Supplementary Materials for "AdaXpert: Adapting Neural Architecture for Growing Data"

Shuaicheng Niu [*1 2] **Jiaxiang Wu** [*3] **Guanghui Xu** [1] **Yifan Zhang** [4]
Yong Guo [1] **Peilin Zhao** [3] **Peng Wang** [5] **Mingkui Tan** [1 6]

In the supplementary, we provide more implementation details and more experimental results of our proposed AdaXpert. We organize the supplementary material as follows.

- In Section A, we provide more implementation details of our proposed AdaXpert.
- In Section B, we provide the evaluation details for all compared network architectures.
- In Section C, we provide more experimental results for the threshold selection of our adaptation condition.
- In Section D, we demonstrate the effectiveness of Wasserstein distance on measuring the data difference.
- In Section E, we provide the sensitivity analysis w.r.t. the trade-off parameter λ (in Eqn. (5)).

A. Implementation Details of AdaXpert

During the whole data growth process, we maintain a supernet $\mathcal{N}_t$ and a controller $\pi(\cdot;\theta_t)$. For each time data growth, we first fine-tune the previous supernet on current data $\mathcal{D}_t$ to obtain $\mathcal{N}_t$. After that, the controller $\pi(\cdot;\theta_t)$ is trained by using the evaluation signals provided by the supernet $\mathcal{N}_t$. We introduce the training details of them as follows.

Supernet: For growing Scenario I (in the main paper), we update the supernet 180 epochs for the 0.2 and 0.4 size of ImageNet-100, and we update the supernet 60 epochs for the 0.8 and 1.0 size of ImageNet-100. For growing Scenario II, we update the supernet 180 epochs for ImageNet-20 and ImageNet-40, and 60 epochs for others. Following (Guo et al., 2020), we train the supernet by uniformly sampling sufficient architectures and train them sequentially. For each data growth, the supernet is fine-tuned with a learning rate of 0.045, a weight decay of 5×10^{-5} and a momentum of 0.9.

Controller: The controller takes the previous architecture α_{t-1} and the difference extent d_t (in Eqn. (1)) between data as inputs, and then outputs an adjusted architecture α'_t. In the following, we first introduce the architecture design of the controller and then its training details.

For the controller design, we first use a two-layer fully connected network (FCN) to extract features of the input architecture. Meanwhile, to represent the extent of data difference, following (Pham et al., 2018), we build a learnable embedding vector for different d_t. We then concatenate the architecture embedding and data-difference embedding, and send them to the controller model. We adopt an LSTM to build the controller model. Since the architecture can be represented by a sequence of tokens (Zoph & Le, 2017; Pham et al., 2018), the controller is able to adjust the network architecture by sequentially predicting the token sequences, including depth, width, and kernel size. Here, we incorporate FCN parameters and the learnable embedding vectors into the parameters of the controller and train them jointly.

For each time the growth of data, we train the controller model for 6k iterations. We use Adam with a learning rate of 2×10^{-4} and a weight decay of 5×10^{-4} as the optimizer. We also add the controller's sample entropy to the reward, which

[*]Equal contribution. This work is done when Shuaicheng Niu works as an intern in Tencent AI Lab. [1]School of Software Engineering, South China University of Technology, China [2]Key Laboratory of Big Data and Intelligent Robot, Ministry of Education, China [3]Tencent AI Lab, China [4]National University of Singapore, Singapore [5]Northwestern Polytechnical University, China [6]Pazhou Laboratory, China. Correspondence to: Mingkui Tan <mingkuitan@scut.edu.cn>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

is weighted by 2×10^{-4}. The trade-off parameter λ in Eqn. (5) is set to 0.5×10^{-4} and 2.5×10^{-4} for Scenarios I and II, respectively. Here, the value of λ is only tuned for the first adjustment (e.g., on ImageNet-20) and then fixed for all the subsequent adjustments. In this way, although the λ for subsequent adjustments may not be optimal, the experimental results show that this has already achieved promising performance. We believe that a careful and efficient tuning of λ may further improve the performance of AdaXpert, which we leave to our future work.

B. Details of Architecture Evaluation

Evaluation on "subsets" of ImageNet-1000. For fair comparisons, we train all architectures (including our AdaXpert) on the subset of ImageNet-1000 via the same setting and then test them on the corresponding test set. To be specific, we train each architecture for 180 epochs with a batch size of 256. We apply an SGD optimizer with a weight decay of 5×10^{-5} and a momentum of 0.9. Moreover, the learning rate starts with 0.1 and is divided by 10 at the 80 and 130 epoch. All architectures are evaluated using Tesla V100 GPUs.

Evaluation on "entire" ImageNet-1000. We report the performance of all compared methods on ImageNet-1000 according to their original papers. For AdaXpert models, we evaluate them using the evaluation methods provided by (Lu et al., 2020). Specifically, to accelerate the evaluation, we initialize our model weights with a pre-trained and publicly available once-for-all network (Cai et al., 2020) and then fine-tune it for 85 epochs. The fine-tune training adopts an RMSProp optimizer with a decay of 0.9 and momentum of 0.9. We set the batch normalization momentum to 0.99 and weight decay to 1e-5. We use a batch size of 512 and an initial learning rate of 0.012 that gradually reduces to zero via the cosine annealing schedule. The regularization settings are similar as in (Cai et al., 2018): we use augmentation policy (Cubuk et al., 2020), drop connect ratio 0.2, and dropout ratio 0.2.

C. Threshold Selection for Adaptation Condition

In this section, we show more results on our adaptation condition (in Eqn. (4)) to help algorithm engineers to choose a suitable threshold ϵ for determining the necessity of architecture adjustment. Same to the main paper, experiments are conducted on two considered scenarios, i.e., increasing data volume (**Scenario I**) and the number of classes (**Scenario II**).

For Scenario I, we report the value of H_t (Eqn. (4)) of ResNet18, MobileNetV2 and our AdaXpert that are well-trained on the 0.2 training set of ImageNet-100. The H_t is then computed between the 0.2 and $\{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ size of ImageNet-100. For Scenario II that label space is growing, we report the H_t of the above models that are well-trained on ImageNet-20. Then, the H_t is computed between the ImageNet-20 and ImageNet-$\{30, 40, 50, 60, 70, 80, 90, 100\}$. The **Left** and **Right** of Figure I show the results of Scenario I and II, respectively. With the growth of new data, the previous models suffer more severe accuracy difference. Based on these results, one can choose a suitable threshold to determine whether to adjust the model architecture, according to the task at hand. In this paper, we set the threshold ϵ to 0.02.

It worth mentioning that for the Scenario II of increasing label space, the model performance must degrade. However, in this case, the users still need this adaptation condition to measure whether the degradation greater than a given threshold ϵ (in Eqn. 4) and then determine whether to adjust.

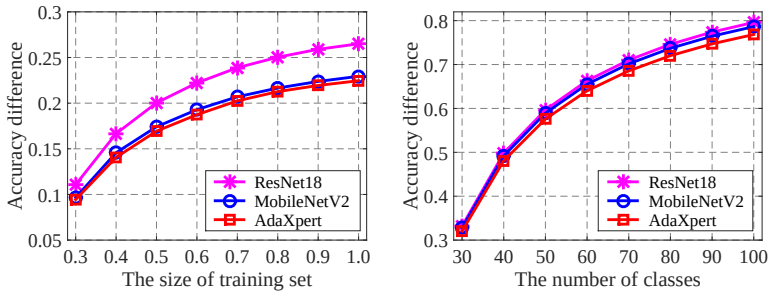

Figure I. An illustration of accuracy difference (H_t in Eqn. (4)) for different growing data. The **Left** and **Right** represent the increasing data volume and the number of classes, respectively.

D. More Discussions on Wasserstein Distance

To compute distribution distance, one can use many metrics including Jensen-Shannon (JS) divergence (Fuglede & Topsoe, 2004), Wasserstein distance (WD) (Sriperumbudur et al., 2010) and etc. In this paper, we choose WD as our metric since 1) it is an effective metric to establish a geometric tool for comparing probability distributions, and it has been widely used in deep learning (e.g., GAN) and 2) it has a little stronger discrimination ability to recognize difference extents than JS divergence in our case. In this section, we compare WD with JS on two considered scenarios, i.e., increasing data volume (**Scenario I**) and the number of classes (**Scenario II**). (1) For Scenario I, based on our architecture searched on 0.2 training set of ImageNet-100, we compute the distance d_t between 0.2 training set and $\{0.4, 0.6, 0.8, 1.0\}$ training set. (2) For Scenario II, based on our architecture obtained on ImageNet-20, we compute the distance between ImageNet-20 and ImageNet-$\{40, 60, 80, 100, 200\}$.

Computation details of WD and JS. As described in Sect. 3.2, we first feed current data $\mathcal{D}_t$ and previous data $\mathcal{D}_{t-1}$ into the previous model α_{t-1}, and then obtain their corresponding feature matrices $\mathbf{M}_t \in \mathbb{R}^{m\times q}$ and $\mathbf{M}_{t-1} \in \mathbb{R}^{n\times q}$, respectively. Here, m and n denote the number of samples in $\mathcal{D}_t$ and $\mathcal{D}_{t-1}$ respectively, and q denotes the feature dimension. We assume that $\mathbf{M}_t$ and $\mathbf{M}_{t-1}$ are from two multivariate Gaussian distributions $\mathbb{P}_t$ and $\mathbb{P}_{t-1}$, and use the Maximum Likelihood Estimation method to estimate its distribution parameters, i.e., $\mathbb{P}_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ and $\mathbb{P}_{t-1} \sim \mathcal{N}(\mu_{t-1}, \Sigma_{t-1})$. Based on the above, the WD (Takatsu et al., 2011) between $\mathcal{D}_t$ and $\mathcal{D}_{t-1}$ is computed as follows:

$$\mathcal{W}(\mathcal{D}_t, \mathcal{D}_{t-1}) = ||\mu_t - \mu_{t-1}||_2^2 + \mathrm{tr}\Big(\Sigma_t + \Sigma_{t-1} - 2(\Sigma_{t-1}^{1/2}\Sigma_t\Sigma_{t-1}^{1/2})^{1/2}\Big), \tag{A}$$

and the JS disvergence (Fuglede & Topsoe, 2004) is computed by:

$$\mathrm{JSD}(\mathcal{D}_t,\mathcal{D}_{t-1}) = \frac{1}{2}\Big(\mathrm{KL}(\mathrm{P}_t||\frac{\mathrm{P}_t+\mathrm{P}_{t-1}}{2}) + \mathrm{KL}(\mathrm{P}_{t-1}||\frac{\mathrm{P}_t+\mathrm{P}_{t-1}}{2})\Big), \tag{B}$$
$$\text{where}\quad \mathrm{KL}(\mathrm{P}_t||\mathrm{P}_{t-1}) = \int_{-\infty}^{+\infty} \mathrm{P}_t(x)\log(\frac{\mathrm{P}_t(x)}{\mathrm{P}_{t-1}(x)})dx.$$

Here, P_t is the probability density function of $\mathbb{P}_t$.

Comparison between WD and JS. As shown in Figure II, both WD and JS are able to recognize the difference extent between current and previous data. In general, the more different the current data from previous data, the larger WD and JS are. For the Scenario II that label space is growing, WD show stronger discrimination ability than JS (see Figure II (**right**)). To be specific, the JS between ImageNet-20 and ImageNet-$\{100, 200\}$ closes to 1 and thus fails to well recognize the difference extent between them. In contrast, the WD is still able to recognize the difference between ImageNet-100 and ImageNet-200.

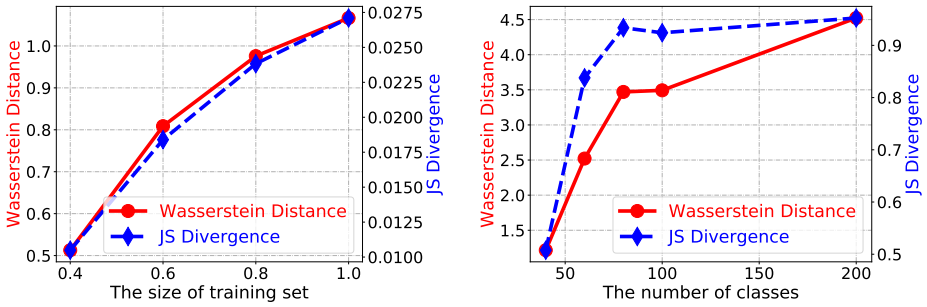

Figure II. Comparison between WD and JS divergence.

Effectiveness of Wasserstein Distance in AdaXpert. In our method, we conduct different architecture adjustment strategies based on the Wasserstein distance (WD) between current and previous data. This enables our adjustment to achieve better model accuracy with moderate computational overhead. We compare our method with AdaXpert w/o WD, i.e., the goal of reward function is only to maximize the model accuracy. Specifically, we conduct experiments on ImageNet-100 on Scenario I. As shown in Table A, with the data growth, 'AdaXpert w/o WD' is prone to obtain a larger network for the

Table A. Effectiveness of Wassertein distance in AdaXpert. 'AdaXpert w/o WD' denotes AdaXpert without considering data difference (WD) in the reward function.

Metric	Method	20% data	40% data	80% data	100% data
Acc. (%)	AdaXpert w/o WD	65.12	73.76	79.10	80.96
	AdaXpert	64.90	73.28	79.28	80.74
MAdds (M)	AdaXpert w/o WD	279	284	298	302
	AdaXpert	171	199	232	252

current data. However, the network performance is comparable with our method in most cases. This further demonstrates the necessity of considering the data difference with WD to conduct the adjustment.

Discussion on Gaussian Assumption. In this paper, we compute the WD between current and previous data by assuming that they are from two multivariate Gaussian distributions. Here, we empirically demonstrate the reasonability of this assumption. Based on our AdaXpert-20 that well-trained on ImageNet-20, we compute the sample matrix of ImageNet-40 (as described in Sect. 3.2) and randomly sample 6 dimensions to visualize its statistical histogram. As shown in Figure III, the sample features of each dimension approximately satisfy a Gaussian distribution. To achieve more accurate computation of WD, one can also use the non-parametric estimation methods (Sriperumbudur et al., 2010) as described in Sect. 3.2.

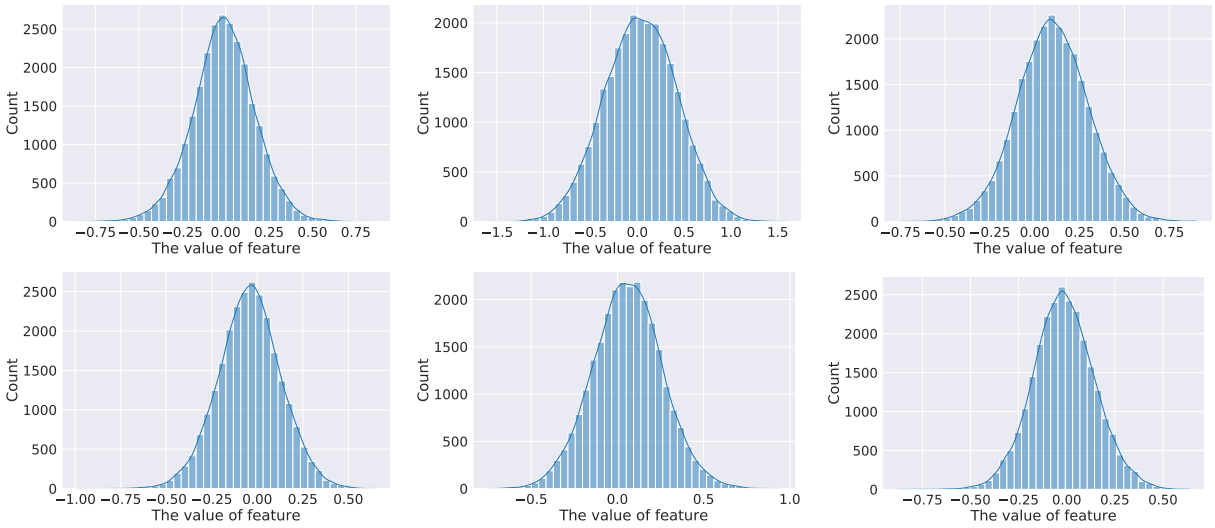

Figure III. Statistical histograms of data matrix on 6 randomly sampled dimensions.

E. Parameter Sensitivity of λ in Eqn. (5)

In this section, we evaluate our method with different trade-off parameter λ in Eqn. (5) from $\{2.0, 2.5, 3.0, 3.5\}\text{e}^{-4}$. The experiments are conducted on adjusting AdaXpert-20 to AdaXpert-40. We report the validation accuracy and #MAdds of the adjusted architectures in Figure IV (**Left**) and (**Right**), respectively.

From the results, with the increase of λ, our AdaXpert tends to find an architecture with fewer MAdds. However, the search accuracy (i.e., validation accuracy) achieves the best when $\lambda = 2.5\text{e}^{-4}$. Compared with $\lambda = 2\text{e}^{-4}$, $\lambda = 2.5\text{e}^{-4}$ achieves better search performance while with fewer #MAdds. This result further demonstrates that a small model is able to achieve better accuracy than a large model in a certain dataset. In this sense, one can design well-performed architectures and meanwhile keep the model MAdds as few as possible.

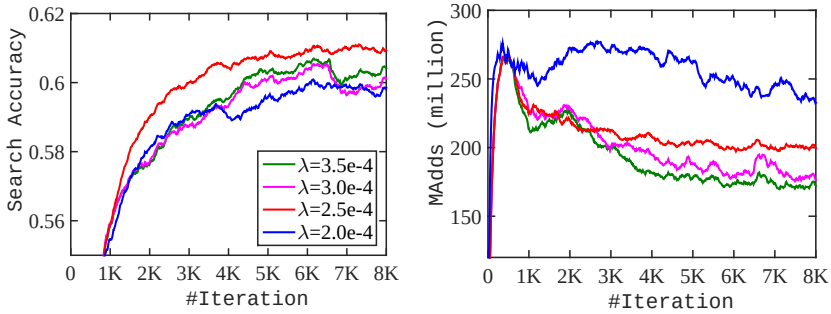

Figure IV. Training curves of our AdaXpert under different trade-off parameter λ.

References

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. Efficient architecture search by network transformation. In *AAAI Conference on Artificial Intelligence*, 2018.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. Randaugment: Practical automated data augmentation with a reduced search space. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 702–703, 2020.

Fuglede, B. and Topsoe, F. Jensen-shannon divergence and hilbert space embedding. In *Proceedings of the IEEE International Symposium on Information Theory*, 2004.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. In *International Conference on Learning Representations*, 2020.

Lu, Z., Sreekumar, G., Goodman, E., Banzhaf, W., Deb, K., and Boddeti, V. N. Neural architecture transfer. *arXiv preprint arXiv:2005.05859*, 2020.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *Proceedings of the International Conference on Machine Learning*, pp. 4092–4101, 2018.

Sriperumbudur, B. K., Fukumizu, K., Gretton, A., Schölkopf, B., and Lanckriet, G. R. Non-parametric estimation of integral probability metrics. In *IEEE International Symposium on Information Theory*, pp. 1428–1432. IEEE, 2010.

Takatsu, A. et al. Wasserstein geometry of gaussian measures. *Osaka Journal of Mathematics*, 48(4):1005–1026, 2011.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.