly extracting feature maps of images of the dataset is an intuitive solution. However, for the same dataset, the best network configurations to perform different tasks like object detection and semantic segmentation may differ a lot. Hence, simply extracting information directly from images may not be a viable approach.

We instead believe that the task-specific contextual knowledge can be mined from the search history (i.e. sets of network-reward pairs). If the same group of networks have similar relative strengths on two tasks, it might mean these tasks are “close” to each other. It is also helpful to break the barriers for cross-task architecture search, since the network-reward pair of information is universal across tasks.

Before searching on a task, we randomly form a few networks m and evaluate their performance r to initialize the search history. The retrieved network-reward pairs are stored in the search history for its initialization. To start the search, we sample a number of network-reward pairs $\{(m, r)_i\}_1^N$ (denoted by $\boldsymbol{c}_{1:N}$ for simplicity) from the search history, which will be fed into the encoder to generate a latent context vector $\boldsymbol{z}$ representing the salient knowledge about the task.

We model the latent context encoding process in a probabilistic manner, because it allows the context encoder to model a distribution over tasks and conduct exploration via posterior sampling. Following the amortized variational inference approach used in [38, 1, 24], we aim to estimate the posterior $p(\boldsymbol{z}|\boldsymbol{c}_{1:N})$ with the encoder $q_\phi(\boldsymbol{z}|\boldsymbol{c}_{1:N})$, parametrized by ϕ. We assume the prior $p(\boldsymbol{z})$ is a unit multivariate Gaussian distribution with diagonal covariance matrix $\mathcal{N}(\mathbf{0}, diag(\mathbf{1}))$, and hence, the posterior $p(\boldsymbol{z}|\boldsymbol{c})$ conditioning on $\boldsymbol{c}$ is Gaussian. Since the network-reward pairs $\boldsymbol{c}_{1:N}$ are independent on a task, we could factor $q_\phi(\boldsymbol{z}|\boldsymbol{c}_{1:N})$ into the product of Gaussian factors conditioning on each piece of contexts $\boldsymbol{c}_i$,

$$q_\phi(\boldsymbol{z}|\boldsymbol{c}_{1:N}) \propto \prod_{i=1}^{N} \mathcal{N}(f_\phi^{\tilde{\mu}}(\boldsymbol{c}_i), diag(f_\phi^{\tilde{\sigma}}(\boldsymbol{c}_i))), \tag{1}$$

where f_ϕ is an inference network parametrized by ϕ, which predicts the mean $\tilde{\boldsymbol{\mu}}_i$ and the standard deviation $\tilde{\boldsymbol{\sigma}}_i$ of $q_\phi(\boldsymbol{z}|\boldsymbol{c}_i)$ as a function of $\boldsymbol{c}_i$ to approximate Gaussian $p(\boldsymbol{z}|\boldsymbol{c}_i)$.

During the forward pass, the encoder network f_ϕ outputs $\tilde{\boldsymbol{\mu}}_i$, $\tilde{\boldsymbol{\sigma}}_i$ of the Gaussian posterior $q_\phi(\boldsymbol{z}|\boldsymbol{c}_i)$ conditioning on each context, then we take their product $q_\phi(\boldsymbol{z}|\boldsymbol{c}_{1:N})$. Each context $\boldsymbol{c}_i$ is $(m, r)_i$, where r is normalized among $\{r\}_{1:N}$ to reflect the relative advantage of each network. All the network-reward pairs in the search history are utilized. We then sample $\boldsymbol{z}$ from $q_\phi(\boldsymbol{z}|\boldsymbol{c}_{1:N})$. Further implementation details can be found in the Appendix.

3.2 Network Sampling

The generation of a network can be treated as a decision-making problem, where each of the RL controller's actions determines one attribute of the resulting architecture. The attribute can be an operation type to form a certain edge in a cell-based search (e.g. skip-connect, convolution operations, etc.), or the shape of a network in a macro-skeleton search (e.g. width, depth, etc.). Both ways are explored in our work.

A network, denoted by m, is represented as a list of actions $[a_1, a_2, ..., a_L]$ taken by the controller in a sequential manner. At each time step l, the controller makes a decision a_l according to its policy π_{θ_c}, parametrized by θ_c. The controller policy takes $\boldsymbol{z}$ and the previous actions $[a_1...a_{l-1}, \mathbf{0}, ..., \mathbf{0}]$ as inputs, and outputs the probability distribution of choosing a certain action $\pi_{\theta_c}(a^l|[a_1...a_{l-1}, \mathbf{0}, ..., \mathbf{0}], \boldsymbol{z})$, where the actions will be sampled accordingly. $\boldsymbol{z}$ is the latent context vector generated by the encoder, and $[a_1...a_{l-1}, \mathbf{0}, ..., \mathbf{0}]$ is a collection of one-hot vectors indicating all the actions taken so far at l-th timestep, leaving untaken actions $[a_l, ..., a_L]$ as zero vectors. The reward for each action is the normalized performance score of the network. The controller samples M networks stochastically as candidates for the network evaluator.

Algorithm 1 Context-based Meta Architecture Search (CATCH)

Inputs: $\{\mathcal{T}_{meta}\}$ (meta-training task pool), $\{\mathcal{T}_{target}\}$ (target task pool), N_{meta} (# of meta epochs), N_{search} (# of search epochs), C (# of contexts to sample), M (# of models to sample)

Meta-training Phase:
1: **for** N_{meta} meta epochs **do**
2: Select meta-training task $\mathcal{T}$ from $\{\mathcal{T}_{meta}\}$
3: Initialize SearchHistory
4: **for** $n = 1$ to N_{search} **do**
5: $\{(m, r)_i\}_1^C$ = SearchHistory.sample_contexts(C)
6: $\boldsymbol{z}$ = Encoder.encode($\{(m, r)_i\}_1^C$)
7: $\{m\}_1^M \leftarrow$ Controller.sample_networks($\boldsymbol{z}$, M)
8: $m' \leftarrow$ Evaluator.choose_best($\{m_j\}_1^M$, $\boldsymbol{z}$)
9: $r \leftarrow$ train_and_evaluate($m', \mathcal{T}$)
10: SearchHistory.add($(m', \boldsymbol{z}, r)$)
11: Encoder, Controller, Evaluator optimization
12: **end for**
13: **end for**

Adaptation Phase:
14: Select target task $\mathcal{T}$ from $\{\mathcal{T}_{target}\}$
15: **Repeat** Line 3-13
16: BestModel $\leftarrow$ SearchHistory.best_model()
17: **return** BestModel

3.3 Network Scoring and Evaluation

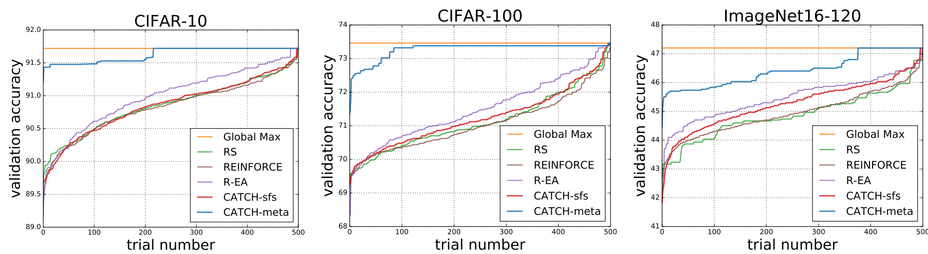
Since the candidate networks are sampled stochastically by the controller, it is almost inevitable that some inferior models will be generated. We set up a filtering mechanism, namely network evaluator, which acts like a Q-learning agent that predicts the actual performance of each network, and selects the top one for training. The predicted value is not necessarily an accurate prediction of the training performance, but should be able to provide a ranking among candidate models roughly similar to their true performance.

The evaluator $f_{\theta_e}(m, \boldsymbol{z})$ is parameterized by θ_e. It takes M tuples of network-context pairs $(m, \boldsymbol{z})$ as inputs, and outputs the predicted performance of input architectures. The network with the highest predicted performance score will be trained to obtain the true reward r. The network-context-reward tuple $(m, \boldsymbol{z}, r)$ is then stored in the evaluator's local memory for future gradient updates.

3.4 Optimization of CATCHer

To optimize the controller policy, we maximize the expected reward for the task it is performed on. The controller is trained using Proximal Policy Optimization (PPO) [43] with a clipped surrogate objective $\mathcal{L}_c$.

To optimize the evaluator, we deploy Prioritized Experience Replay (PER) [42], a Deep Q-learning [34] optimization technique. During the update, it prompts the evaluator to prioritize sampling entries that it makes the most mistakes on,



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet16-120

Fig. 3: (a)-(c) show the results of 500 trials for CATCH-meta, CATCH-sfs(search from scratch) and other sample-based algorithms. Each individual trial is sorted by the final validation accuracy of the searched network.

and thus improves sample efficiency. The loss of the evaluator $\mathcal{L}_e$ is the Huber loss [22] between the evaluator's prediction $\tilde{r}$ and the normalized true performance score. Further details of $\mathcal{L}_c$ and $\mathcal{L}_e$ can be found in the Appendix.

To optimize the encoder, we take $\mathcal{L}_c$ and $\mathcal{L}_e$ as part of the objective. The resulting variational lower bound for each task $\mathcal{T}$ is

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|c^{\mathcal{T}})}[\mathcal{L}_c + \mathcal{L}_e + \beta D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{c}^{\mathcal{T}})||p(\boldsymbol{z}))], \tag{2}$$

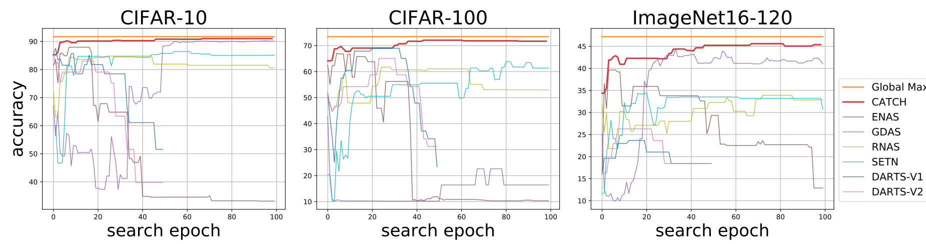
where D_{KL} serves as an approximation to a variational information bottleneck that constrains the mutual information between $\boldsymbol{z}$ and $\boldsymbol{c}$, as is shown in [1, 38]. This information bottleneck acts as a regularizer to avoid overfitting to training tasks. β is the weight of D_{KL} in the objective, and $p(\boldsymbol{z})$ is a unit Gaussian prior. Since (1) the latent context $\boldsymbol{z}$ serves as input to both controller and evaluator, and (2) $q_\phi(\boldsymbol{z}|\boldsymbol{c})$ and $p(\boldsymbol{z})$ are Gaussian, with D_{KL} computed using their mean and variance, gradient of Eq. 2 can be back-propagated end-to-end to the encoder with the reparameterization trick.

4 Experiments

4.1 Implementation Details

We use Multi-layer Perceptrons (MLP) as the controller policy network to generate the probability of choosing a certain action. The parameters θ_c of the controller is trained on-policy via the PPO algorithm. We mask invalid actions by zeroing out their probabilities in the controller's outputs, then softmax the remaining probabilities and sample actions accordingly.

The evaluator is an MLP to generate the predicted score of a network. In the meta-training phase, we reset ϵ in the ϵ-greedy exploration strategy each time when the agent initializes a new task. We sample 80% of the entries as a batch from the replay buffer using PER.



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet16-120

Fig. 4: Learning curves of one-shot algorithms and CATCH. Each curve is an average of three runs. We plot the first 100 search epochs for algorithms except for DARTS, which is trained only for 50 search epochs.

The encoder MLP outputs a 10-dim latent context vector $\boldsymbol{z}$, and the weight of the KL-Divergence β in the combined loss is set to be 0.1. More details of the components' hyperparameters can be found in the Appendix.

4.2 Benchmark on NAS-Bench-201

As recent work [56] indicated, NAS algorithms are usually compared unfairly under different settings. To mitigate such problems, we first tested CATCH on NAS-Bench-201. It is a benchmark dataset that enables fair comparisons among NAS methods under the same configurations. It supports searching over cell-based architectures, where a directed acyclic graph represents each cell with 4 nodes and 5 possible connection operations on each edge. It provides the validation and test accuracies of 15,625 architectures on CIFAR-10, CIFAR-100, and ImageNet16-120 datasets. ImageNet16-120 is a subdataset for ImageNet, which downsampled all its images to 16×16, and contains only the first 120 classes of ImageNet.

Experiment Settings. In the meta-training phase, each task is formed as a classification task on an X-class sub-dataset of ImageNet16 (ImageNet downsampled to 16×16) to maintain consistency with the configurations in NAS-Bench-201. The number of classes $X \in [10, 20, 30]$. In each meta-epoch, the agent searches 20 networks whose validation accuracies after 12 training epochs are used as the reward signals. The hyperparameters used for training the networks in both phases are identical to those in NAS-Bench-201. In the following experiments, CATCH-meta is meta-trained with 25 meta epochs for 10.5 GPU hours on Tesla V100. We apply the same configurations as those in NAS-Bench-201.

Comparison with Sample-based Algorithms. We display the search results of the meta-trained version (CATCH-meta) and the search-from-scratch

Table 1: Comparison of CATCH with one-shot algorithms. The top accuracies of identified models, standard deviations, search time (hour), total search time (hour), and the highest validation accuracies among all the networks in NAS-Bench-201 are reported. The same three random seeds are used to run through each algorithm. The time budget for search on CIFAR-10, CIFAR-100, and ImageNet16-120 are 3, 4, and 5 hours respectively.

Algorithm	CIFAR-10		CIFAR-100		ImageNet16-120		Total Time
	Acc ±std	Time	Acc±std	Time	Acc±std	Time	
DARTS-V1 [33]	88.08±1.89	2.46	68.99±1.93	2.44	23.66±0	4.55	9.45
DARTS-V2 [33]	87.16±0.39	9	65.06±2.95	7.91	26.29±0	22.14	39.05
GDAS [12]	90.32±0.08	6	70.33±0.85	6.23	44.81±0.97	17	29.23
R-NAS [26]	90.45±0.43	2.19	70.39±1.36	2.26	44.12±1.04	5.94	10.39
ENAS [37]	90.2±0.63	4.22	69.99±1.03	4.26	44.92±0.51	5.18	13.66
SETN [11]	90.26±0.75	7.62	68.01±0.21	7.74	41.04±1.64	20.33	35.69
CATCH-meta	**91.33±0.07**	**3**	**72.57±0.81**	**4**	**46.07±0.6**	**5**	22.5
Max Acc.	91.719		73.45		47.19		—

version (CATCH-sfs where the meta-training phase is skipped) of our method, and compare them with other sample-based algorithms: Random Search (RS) [2], Regularized Evolution Algorithm (R-EA) [40], and REINFORCE [51]. The results of other methods are reproduced by running the code and configurations originally provided by NAS-bench-201. Each experiment is repeated for 500 trials with different seeds. The algorithms are trained for 50 search epochs in each trial. Figure 3 presents the search results on CIFAR-10, CIFAR-100, ImageNet16-120, with the highest validation accuracy on each task.

The reproduced results are consistent with the experiments performed in NAS-Bench-201. The performance of CATCH-sfs is similar to the other four methods, but CATCH-meta dominates all other algorithms in the searched network accuracies. On CIFAR-10, CATCH-meta finds the best model in 280/500 trials. On CIFAR-100, over half of them find top-3 performance networks within 50 samples, while other algorithms barely touch the roof. On ImageNet16-120, CATCH reaches the best network for more than 22% trials. We can see tremendous benefits for using the meta-trained CATCH to reduce time and cost.

Comparison with One-shot Algorithms. One of the central controversies around meta-NAS algorithms is: given the high searching efficiency of one-shot methods, can sample-based algorithms outperform them? We therefore compare the performance of CATCH with many state-of-the-art one-shot NAS solutions. For fair comparisons, instead of querying the NAS-Bench-201 network database, we train each child network for 12 epochs and obtain their early-stop validation accuracies as training feedbacks. The early-stop training setup is the same as the one in the meta-training phase. The one-shot algorithms involved are first-order DARTS (DARTS-V1) [33], second-order DARTS (DARTS-V2), GDAS [12], Random NAS (R-NAS) [26], ENAS [37], and SETN [11]. We run the algorithms with

Table 2: Results on ImageNet compared to manually designed and NAS searched architectures. Latency is measured on one Tesla V100 with one image with shape (3, 720, 1080).

Network	Top-1 Acc (%)	Top-5 Acc (%)	Latency (ms)
ResNet50 [19]	77.15	93.29	16.4
DenseNet201 [20]	77.42	93.66	31.6
ResNext101 [54]	79.31	94.5	76.7
Inception-V3 [45]	78.8	94.4	16.4
EfficientNet-B1 [47]	77.3	93.5	29.5
EfficientNet-B2	79.2	94.5	47.6
NASNet-A [61]	78.6	94.2	-
BASE [44]	74.3	91.9	-
CATCH-Net-A	79.04	94.43	**16.9**
CATCH-Net-B	**79.46**	**94.7**	33.7

the original code and configurations released from NAS-Bench-201. DARTS-V1 and DARTS-V2 are run for 50 search epochs, and other algorithms are trained for 250 search epochs.

Figure 4 presents the learning curves of each algorithm in the first 100 search epochs. For CATCH, at each search epoch, we identify networks with the best partially trained accuracy found so far, and report their fully trained accuracies. Both DARTS and ENAS have a relatively strong performance at the beginning, but the curves drop significantly afterward. SETN resembles Random NAS a lot. GDAS is among the best one-shot algorithms, but it seems to plateau at local maximums after a few search epochs. CATCH has the best performance among all, as it quickly adapts and identifies promising architectures that are beyond other algorithms' search capacity.

In Table 1, we report the best fully trained accuracy of networks that each algorithm identifies over their complete training process. We set the time budget for CATCH to search on CIFAR-10, CIFAR-100, and ImageNet16-120 as 3, 4, and 5 hours. It is roughly equivalent to cutting the search on these tasks at 70, 50, and 40 search epochs, respectively. Although DARTS-V1, R-NAS, and ENAS spend less time in total, they are highly unstable and the performance of DARTS and ENAS tends to deteriorate over time. CATCH spends 22.5 (10.5 meta + 12 adaptation) hours on all three tasks, and its searched networks surpass all other algorithms. The presented results have proved that CATCH is swiftly adaptive, and it is able to identify networks beyond many one-shot algorithms' reach within a reasonable time.

4.3 Experiments on Residual Block-based Search Space

Having proved that CATCH can adapt to new tasks efficiently with meta-training, we further inquire whether CATCH has the ability to transfer across different domains including image classification, objection detection, and semantic segmentation. In this section, we consider a more challenging setting where

Table 3: Results on COCO compared to manually designed and NAS searched backbones. Latency results of networks except CATCH are referred from [57].

Method	Backbone	Input size	Latency (ms)	mAP
RetinaNet [29]	ResNet101-FPN	1333x800	91.7 (V100)	39.1
FSAF [59]	ResNet101-FPN	1333x800	92.5 (V100)	40.9
GA-Faster RCNN [48]	ResNet50-FPN	1333x800	104.2 (V100)	39.8
Faster-RCNN [41]	ResNet101-FPN	1333x800	84.0 (V100)	39.4
Mask-RCNN [18]	ResNet101-FPN	1333x800	105.0 (V100)	40.2
DetNAS [8]	Searched Backbone	1333x800	-	42.0
SM-NAS: E3	Searched Backbone	800x600	50.7(V100)	42.8
SM-NAS: E5	Searched Backbone	1333x800	108.1(V100)	45.9
Auto-FPN [55]	Searched Backbone	1333x800	-	40.5
CATCH	CATCH-Net-C	1333x800	123.5 (V100)	**43.2**

the meta-training phase contain only image classification tasks while tasks in all the three domains are targeted in the adaptation phase. The architectures are very different among these domains, so we search for their common component - the feature extractor (backbone). ResNet is one popular backbone for these tasks, thus we design the search space following [49, 57].

Constructing a model in the Residual block-based search space requires the controller to make several decisions: (1) select the network's base channel from $[48, 56, 64, 72]$, (2) decide the network's depth within $[15, 20, 25, 30]$, (3) choose the number of stages s, which is either 4 or 5, (4) schedule the number of blocks contained in each stage, and (5) arrange the distribution of blocks holding different channels. Details of the Residual block-based search space can be found in the Appendix.

Experiment Settings. We use the same meta-training settings as the ones we used in NAS-Bench-201. For each meta epoch, an ImageNet sub-dataset is created. To form such sub-datasets, we sample X classes from all classes of ImageNet, where $X \in [10, 20, 30]$. Then the images are resize to 16×16, 32×32, or 224×224. Thus there are $3 \times \left[\binom{1000}{10} + \binom{1000}{20} + \binom{1000}{30}\right]$ possible sub-datasets.

To achieve the balance between inference latency and network performance, we adopt the multi-objectve reward function $R = P(m) \times [\frac{LAT(m)}{T_{target}}]^w$ in [46], where $P(m)$ denotes the model 's performance (e.g. validation accuracy for classification, mAP for object detection or mIoU for semantic segmentation), $LAT(m)$ measures the model's inference latency, and T_{target} is the target latency. w serves as a hyperparameter adjusting the performance-latency tradeoff. In our experiments, we set $w = -0.05$. With this reward, we hope to find models that excel not only in performance but also in inference speed. We meta train CATCHer for 5 GPU days, and adapt on each target task to search for 10 architectures. We target ImageNet dataset for image classification, COCO dataset for object detection and Cityscapes dataset for semantic segmentation. The detailed settings can be found in the Appendix.

Table 4: Results on Cityscapes compared to manually designed and NAS searched backbones. Latency is measured on Tesla V100 with one image with shape (3, 1024, 1024). SS and MS denote for single scale and multiple scale testing respectively.

Method	Backbone	Latency (ms)	mIoU (SS)	mIoU (MS)
BiSeNet [58]	ResNet101	41	-	80.3
DeepLabv3+ [7]	Xception-65	85	77.82	79.3
CCNet [21]	ResNet50	175	-	78.5
DUC [50]	ResNet152	-	76.7	-
DANet [17]	ResNet50	-	76.34	-
Auto-DeepLab [32]	Searched Backbone	-	79.94	-
DPC [6]	Xception-71	-	80.1	-
CATCH	CATCH-Net-D	**27**	79.52	**81.12**

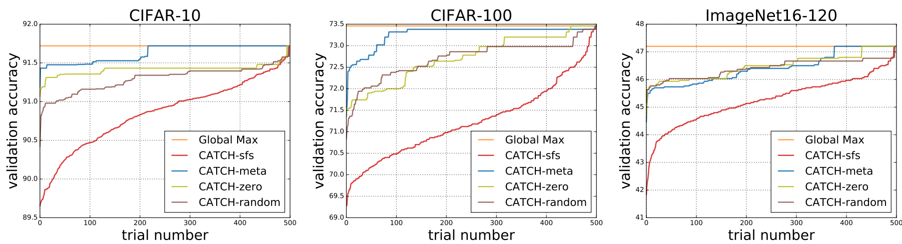
Search Results. Table 2 compares the searched architectures with other widely-recognized networks on ImageNet. CATCH-Net-A outperforms many listed networks. Its accuracy is comparable with EfficientNet-B1 and ResNext-101, yet it is 2.82X and 4.54X faster. CATCH-Net-B outperforms ResNext-101 while shortens the latency by 2.28X. The network comparison on COCO and Cityscapes is presented in Table 3 and Table 4. Our network again shows faster inference time and competitive performance. We also transfer CATCH-Net-B found during the search on ImageNet to COCO and Cityscapes, which yield 42% mAP with 136ms inference time and 80.87% mIoU (MS) with 52ms latency, respectively. Our results again show that directly transferring top architectures from one task to another cannot guarantee optimality. It also reveals CATCH's potentials to transfer across tasks even when they are distant from the meta-training ones.

5 Ablation Study

The context encoder is the spotlight component of our algorithm. We are especially curious about: (1) Is the encoder actually helpful for adaptation (compared with simply plugging in the meta-learned controller and evaluator priors)? (2) If so, does the improvement come from good estimates of the posterior, or is it from the stochastic generation of $\boldsymbol{z}$ that encourages exploration and benefits generalization?

To answer these questions, we designed two extra sets of experiments: (1) CATCH-zero: We set $\boldsymbol{z} = \mathbf{0}$, and thereby completely eliminate the encoder's effect on both the controller and the evaluator; (2) CATCH-random: We sample each $\boldsymbol{z}$ from a unit Gaussian prior $\mathcal{N}(\mathbf{0}, diag(\mathbf{1}))$ during the search as random inputs. The results are presented in Figure 5 (a)-(c). In both settings, the agents are still meta-trained for 10.5 hours before they are plugged in for adaptation.

The gaps among the lines in Figure 5 answered our questions. The encoder not only helps with adaptation (through comparing CATCH-meta and CATCH-zero), but also provides assistance in a much more meaningful way than using



(a) (b) (c)

Fig. 5: (a)-(c) compare results of 500 trials for CATCH-meta, CATCH-sfs(search from scratch), CATCH-zero, CATCH-random.

random inputs for exploration, as CATCH-meta outperforms CATCH-random on both CIFAR-10 and CIFAR-100. Interestingly, we observe less significant improvement on ImageNet16-120. One hypothesis is since we perform the meta-training phase on sub-datasets of ImageNet16, the meta-trained controller and evaluator are already tuned towards policies that fit the search on ImageNet16. Hence, the transferred policies require less adaptation assistance from the encoder. More ablation studies can be found in the Appendix.

6 Conclusion and Discussion

In this work, we propose CATCH, a transferrable NAS approach, by designing an efficient learning framework that leverages the benefits of context-based meta reinforcement learning. The key contribution of CATCH is to boost NAS efficiency by extracting and utilizing task-specific latent contexts, while maintaining universality and robustness in various settings. Experiments and ablation studies show its dominant position in search efficiency and performance over non-transferrable schemes on NAS-Bench-201. Extensive experiments on residual block-based search space also demonstrate its capability in handling cross-task architecture search. As a task-agnostic transferrable NAS framework, CATCH possesses great potentials in scaling NAS to large datasets and various domains efficiently.

During our research into transferrable NAS frameworks, we identified many potentially valuable questions to be explored. Efficient adaptation among domains is challenging, and we demonstrated a first attempt to simplify it by searching for backbones with a shared search space. A possible future investigation would be to generalize cross-task architecture search to flexibly include more decisions, such as searching for detection and segmentation heads. Meanwhile, our meta-training tasks involve only classification tasks, but it is also possible to diversify the pool and explore whether it leads to further performance boosts.

References

1. Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.
2. James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.
3. James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123, 2013.
4. James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.
5. Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, pages 8699–8710, 2018.
6. Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8699–8710. Curran Associates, Inc., 2018.
7. Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *The European Conference on Computer Vision (ECCV)*, September 2018.
8. Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. Detnas: Neural architecture search on object detection. *arXiv preprint arXiv:1903.10979*, 2019.
9. Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
10. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
11. Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3681–3690, 2019.
12. Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.
13. Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.
14. Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.
15. Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. *arXiv preprint arXiv:1911.11090*, 2019.

16. Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.
17. Jun Fu, Jing Liu, Haijie Tian, Yong Li, Yongjun Bao, Zhiwei Fang, and Hanqing Lu. Dual attention network for scene segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
18. Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.
19. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
20. Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
21. Zilong Huang, Xinggang Wang, Lichao Huang, Chang Huang, Yunchao Wei, and Wenyu Liu. Ccnet: Criss-cross attention for semantic segmentation. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.
22. Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.
23. Jaehong Kim, Sangyeul Lee, Sungwan Kim, Moonsu Cha, Jung Kwon Lee, Youngduck Choi, Yongseok Choi, Dong-Yeon Cho, and Jiwon Kim. Auto-meta: Automated gradient based meta learner search. *arXiv preprint arXiv:1806.06927*, 2018.
24. Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
25. Lin Lan, Zhenguo Li, Xiaohong Guan, and Pinghui Wang. Meta reinforcement learning with task embedding and shared policy. *arXiv preprint arXiv:1905.06527*, 2019.
26. Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.
27. Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.
28. Dongze Lian, Yin Zheng, Yintao Xu, Yanxiong Lu, Leyu Lin, Peilin Zhao, Junzhou Huang, and Shenghua Gao. Towards fast adaptation of neural architectures with meta learning. In *International Conference on Learning Representations*, 2020.
29. Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.
30. Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and Larry Zitnick. Microsoft coco: Common objects in context. In *ECCV*. European Conference on Computer Vision, September 2014.
31. Bin Liu, Chenxu Zhu, Guilin Li, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li, and Yong Yu. Autofis: Automatic feature interaction selection in factorization models for click-through rate prediction. *arXiv preprint arXiv:2003.11235*, 2020.
32. Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019.
33. Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

34. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
35. Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.
36. Ramakanth Pasunuru and Mohit Bansal. Continual and multi-task architecture search. *arXiv preprint arXiv:1906.05226*, 2019.
37. Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.
38. Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv preprint arXiv:1903.08254*, 2019.
39. E Real, A Aggarwal, Y Huang, and QV Le. Aging evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, 2019.
40. Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.
41. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
42. Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.
43. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
44. Albert Shaw, Wei Wei, Weiyang Liu, Le Song, and Bo Dai. Meta architecture search. In *Advances in Neural Information Processing Systems*, pages 11225–11235, 2019.
45. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
46. Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
47. Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.
48. Jiaqi Wang, Kai Chen, Shuo Yang, Chen Change Loy, and Dahua Lin. Region proposal by guided anchoring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2965–2974, 2019.
49. Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, and Chunhua Shen. NAS-FCOS: fast neural architecture search for object detection. *CoRR*, abs/1906.04423, 2019.
50. P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell. Understanding convolution for semantic segmentation. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1451–1460, March 2018.
51. Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.
52. Martin Wistuba. Finding competitive network architectures within a day using uct. *arXiv preprint arXiv:1712.07420*, 2017.

53. Catherine Wong, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo. Transfer learning with neural automl. In *Advances in Neural Information Processing Systems*, pages 8356–8365, 2018.
54. Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.
55. Hang Xu, Lewei Yao, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Auto-fpn: Automatic network architecture adaptation for object detection beyond classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6649–6658, 2019.
56. Antoine Yang, Pedro M. EsperanÃ§a, and Fabio M. Carlucci. Nas evaluation is frustratingly hard. In *International Conference on Learning Representations*, 2020.
57. Lewei Yao, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Sm-nas: Structural-to-modular neural architecture search for object detection. *arXiv preprint arXiv:1911.09929*, 2019.
58. Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *The European Conference on Computer Vision (ECCV)*, September 2018.
59. Chenchen Zhu, Yihui He, and Marios Savvides. Feature selective anchor-free module for single-shot object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 840–849, 2019.
60. Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.
61. Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.