Appendix for 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

1 Learning Curve Comparison with Sample-based Algorithms

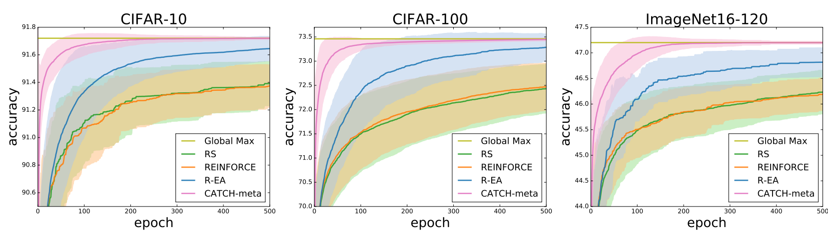


Fig. 1: Comparison of CATCH with other sample-based algorithms on CIFAR-10 [6], CIFAR-100 [6], and ImageNet16-120 [3].

We compare the learning curve of CATCH with other sample-based algorithms in Figure 1. We plot each curve with the highest fully-train validation accuracy the agent has seen at each search epoch. Each curve is plotted with an average of 500 trials. The shaded area shows the mean ± standard deviation among all trials at each search epoch. CATCH stands out among others with higher performance and lower variation on all three datasets (CIFAR-10, CIFAR-100, and ImageNet16-120). It is also on average a magnitude faster than other algorithms to find their best architectures after 500 searching epochs. On ImageNet16-120, none of the algorithms except CATCH could even identify the best architecture within 500 searching epochs across all 500 trials. CATCH is also more stable, as is indicated by its much lower variation compared with other algorithms. Its

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

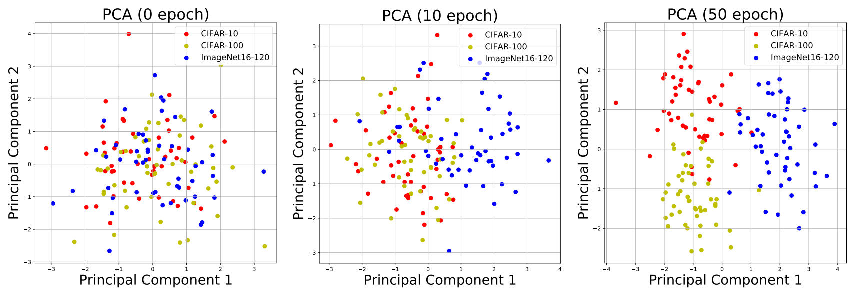


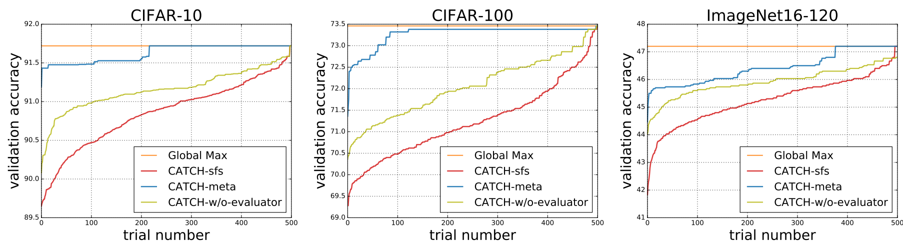
Fig. 2: The encoder's adaptation process. It learns to distinguish different datasets throughout the learning process, and thus provide informed input to the controller and the evaluator.

variance tends to shrink over time, while R-EA and REINFORCE policies are almost as unstable as random search. Through this comparison, we further prove the adaptation speed and stability of CATCH, along with its competency across various datasets and random seeds.

2 Encoder's Adaptation Result

Throughout the adaptation process, we hypothesize that the encoder can provide dataset-specific guidance to the controller and the evaluator. To test this hypothesis, we visualize the encoded latent context vector z of each dataset through Principle Component Analysis, with the results presented in Figure 2. Each point is generated by randomly selecting and encoding 80% network-reward pairs from the search history. We freeze the weights of the meta-trained controller and evaluator policy, and only allow gradient updates for the encoder. This operation eliminates influence from the changing controller and evaluator policies, and thus enables us to closely observe just the behaviors of the encoder. When the encoder is first adapted to CIFAR-10, CIFAR-100, and ImageNet16-120, the generated context vectors are not distinguishable across the three datasets. However, after just 10 search epochs of adaptation, we can already identify a cluster of ImageNet16-120 context vectors. The clusters then quickly evolve as the encoder sees more architectures. By the 50-th search epoch, we can see three distinctive clusters as a result of the encoder's fast adaptation towards the three datasets.

This observation is consistent with the results of NAS-Bench-201 [3]. In the original paper, the network-performance pairs have higher correlation between CIFAR-10 and CIFAR-100 (0.968) than that between CIFAR-10 and ImageNet16-120 (0.827). This correlation is also higher than the correlation between CIFAR-100 and ImageNet16-120 (0.91). This attributes to the reason why the encoder takes more search epochs to distinguish CIFAR-10 from CIFAR-100. The results are in support of our hypothesis, and show the encoder's capability to learn and express dataset-specific information effectively.



(a) (b) (c)

Fig. 3: Comparison of CATCH-meta, CATCH-sfs with CATCH-without-evaluator. Including the evaluator significantly raises the performance.

3 Ablation Study on the Evaluator

We also explored the effects of the evaluator by eliminating it from both the meta-training and adaptation phase, and its performance is presented in Figure 3 (a)-(c). As the figure shows, the evaluator lifts the performance by a large margin, making it a crucial component in the search algorithm. Table 1 provides further information on the evaluator when comparing it with CATCH using ground truth as the evaluator (CATCH-GT). CATCH-GT is a hard-to-defeat baseline, but CATCH-meta managed to get very close to it and the global max accuracy.

Table 1: Comparison of CATCH when using ground truth as the evaluator (CATCH-GT), CATCH without evaluator (CATCH-w/o-evaluator), and CATCH-meta. The results are taken from 100 trials where each trail contains 50 search epochs. We report the mean ± std for each setting in the table.

	CIFAR-10	CIFAR-100	ImageNet16-120
CATCH-GT	91.64±0.09	73.31±0.16	47.18±0.09
CATCH-w/o-evaluator	91.17±0.25	72.08±0.68	45.86±0.54
CATCH-meta	91.63±0.11	73.29±0.31	46.37±0.53
Max Acc.	91.719	73.45	47.19

4 CATCHer Training Details

4.1 Controller Settings and Hyperparameters

The controller is trained with Proximal Policy Optimization (PPO) [10] algorithm, and its loss $\mathcal{L}_c$ is defined following the original PPO loss:

$$\mathcal{L}_c = \hat{\mathbb{E}}_t \left[min \left(r_t\left(\theta_c\right) \hat{A}_t, clip\left(r_t\left(\theta_c\right), 1-\epsilon, 1+\epsilon\right) \hat{A}_t \right) \right]$$

ϵ is the PPO clipping parameter, $r_t(\theta_c) = \frac{\pi_{\theta_c}(a_l|s_t)}{\pi_{\theta_{old}}(a_l|s_t)}$ is the probability ratio, and $\hat{A}_t$ is the General Advantage Estimate (GAE) [9] estimate:

$$\hat{A}_t = \sum_{l=0}^{t} (\gamma\lambda)^l \delta_l^V$$

where $\delta_l^V = r_t + \gamma V(s_{l+1}) - V(s_l)$ is the Bellman residual term. The definition of s_l can be found in Table 3. We show the training hyperparameters and our settings on translating architecture search elements as Markov Decision Processes (MDP) in the following tables.

Table 2: Controller hyperparameters

Hyperparameter	Value (meta-train)	NAS-Bench-201 [3] (adaptation)	Residual Block Search Space (adaptation)
Learning rate	0.001	0.001	0.0001
Adam scheduler step size	20	20	20
Adam scheduler gamma	0.99	0.99	0.99
Update frequency	1 epoch	1 epoch	1 epoch
Clipping parameter ϵ	0.2	0.2	0.2
Memory size	200	200	200
Discount γ	0.99	0.99	0.99
GAE parameter λ	0.95	0.95	0.95
Value Function coeff.	1	1	1
Entropy coeff.	0.01	0.03	0.05

Table 3: A mapping of Neural Architecture Search elements to MDP factors for controller training. l denotes the current timestep. Invalid actions are masked by zeroing out their probabilities in the outputs, then softmax the remaining probabilities and sample accordingly.

MDP Factor	Value	Explanation
Current state s_l	$(z, [a_1...a_{l-1}])$	Latent context and the current network design.
Current action a	a^l	A one-hot vector of the current design choice.
Reward r	R	A function of the evaluated network's performance.
Next state s_{l+1}	$(z, [a_1...a_l])$	Latent context and the current network design.

4.2 Encoder and Evaluator Settings

The encoder generates the latent conext through the network-reward information (m, r). This is done by taking the encoder output as the means and variances

Algorithm 1 Pseudocode of Latent Context Encoding Procedure in a PyTorch-like style.

```
def encode_z(B, D, Contexts, Encoder):
    # Contexts: a batch of contexts {(m, r)} use for encoding
    # B: len(Contexts), batch
    # D: the dimension of latent context variable z
    # Encoder: 3-layer MLP mapping (m, r) to (mean, var) of z_i

    # encode each (m, r) to (mean, var) of z
    context_batch.rewards = normalize(context_batch.rewards)
    params = Encoder.forward(context_batch) # shape: [B, 2*D]

    # get mean and var; t(): matrix transpose
    means = params[..., :D].t() # shape: [D, B]
    vars = F.softplus(params[..., D:].t()) # shape: [D, B]

    # get mean & var of each z_i; ds: torch.distributions
    posteriors = []
    for ms, vs in zip(unbind(means), unbind(vars)):
        z_i_mean, z_i_var = _product_of_gaussian(ms, vs)
        # form a Gaussian Posterior from z_i_mean, sqrt(z_i_var)
        z_i_posterior = ds.Gaussian(z_i_mean, sqrt(z_i_var))
        posteriors.append(z_i_posterior)

    # sample z from q(z|Contexts); rsample(): random sample
    z = [d.rsample() for d in posteriors]
    return torch.stack(z)
```

of a D-dimensional Gaussian distribution, from which we sample $\boldsymbol{z}$. We provide pseudocode for this process in Algorithm 1.

The evaluator uses the Huber loss [5] to close the gap between its predicted network performance $\tilde{r}$ and the actual performance r.

$$\mathcal{L}_e = \frac{1}{n}\sum_i loss(r_i, \tilde{r_i}), \text{where } loss(r, \tilde{r}) = \begin{cases} 0.5(r_i - \tilde{r}_i)^2 & if \mid r_i - \tilde{r}_i \mid < 1, \\ \mid r_i - \tilde{r}_i \mid -0.5 & otherwise. \end{cases} \tag{1}$$

Table 4: Encoder hyperparameters

Hyperparameter	Value
Learning rate	0.01
Dimension of z	10
KL weight β	0.1

Table 5: Evaluator hyperparameters

Hyperparameter	Value (meta-train)	Value (adaptation)
Learning rate	0.0001	0.0001
Exploration factor ϵ initial value	1.0	0.5
Exploration factor ϵ decay rate	0.025	0.025
Exploration factor ϵ decay step	20	20
Number of networks evaluated per epoch	25	25
PER [8] prioritization factor α	0.5	0.5
PER bias correction factor β	0.575	0.575
PER β annealing step size	0.01	0.01

5 ImageNet, COCO, and Cityscapes Training Settings

Table 6-8 shows our training configurations on ImageNet [2] , COCO [7], and Cityscapes [1] . On COCO, Faster R-CNN with the ResNet backbone and Cascade FPN is used as our baseline. It is extremely costly to perform ImageNet pretrain for search, but training detection networks without ImageNet pretrain was made possible by [4]. For COCO and Cityscapes, we use Group Normalization with halved-base-channel groups instead of Batch Normalization. Conv2D with weight standardization (ConvWS2D) is also applied.

Table 6: ImageNet training hyperparameters with 8 GPUs.

Hyperparameter	Value (partial-train)	Value (fully-train)
Learning rate	0.1	0.1
Learning rate momentum	0.9	0.9
Weight decay	1×10^{-3}	4×10^{-5}
Learning rate warmup	linear for 3 epochs	linear for 3 epochs
Learning rate decay policy	cosine	cosine
Total epoch	40	240
Batch size	1024	512

Table 7: COCO training hyperparameters with 8 GPUs.

Hyperparameters	Value (partial-train)	Value (fully-train)
Normalization	Group Normalization	Batch Normalization
Batch size	16	16
Learning rate	0.18	0.02
Learning rate momentum	0.9	0.9
Weight decay	0.0001	0.0001
Learning rate decay policy	cosine	step
Total epoch	9	24

Table 8: Cityscapes training hyperparameters with 8 GPUs.

Hyperparameters	Value (partial-train)	Value (fully-train)
Baseline model	BiSeNet [13]	BiSeNet
Convolution	ConvWS2D	Conv2D
Normalization	Group Normalization	Synchronized BN
Batch size	32	16
Learning rate	0.02	0.025
Learning rate momentum	0.9	0.9
Weight decay	5×10^{-4}	1×10^{-4}
Learning rate warmup	linear for 5 epochs	linear for 5 epochs
Learning rate decay policy	cosine	polynomial
Total epoch	40	100

6 Searched Models of Residual Block Search Space

We show an example model in our Residual Block search space in Figure 3. It consists of 5 stages, with depth=15, stage distribution=[3,3,4,5], and channel distribution=[2,2,4,7]. We use the same notation format to show the searched models in Table 9.

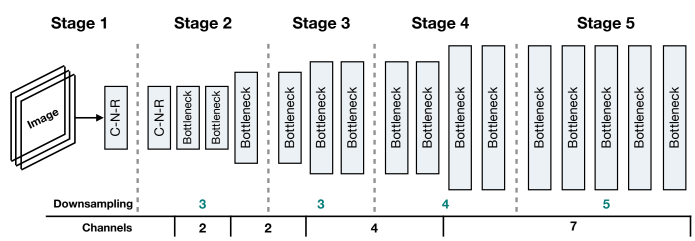


Fig. 4: An example model in the Residual Block search space following [12, 11]. C-N-R stands for a combination of Convolution layer, Normalization layer, and a ReLU operation.

Table 9: Searched models in Residual Block search space.

Searched Model	Input Channel	Depth	Stage Distribution	Channel Distribution	FLOPS(G)	Params(MB)
CATCH-Net-A	64	20	[2, 7, 8, 3]	[5, 4, 8, 3]	4.45	25.96
CATCH-Net-B	64	25	[8, 5, 8, 4]	[3, 10, 8, 4]	9.84	32.16
CATCH-Net-C	64	20	[5, 4, 5, 6]	[1, 8, 5, 6]	8.08	37.03
CATCH-Net-D	64	20	[1, 8, 5, 6]	[2, 7, 7, 4]	4.46	30.98

References

1. 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.
2. 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.
3. Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.
4. Kaiming He, Ross B. Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *CoRR*, abs/1811.08883, 2018.
5. Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.
6. Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
7. 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.
8. Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.
9. John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.
10. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
11. 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.
12. 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.
13. 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.