DON'T BE PICKY, ALL STUDENTS IN THE RIGHT FAMILY CAN LEARN FROM GOOD TEACHERS

Anonymous authors
Paper under double-blind review

ABSTRACT

State-of-the-art results in deep learning have been improving steadily, in good part due to the use of larger models. However, widespread use is constrained by device hardware limitations, resulting in a substantial performance gap between state-of-the-art models and those that can be effectively deployed on small devices.

While Knowledge Distillation (KD) theoretically enables small *student* models to emulate larger *teacher* models, in practice selecting a good student architecture requires considerable human expertise. Neural Architecture Search (NAS) appears as a natural solution to this problem but most approaches can be inefficient, as most of the computation is spent comparing architectures *sampled from the same distribution*, with negligible differences in performance.

In this paper, we propose to instead search for a *family* of student architectures sharing the property of being good at learning from a given teacher. Our approach AutoKD, powered by Bayesian Optimization, explores a flexible graph-based search space, enabling us to automatically learn the optimal student architecture distribution and KD parameters, while being $20\times$ more sample efficient compared to existing state-of-the-art. We evaluate our method on 3 datasets; on large images specifically, we reach the teacher performance while using $3\times$ less memory and $10\times$ less parameters. Finally, while AutoKD uses the traditional KD loss, it outperforms more advanced KD variants using hand-designed students.

1 INTRODUCTION

Recently-developed deep learning models have achieved remarkable performance in a variety of tasks. However, breakthroughs leading to state-of-the-art (SOTA) results often rely on very large models: GPipe, Big Transfer and GPT-3 use 556 million, 928 million and 175 billion parameters, respectively (Huang et al., 2019; Kolesnikov et al., 2020; Brown et al., 2020).

Deploying these models on user devices (e.g. smartphones) is currently impractical as they require large amounts of memory and computation; and even when large devices are an option (e.g. GPU clusters), the cost of large-scale deployment (e.g. continual inference) can be very high (Cheng et al., 2017). Additionally, target hardware does not always natively or efficiently support all operations used by SOTA architectures. The applicability of these architectures is, therefore, severely limited, and workarounds using smaller or simplified models lead to a *performance gap* between the technology available at the frontier of deep learning research and that usable in industry applications.

In order to bridge this gap, Knowledge Distillation (KD) emerges as a potential solution, allowing small *student* models to learn from, and emulate the performance of, large *teacher* models (Hinton et al., 2015a). The student model can be constrained in its size and type of operations used, so that it will satisfy the requirements of the target computational environment. Unfortunately, successfully achieving this in practice is extremely challenging, requiring extensive human expertise. For example, while we know that the architecture of the student is important for distillation (Liu et al., 2019b), it remains unclear how to design the optimal network given some hardware constraints.

With Neural Architecture Search (NAS) it is possible to discover an optimal student architecture. NAS automates the choice of neural network architecture for a specific task and dataset, given a *search space* of architectures and a *search strategy* to navigate that space (Pham et al., 2018; Real et al., 2017; Liu et al., 2019a; Carlucci et al., 2019; Zela et al., 2018; Ru et al., 2020). One important

limitation of most NAS approaches is that the search space is very restricted, with a high proportion of resources spent on evaluating very similar architectures, thus rendering the approach limited in its effectiveness (Yang et al., 2020). Conversely, properly exploring a large space requires huge computational resources: for example, recent work by Liu et al. (2019b) investigating how to find the optimal student requires evaluating $10,000$ models.

To overcome these limitations, we propose an automated approach to knowledge distillation, in which we look for a *family* of good students rather than a specific model. We find that even though our method, AutoKD, does not output one specific architecture, all architectures sampled from the optimal *family* of students perform well when trained with KD. This reformulation of the NAS problem provides a more expressive search space containing very diverse architectures, thus increasing the effectiveness of the search procedure in finding good student networks.

Our contributions are as follows: (**A**) a framework for combining KD with NAS and effectively emulate large models while using a fraction of the memory and of the parameters; (**B**) By searching for an optimal student family, rather than for specific architectures, our algorithm is up to 20x more sample efficient than alternative NAS-based KD solutions; (**C**) We significantly outperform advanced KD methods on a benchmark of vision datasets, despite using the traditional KD loss, showcasing the efficacy of our found students.

2 Related Work

Model compression has been studied since the beginning of the machine learning era, with multiple solutions being proposed (Choudhary et al., 2020; Cheng et al., 2017). Pruning based methods allow the removal of non-essential parameters from the model, with little-to-none drop in final performance. The primary motive of these approaches was to reduce the storage requirement, but they can also be used to speed up the model (LeCun et al., 1990; Han et al., 2015; Li et al., 2016). The idea behind quantization methods is to reduce the number of bits used to represent the weights and the activations in a model; depending on the specific implementation this can lead to reduced storage, reduced memory consumption and a general speed-up of the network (Fiesler et al., 1990; Soudry et al., 2014; Rastegari et al., 2016; Zhu et al., 2016). In low rank factorization approaches, a given weight matrix is decomposed into the product of smaller ones, for example using singular value decomposition. When applied to fully connected layers this leads to reduced storage, while when applied to convolutional filters, it leads to faster inference (Choudhary et al., 2020).

All the above mentioned techniques can successfully reduce the complexity of a given model, but are not designed to substitute specific operations. For example, specialized hardware devices might only support a small subset of all the operations offered by modern deep learning frameworks. In Knowledge Distillation approaches, a large model (the teacher) distills its knowledge into a smaller student architecture (Hinton et al., 2015b). This knowledge is assumed to be represented in the neural network's output distribution, hence in the standard KD framework, the output distribution of a student's network is optimized to match the teacher's output distribution for all the training data (Yun et al., 2020; Ahn et al., 2019; Yuan et al., 2020; Tian et al., 2020; Tung & Mori, 2019).

The work of Liu et al. (2019b) shows that the architecture of a student network is a contributing factor in its ability to learn from a given teacher. The authors propose combining KD with a traditional NAS pipeline, based on Reinforcement Learning, to find the optimal student. While this setup leads to good results, it does so at a huge computational cost, requiring over 5 days on 200 TPUs. Similarly, Gu & Tresp (2020) also look for the optimal student architecture, but do so by searching for a subgraph of the original teacher; therefore, it cannot be used to substitute unsupported operations.

Orthogonal approaches, looking at how KD can improve NAS, are explored by Trofimov et al. (2020) and Li et al. (2020). The first establishes that KD improves the correlation between different budgets in multi-fidelity methods, while the second uses the teacher supervision to search the architecture in a blockwise fashion.

3 Searching for the optimal student network generator

The AutoKD framework (Fig. 1) combines Bayesian Optimization (BO), Neural Architecture Search (NAS) and Knowledge Distillation (KD). AutoKD defines a family of random network generators

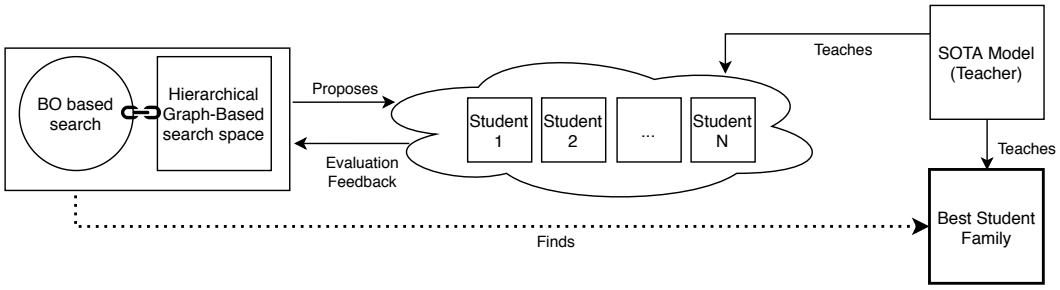


Figure 1: AutoKD leverages multi-fidelity Bayesian Optimization, a hierarchical graph-based search space coupled with an architecture generator optimization pipeline, to find the optimal student for knowledge distillation. See section 3 for a detailed description.

$G(\mathbf{\Theta})$ parameterized by a hyperparameter $\mathbf{\Theta}$, from where student networks are sampled. BO uses a surrogate model to propose generator hyperparameters, while students from these generators are trained with KD using a state-of-the-art teacher network. The student performances are evaluated and provided as feedback to update the BO surrogate model. To improve our BO surrogate model, the *search* procedure is iterated, until the best family of student networks $G(\mathbf{\Theta}^*)$ is selected. In this section we specify all components of AutoKD. See also Fig. 1 and Algorithm 1 for an overview.

3.1 Knowledge Distillation

Knowledge Distillation (KD; Hinton et al., 2015b) is a method to transfer, or distill, knowledge from one model to another—usually from a large model to small one—such that the small *student* model learns to emulate the performance of the large *teacher* model. KD can be formalized as minimizing the objective function:

$$\mathcal{L}_{\text{KD}} = \sum_{x_i \in X} l(f_T(x_i), f_S(x_i)) \tag{1}$$

where l is the loss function that measures the difference in performance between the teacher f_T and the student f_S, x_i is the ith input, y_i is the ith target. The conventional loss function l used in practice is a linear combination of the traditional cross entropy loss L_{CE} and the Kullback–Leibler divergence L_{KL} of the pre-softmax outputs for f_T and f_S:

$$l = (1-\alpha)L_{\text{CE}} + \alpha L_{\text{KL}}\left(\sigma\left(f_T(x_i)/\tau\right), \sigma\left(f_S(x_i)/\tau\right)\right) \tag{2}$$

where σ is the softmax function $\sigma(x) = 1/(1+\exp(-x))$, and τ is the softmax temperature. Hinton et al. (2015b) propose "softening" the probabilities using temperature scaling with $\tau \geq 1$. The parameter α represents the weight trade-off between the KL loss and the cross entropy loss L_{CE}. The $\mathcal{L}_{\text{KD}}$ loss is characterized by the hyper-parameters: α and τ; popular choices are $\tau \in \{3, 4, 5\}$ and $\alpha = 0.9$ (Huang & Wang, 2017; Zagoruyko & Komodakis, 2016; Zhu et al., 2018). Numerous other methods (Polino et al., 2018; Huang & Wang, 2017; Tung & Mori, 2019) can be formulated as a form of Equation (2), but in this paper we use the conventional loss function l.

Traditionally in KD, both the teacher and the student network have predefined architectures. In contrast, AutoKD defines a search space of student network architectures and finds the optimal student by leveraging neural architecture search, as detailed below.

3.2 Student search via Generator Optimization

Most NAS method for vision tasks employ a cell-based search space, where networks are built by stacking building blocks (*cells*) and the operations inside the cell are searched (Pham et al., 2018; Real et al., 2017; Liu et al., 2019a). This results in a single architecture being output by the NAS procedure. In contrast, more flexible search spaces have recently been proposed that are based on random graph generators (Xie et al., 2019). The hyperparameters of the generator define the characteristics of the networks that it represents.

NAGO optimizes an architecture generator instead of a single architecture and proposes a hierarchical graph-based space which is highly expressive yet low-dimensional (Ru et al., 2020). Specifically,

Algorithm 1: AutoKD

Input: Network generator G, BO surrogate model $p(f|\mathbf{\Theta}, D)$ and acquisition function $\alpha(\mathbf{\Theta}|D)$, T iterations, B configurations to be evaluated in each iteration.
for $t = 1$ **to** T **do**
  Recommend $\{\mathbf{\Theta}_t^j\}_{j=1}^B = \arg\max \alpha_{t-1}(\mathbf{\Theta}|D)$
  for $j = 1$ **to** B **in parallel do**
    Sample an architecture from $G(\mathbf{\Theta}_t^j)$, train it on the KD loss $\mathcal{L}_{\text{KD}}$ (equations 1 and 2), and evaluate its validation performance f_t^j
  end for
  Update D and thus $p(f|\mathbf{\Theta}, D)$ with $\{\mathbf{\Theta}_t^j, f_t^j\}_{j=1}^B$
end for
Obtain the best performing configuration $\mathbf{\Theta}^*$ for the student network generator.
Sample k architectures from $G(\mathbf{\Theta}^*)$, train them to completion, and obtain test performance.

the search space of NAGO comprises three levels of graphs (where the node in the higher level is a lower-level graph). The top level is a graph of cells (G_{top}) and each cell is itself a graph of middle-level modules (G_{mid}). Each module further corresponds to a graph of bottom-level operation units (G_{bottom}) such as a relu-conv3×3-bn triplet. NAGO adopts three random graph generators to define the connectivity/topology of G_{top}, G_{mid} and G_{bottom} respectively, and thus is able to produce a wide variety of architectures with only a few generator hyperparameters. AutoKD employs NAGO as the NAS backbone for finding the optimal student family.

Our pipeline consists of two phases. In the first phase (*search*), a multi-fidelity Bayesian optimisation technique, BOHB (Falkner et al., 2018), is employed to optimise the low-dimensional search space. BOHB uses partial evaluations with smaller-than-full budget to exclude bad configurations early in the search process, thus saving resources to evaluate more promising configurations. Given the same time constraint, BOHB evaluates many more configurations than conventional BO which evaluates all configurations with full budget. As Ru et al. (2020) empirically observe that good generator hyperparameters lead to a tight distribution of well-performing architectures (small performance standard deviation), we similarly assess the performance of a particular generator hyperparameter value with only one architecture sample. In the second phase (*retrain*A), AutoKD uniformly samples multiple architectures from the optimal generator found during the search phase and evaluates them with longer training budgets to obtain the best architecture performance.

Instead of the traditionally used cross-entropy loss, AutoKD uses the KD loss in equation 2 to allow the sampled architecture to distill knowledge from its teacher. The KD hyperparameters temperature τ and loss weight α are included in the search space and optimized simultaneously with the architecture to ensure that the student architectures can efficiently distill knowledge both from the designated teacher and the data distribution. A full overview of the framework is shown in Fig. 1.

4 EXPERIMENTS

The first part of this section studies how KD can improve the performance of our chosen NAS backbone (NAGO). In the second part, we show how a family of students, when trained with KD (AutoKD), can emulate much larger teachers, significantly outperforming current hand-crafted architectures.

Experimental setup. All of our experiments were run on the two, small-image, standard object recognition datasets *CIFAR10* and *CIFAR100* (Krizhevsky, 2009), as well as *MIT67* for large-image scene recognition (Quattoni & Torralba, 2009). We limit the number of student network parameters to 4.0M for small-image tasks and 6.0M for large-image tasks. Following Liu et al. (2019b), we picked Inception-Resnet-V2 (Szegedy et al., 2016) as a teacher for the large image dataset. As that model could not be directly applied to small images, and to explore the use of a machine-designed network as a teacher, we decided to use the best DARTS (Liu et al., 2019a) architecture to guide the search on the CIFAR datasets. All experiments are run on NVIDIA Tesla V100 GPUs.

Temperature \ Weight	0	0.3	0.5	0.7	0.9
1	92.13	91.06	91.09	92.53	92.38
2	90.45	91.66	92.19	93.29	93.42
3	91.55	92.49	93.15	93.72	93.45
4	91.02	92.27	93.37	94.51	94.29
5	91.35	93.1	93.42	94.27	94.25
7	91.4	93.01	93.66	93.25	93.93
10	91.28	93.15	93.96	93.45	93.96
15	90.93	93.6	94.32	93.23	93.93
20	90.88	93.65	93.6	93.49	94.45
40	90.96	93.25	93.25	93.38	93.78

Figure 2: Student model test accuracy for various temperature (τ) and loss weight (α) combinations. The model was sampled from a generator with random parameters, and trained with KD on CIFAR10 using the DARTS teacher. The table suggests that there is a positive correlation between the KD loss weight and the performance of the student model.

NAS implementation. Our approach follows the search space and BO-based search protocol proposed by NAGO (Ru et al., 2020), as such our student architectures are based on hierarchical random graphs. Likewise, we employ a multi-fidelity evaluation scheme based on BOHB (Falkner et al., 2018) where candidates are trained for different epochs (30, 60 and 120) and then evaluated on the validation set. In total, only $\sim$300 models are trained during the search procedure: using 8 GPUs, this amounts to $\sim$2.5 days of compute on the considered datasets. At the end of the search, we sample 8 architectures from the best found generator, train them for 600 epochs (with KD, using the optimal temperature and loss weight found during the search), and report the average performance (top-1 test accuracy). All remaining training parameters were set following Ru et al. (2020).

In AutoKD, we include the knowledge distillation hyperparameters, *temperature* and *weight*, in the search space, so that they are optimized alongside the architecture. The temperature ranges from 1 to 10, while the weight ranges from 0 to 1. Fig. 2 illustrates the importance of these hyperparameters when training a randomly sampled model, lending support to their inclusion in the search.

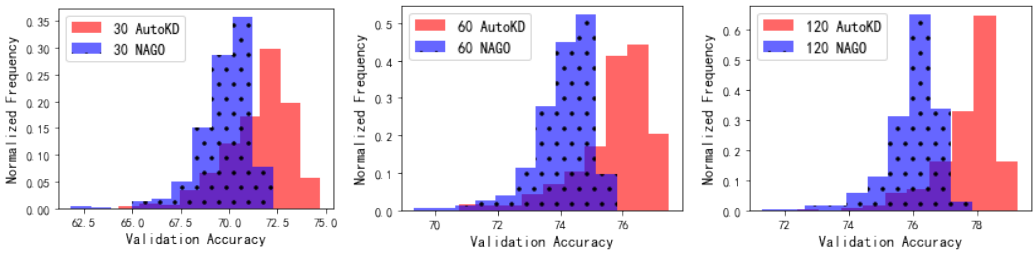
4.1 Impact of Knowledge Distillation on NAS

To understand the contribution from KD, we first compare vanilla NAGO with AutoKD on CIFAR100. Fig. 3 shows the validation accuracy distribution at different epochs: clearly, using KD leads to better performing models. Indeed this can be seen in more detail in Fig. 4, where we show the performance of the best found model vs the wall clock time for each budget. It is worth mentioning that while the KD version takes longer (as it needs to compute the lessons on the fly), it consistently outperforms vanilla NAGO by a significant margin on all three datasets.

Not only does AutoKD offer better absolute performance, but it also enables better multi-fidelity correlation, as can be seen in Fig. 5. For example, the correlation between 30 and 120 epochs improves from 0.49 to 0.82 by using KD, a result that is consistent with the findings in Trofimov et al. (2020). Note that multi-fidelity methods work under the assumption that the rankings at different budgets remains consistent to guarantee that the best models progress to the next stage. A high correlation between the rankings is, as such, crucial.

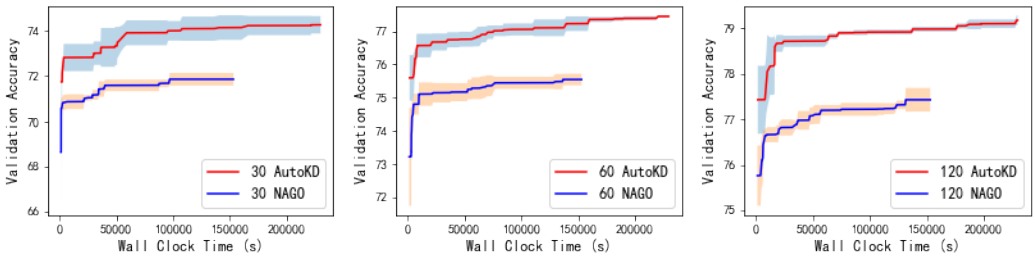
4.2 Large model emulation

At its core, AutoKD's goal is to emulate the performance of large SOTA models with smaller students. Fig. 7 shows how the proposed method manages to reach the teacher's performance while using only 1/9th of the memory on small image datasets. On MIT67, the found architecture is not only using 1/3rd of the memory, but also 1/10th of parameters. Finally, it is worth noting how AutoKD increases student performance, as such the high final accuracy cannot only be explained by the NAS procedure. Indeed, looking at Fig. 8 it is clear how KD improves both the speed of convergence and the final accuracy. Furthermore, as shown in Fig. 6, the optimal family of architectures is actually different when searched with KD.



(a) 30 epochs (b) 60 epochs (c) 120 epochs

Figure 3: Top-1 accuracy distribution for AutoKD and standard NAGO at different budgets on CIFAR100. The histograms are tallied across 5 runs. Across all budgets, AutoKD samples architectures with improved performances in top-1 accuracy compared to NAGO.



(a) 30 epochs (b) 60 epochs (c) 120 epochs

Figure 4: Top-1 accuracy of the best found model at a given computation time on CIFAR100 for AutoKD (red) and NAGO (blue) across different budgets. Each method was run 8 times with the bold curve showing the average performance and the shaded region the standard deviation.

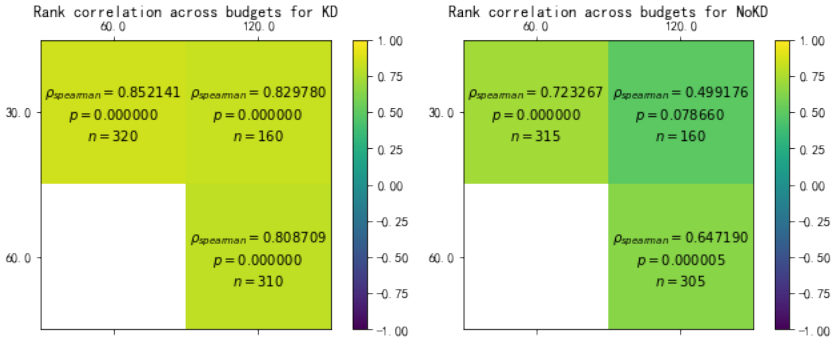


Figure 5: Rank Correlations between different epoch budgets for AutoKD (KD; left) and Standard NAGO (right) computed for 5 runs of NAGO and AutoKD respectively. NAGO reports a rank correlation coefficient of $5 \cdot 10^{-1}$ for epoch pair 30-120, which is $3.3 \cdot 10^{-1}$ less than that of the KD rank correlation. These results show that the rank correlation across all budget pairs vastly improves when knowledge distillation is applied.

Table 1 shows the comparison of AutoKD with other KD methods. AutoKD is able to outperform a variety of more advanced KD approaches while simultaneously requiring much fewer parameters. The exception to this is CIFAR10, where AutoKD outperforms other methods but with a larger number of parameters. This is because the default networks in the NAGO search space have 4M parameters, which is too large for this application. Accuracy-wise, the only method doing better on CIFAR100, Yuan et al. (2020), does so with a student with significantly more parameters (34M vs 4M).

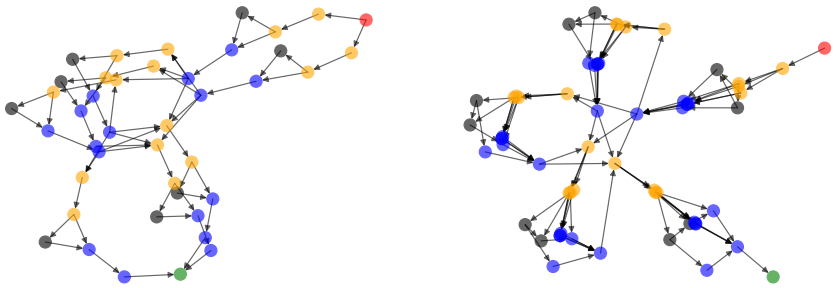


Figure 6: Networks sampled from the best generator parameters found by AutoKD (left) and NAGO (right) on CIFAR10. Note how the former is organized with 8 clusters of 3 nodes, while the latter has 5 clusters of 10 nodes, showcasing how the optimal configuration depends on teacher supervision.

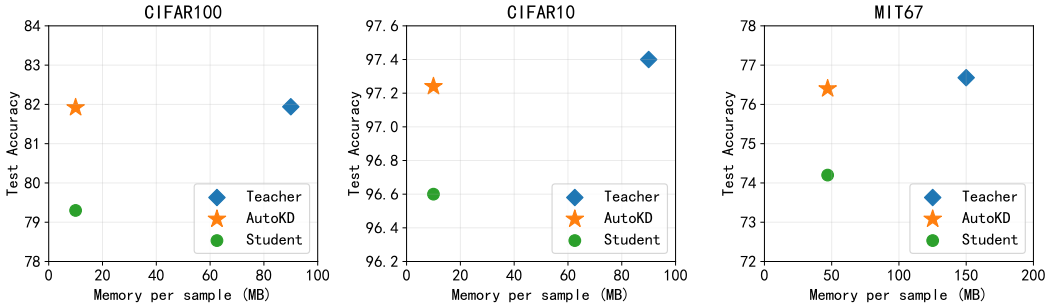


Figure 7: Accuracy vs memory per sample for teacher and best student. Note that the MIT67 teacher has almost 10× the number of parameters of the student (54M vs 6M).

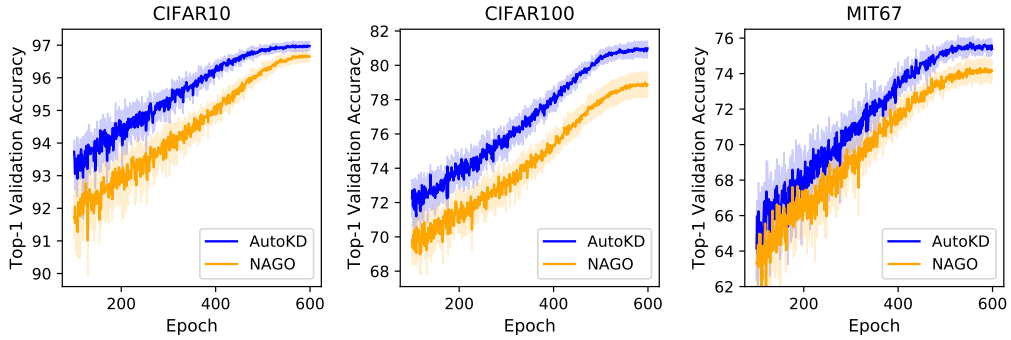


Figure 8: Final training curves for the top generator found by NAGO and AutoKD, for CIFAR10, CIFAR100 and MIT67. Each generator was sampled 8 times and the 8 corresponding architectures trained for 600 epochs. Bold line represents the average; shaded region represents *std* deviation.

5 Discussion and Conclusion

Improving Knowledge Distillation by searching for the optimal student architecture is a promising idea that has recently started to gain attention in the community (Liu et al., 2019b; Trofimov et al., 2020; Gu & Tresp, 2020). In contrast with earlier KD-NAS approaches, which search for specific architectures, our method searches for a *family* of networks sharing the same characteristics. The main benefit of this approach is sample efficiency: while traditional methods spend many computational resources evaluating similar architectures (Yang et al., 2020), AutoKD is able to avoid this pitfall: for instance, the method of Liu et al. (2019b) requires $\sim 10,000$ architecture samples, while AutoKD can effectively search for the optimal student family with only 300 samples. Compared to

Table 1: Comparison with KD state-of-the-art. AutoKD uses the standard KD loss (Hinton et al., 2015b), while competing methods are using modern variants. Improvement of student accuracy in parenthesis is with respect to the same student without KD. "$\uparrow \times f$" and "$\downarrow \times f$" denotes the increase/decrease in parameter count by the factor f relative to AutoKD for the same dataset. The top performing student accuracy (*S acc*) for each dataset is specified in bold. For each dataset, we sampled 8 architectures and averaged them over 5 runs. For MIT67, the VID method is a transfer learning task from ImageNet to MIT67, hence the absence of teacher accuracy (*T acc*) statistics.

Method†	Teacher (T)	Student (S)	S params	T acc	S acc
		MIT67			
SKD	ResNet-18	ResNet-18	11.5M $\uparrow\times 1.9$	55.3	60.4 (+5.1)
VID	ResNet-34	ResNet-18	11.5M $\uparrow\times 1.9$	—	71.9 (+0.9)
VID	ResNet-34	VGG-9	10.9M $\uparrow\times 1.8$	—	72.0 (+6.0)
AutoKD (ours)	InceptionResNetV2	NAGO	6.0M	76.6	**76.0** (+1.8)
		CIFAR100			
CRD	ResNet-32 $\times$ 4	ShuffleNetV2	7.4M $\uparrow\times 1.9$	79.4	75.7 (+3.8)
CRD	WRN 40-2	ShuffleNetV1	3.4M $\downarrow\times 1.2$	75.6	76.0 (+5.6)
VID	WRN 40-2	WRN 40-2	2.2M $\downarrow\times 1.8$	74.2	76.1 (+1.8)
KD-LSR	ResNet-18	ResNet-18	11.5M $\uparrow\times 2.9$	75.9	77.4 (+1.5)
SKD	ResNet-18	ResNet-18	11.5M $\uparrow\times 2.9$	75.3	79.6 (+4.3)
KD-LSR	DenseNet-121	DenseNet-121	7.0M $\uparrow\times 1.8$	79.0	80.3 (+1.3)
KD-LSR	ResNeXt29	ResNeXt29	34.2M $\uparrow\times 8.6$	81.0	**82.1** (+1.1)
AutoKD (ours)	DARTS	NAGO	4.0M	81.9	81.2 (+2.6)
		CIFAR10			
VID	WRN 40-2	WRN 16-1	0.7M $\downarrow\times 5.7$	94.3	91.9 (+1.2)
SPKD	WRN 16-8	WRN 40-2	2.2M $\downarrow\times 1.8$	95.8	95.5 (+0.6)
AT	WRN 16-8	WRN 40-2	2.2M $\downarrow\times 1.8$	95.8	95.5 (+0.6)
AutoKD (ours)	DARTS	NAGO	4.0M	97.4	**97.1** (+0.8)

† SKD (Yun et al., 2020), VID (Ahn et al., 2019), KD-LSR (Yuan et al., 2020), CRD (Tian et al., 2020), SPKD (Tung & Mori, 2019), AT (Zagoruyko & Komodakis, 2016).

traditional KD methods, AutoKD is capable of achieving better performance with student architectures that have less parameters (and/or use less memory) than hand-defined ones.

Our message "DON'T BE PICKY" refers to the fact that the macro-structure (connectivity and capacity) of a network is more important than its micro-structure (the specific operations). This has been shown to be true for non-KD NAS (Xie et al., 2019; Ru et al., 2020) and is here experimentally confirmed for KD-NAS as well. Changing the focus of optimization in this way releases computational resources that can be used to effectively optimize the global properties of the network. Additionally, the fact that a family of architectures can characterized by a small number of hyperparameters makes the comparison of architectures more meaningful and interpretable. In the current implementation, AutoKD finds the optimal student family, in which all sampled architectures perform well: future work should explore how to fully exploit this distribution, possibly finetuning the network distribution to obtain an ever better performing model.

To summarize, AutoKD offers a strategy to efficiently emulate large, state-of-the-art models with a fraction of the model size. Indeed, our family of searched students consistently outperforms the best hand-crafted students on CIFAR10, CIFAR100 and MIT67.

REFERENCES

Sungsoo Ahn, Shell Xu Hu, Andreas Damianou, Neil D Lawrence, and Zhenwen Dai. Variational information distillation for knowledge transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9163–9171, 2019.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv:2005.14165*, 2020.

Fabio Carlucci, Pedro M. Esperança, Marco Singh, Antoine Yang, Victor Gabillon, Hang Xu, Zewei Chen, and Jun Wang. MANAS: Multi-agent neural architecture search. *arxiv:1909.01051*, 2019.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv:1710.09282*, 2017.

Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, pp. 1–43, 2020.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning (ICML)*, pp. 1436–1445, 2018.

Emile Fiesler, Amar Choudry, and H John Caulfield. Weight discretization paradigm for optical neural networks. In *Optical interconnections and networks*, volume 1281, pp. 164–173. International Society for Optics and Photonics, 1990.

Jindong Gu and Volker Tresp. Search for better students to learn distilled knowledge. *arXiv preprint arXiv:2001.11612*, 2020.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015a.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015b.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems (NeurIPS*, pp. 103–112, 2019.

Zehao Huang and Naiyan Wang. Like what you like: Knowledge distill via neuron selectivity transfer. *arXiv preprint arXiv:1707.01219*, 2017.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big Transfer (BiT): General visual representation learning. In *European Conference on Computer Vision (ECCV)*, 2020.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1989–1998, 2020.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. 2016.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019a.

Yu Liu, Xuhui Jia, Mingxing Tan, Raviteja Vemulapalli, Yukun Zhu, Bradley Green, and Xiaogang Wang. Search to distill: Pearls are everywhere but not the eyes. *arXiv preprint arXiv:1911.09074*, 2019b.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning (ICML)*, pp. 4092–4101, 2018.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *Computer Vision and Pattern Recognition (CVPR)*, pp. 413–420, 2009.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning (ICML)*, pp. 2902–2911, 2017.

Binxin Ru, Pedro M Esperanca, and Fabio M Carlucci. Neural Architecture Generator Optimization. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in neural information processing systems*, pp. 963–971, 2014.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive representation distillation. In *International Conference on Learning Representations*, 2020.

Ilya Trofimov, Nikita Klyuchnikov, Mikhail Salnikov, Alexander Filippov, and Evgeny Burnaev. Multi-fidelity neural architecture search with knowledge distillation. *arXiv preprint arXiv:2006.08341*, 2020.

Frederick Tung and Greg Mori. Similarity-preserving knowledge distillation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1365–1374, 2019.

Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *arXiv:1904.01569*, 2019.

Antoine Yang, Pedro M Esperança, and Fabio M Carlucci. NAS evaluation is frustratingly hard. In *International Conference on Learning Representations (ICLR)*, 2020.

Li Yuan, Francis EH Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3903–3911, 2020.

Sukmin Yun, Jongjin Park, Kimin Lee, and Jinwoo Shin. Regularizing class-wise predictions via self-knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13876–13885, 2020.

Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*, 2016.

Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv:1807.06906*, 2018.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

Xiatian Zhu, Shaogang Gong, et al. Knowledge distillation by on-the-fly native ensemble. In *Advances in neural information processing systems*, pp. 7517–7527, 2018.