Optimizing Slimmable Networks for Multiple Target Platforms

Zifu Wang*[1] and Matthew B. Blaschko[1]

[1]ESAT-PSI, KU Leuven, Belgium

Abstract

In this work, we extend platform-aware adaptive training to the weighted average of multiple target platforms, where the weighting is determined e.g. by the market share of the target platform. To simulate different market regimes, we generate different weight settings by a Chinese restaurant process to benchmark optimization strategies. We use a neural architecture search framework [2] based on Markov Random Fields to efficiently find the optimal channel configurations for each platform, and investigate different sampling strategies to train a single slimmable network [15, 17] that can be deployed to multiple platforms at the same time. Empirical results on CIFAR-100 demonstrate improved performance over the original slimmable network across different weight settings, while maintaining efficient training.

1 Introduction

In many realistic applications where resources (energy, latency) are limited, such as autonomous driving, mobile phones and Internet of Things (IoT) devices, neural networks must be adapted to platform-dependent requirements. This leads to recent interests in designing efficient architectures while still maintaining a comparable accuracy [4, 5, 6, 8, 12, 13]. However, various platforms often have a diverse range of computational budgets and they are able to support networks of different sizes. Even on the same platform, the computational capacity varies, for instance, background applications often consume lots of memory, and it may be desirable to adjust the amount of computation adaptively. Deploying a single network to different platforms can either lead to a waste of unused resources of high-capacity platforms or unsatisfactory user experience of low-capacity platforms due to high latency.

To deal with this, for instance, MobileNets [6, 7, 10] can execute at various widths by shrinking all layers with the same ratio, but this would require to train a single network for each target platform. Subsequent extensions of the slimmable network framework [15, 17] tackle this issue by training a single network with shared weights that can execute at different widths during the inference time to achieve a accuracy-efficiency trade-off. However, it still requires to scale all layers uniformly, and it has been shown that this can be sub-optimal [2, 14].

Moreover, for suppliers, not all platforms are of the same importance. For example, based on recent statistics in October 2020 [11], the mobile vendor market share in US can be well approximated by a Chinese restaurant process (Figure 1). This is a natural model for market share as each additional customer tends to pick a brand proportional to the usage of their peers. Developers who have very limited computational budgets or time constraints may want to deploy a single network across multiple platforms with child-models sharing the same weights. Moreover, a natural objective is to give higher weight to the performance of platforms of greater value, as represented by the number of end-users or overall purchasing power of a target platform.

In this paper, we first generate several weight distributions using a Chinese restaurant process (CRP). Then we extend AOWS [2] to the multi-platform setting and model a slimmable network [15, 17] as a pairwise Markov random field (MRF). With Viterbi inference, we can efficiently search for optimal channel configurations that cater to each platform's capacity requirement. Subsequently, we retrain a single slimmable network with the obtained channel configurations that can be de-

*Corresponding Author: zifu.wang@esat.kuleuven.be

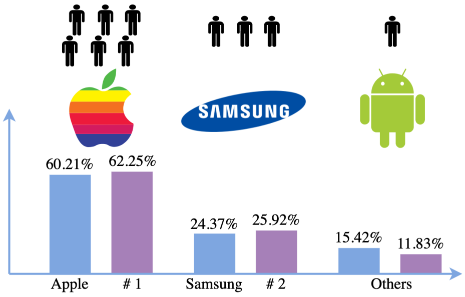

Figure 1: Mobile vendor market share in the United States of America in October, 2020 vs. a Chinese restaurant process with $\alpha = 0.3$ and $\theta = 0.15$.

ployed across multiple platforms, and investigate various sampling strategies so as to maximize the weighted accuracy. Finally, we evaluate our method on CIFAR-100 with different weight distributions and show improved performance over the original slimmable network.

2 Slimmable network and AOWS

2.1 Slimmable Networks

Slimmable networks [15, 17] are a family of neural networks with shared parameters that can execute at various predefined width configurations at inference time to achieve accuracy-efficiency trade-offs. For each layer i of a neural network, the width c_i is chosen from a width set $C_i \in [c_{i,min}, ..., c_{i,max}]$. Then a channel configuration $\boldsymbol{c} = (c_1, ..., c_n)$ uniquely defines a single child-network. Let $\boldsymbol{c}_{max} = (c_{1,max}, ..., c_{n,max})$ be the full network and $\boldsymbol{c}_{min} = (c_{1,min}, ..., c_{n,min})$ the minimum network.

The so-called sandwich rule [15] is based on the assumption that the performance of all intermediate widths is bounded above by the maximum network $\boldsymbol{c}_{max}$ and below by the minimum network $\boldsymbol{c}_{min}$. In other words, optimizing the performance of lower and upper bound can implicitly improve the performance of all child-networks. This leads to the following training procedure for slimmable networks: at each training iteration t, the network first executes with the maximum network $\boldsymbol{c}_{max}$ and back-propagates the loss; then executes with the minimum network $\boldsymbol{c}_{min}$ as well as k random configurations $\boldsymbol{c}^t = (c_1^t, ..., c_n^t)$ in turn.

2.2 AOWS

AOWS [2] is a platform-aware NAS method building on the slimmable network framework [15, 17]. The work is based on the assumption that the performance of a slimmable network executed at a given channel configuration $\boldsymbol{c} = (c_1, ..., c_n)$ is a good proxy for the performance of a neural network trained from scratch with only that channel configuration.

By modeling a slimmable network as a pairwise Markov random field (MRF), AOWS can efficiently search for optimal channel configurations with Viterbi inference. Moreover, a smoothed Viterbi inference [9] is used during training to define a channel configuration sampling scheme, leading to an improved accuracy-efficiency trade-off in their experiments.

In this work, we call the model that uses smoothed Viterbi as AOWS, and OWS for the model that simply performs Viterbi inference. Our work is closely related to AOWS and in the next section, we will show more details of their approach and extend it to the multi-platform setting.

3 Multiple platform extension

For m platforms each with a different resource constraint (FLOPS, latency, memory) R_k and an economic importance μ_k, we consider the following multi-objective optimization problem where the loss weighted by the economic importance μ_k is minimized:

$$\begin{aligned} &\min_{\boldsymbol{w}} \min_{\boldsymbol{c}_1,...,\boldsymbol{c}_m} \mathcal{L}(\boldsymbol{c}_1, ..., \boldsymbol{c}_m, \boldsymbol{w}) \\ = &\min_{\boldsymbol{w}} \sum_{k=1}^{m} \mu_k \min_{\boldsymbol{c}_k} \mathcal{L}(\boldsymbol{c}_k, \boldsymbol{w}) \\ &\text{s.t. } \mathcal{R}(\boldsymbol{c}_1) \leq R_1 \\ &\qquad ... \\ &\qquad \mathcal{R}(\boldsymbol{c}_m) \leq R_m, \end{aligned} \tag{1}$$

where $\mathcal{R}(\boldsymbol{c}_k)$ representing resource measurement of platform k with channel configuration $\boldsymbol{c}_k$. Note that m platforms use a single network with shared parameters $\boldsymbol{w}$, and we want to optimize for $\boldsymbol{c}_k$ and $\boldsymbol{w}$ at the same time.

The optimization of this objective is dependent on *a priori* defined weights μ_k specifying the relative importance of a given weight. In order to explore the impact of these weights under different regimes, we vary these parameters in our experiments. Motivated by the empirical good fit of mobile phone markets with a Chinese restaurant process, we have selected a range of settings computed from such a distribution.

3.1 Chinese restaurant process

A Chinese restaurant process [1] is a discrete-time stochastic process, modelling people's behavior in a Chinese restaurant where customers do not know each other but share a table. Specifically, in a Chinese restaurant with an infinite number of tables and each with infinite capacity, the first customer sits at the first table, and the next customer either sits at the first table or picks a vacant table. At time $t+1$, the $t+1$th customer arrives and there are $|B|$ occupied tables, then he would find a new table with probability $\frac{\theta+\alpha|B|}{t+\theta}$ or at an occupied table b of size $|b|$ with probability $\frac{|b|-\alpha}{t+\theta}$ where strength parameter α and discount parameter θ should be either $\alpha < 0$ and $\theta = -L\alpha$ for $L \in \mathbb{N}^*$ or $0 \leq \alpha < 1$ and $\theta > -\alpha$.

As Figure 1 shows, the mobile vendor market share in USA in October 2020 is very similar to a realization of a Chinese restaurant process with $\alpha = 0.3$ and $\theta = 0.15$. We use the Chinese restaurant process as a model for economic importance μ_k.

3.2 Stochastic gradient descent strategies

We follow the assumption in [2] that the loss can be decomposed over individual channel choices and the resource measurement can be written as pairs of successive terms. That is

$$\mathcal{L}(\boldsymbol{c}, \boldsymbol{w}) = \sum_{i=1}^{n-1} \mathcal{L}_i(c_i, \boldsymbol{w}) \tag{2}$$

$$\mathcal{R}(\boldsymbol{c}) = \sum_{i=0}^{n-1} \mathcal{R}_i(c_i, c_{i+1}). \tag{3}$$

The intuition for (2) is to use a sliding window to average per-channel errors when a specific channel number c_i is chosen during slimmable network training; for (3), they develop a black-box latency model and here we use FLOPs which can be expressed in closed form as a special case of their latency model.

From Equations (2) and (3), and considering the Lagrangian of (1), we have

$$\begin{aligned}
&\min_{\boldsymbol{w}} \sum_{k=1}^{m} \min_{\boldsymbol{c}_k} \Bigg(\mu_k \sum_{i=1}^{n-1} \mathcal{L}_i(c_{k,i}, \boldsymbol{w}) + \\
&\qquad \gamma_k \Big(\sum_{i=1}^{n} R_i(c_{k,i-1}, c_{k,i}) - R_k \Big) \Bigg) \\
&= \min_{\boldsymbol{w}} \sum_{k=1}^{m} \nu_k \min_{\boldsymbol{c}_k} \Bigg(\frac{\mu_k}{\nu_k} \sum_{i=1}^{n-1} \mathcal{L}_i(c_{k,i}, \boldsymbol{w}) + \\
&\qquad \frac{\gamma_k}{\nu_k} \Big(\sum_{i=1}^{n} R_i(c_{k,i-1}, c_{k,i}) - R_k \Big) \Bigg)
\end{aligned} \tag{4}$$

where ν_k is a predefined scale factor for platform k and $\sum_{k=1}^{m} \nu_k = 1$. Note that since the inner minimization problem boils down to single-platform and is already decomposed as layer-wise terms, we can follow the approach in [2] and model it as pairwise Markov random filed (MRF) which can be solved efficiently with Viterbi inference.

Then we use the approach in [2] by smoothing the min operation by a log-sum-exp operation in the Viterbi forward pass. The messages sent from variable c_i to c_{i+1} then becomes

$$\begin{aligned}
m(c_{i+1}) = \log \sum_{c_i} \exp -\frac{1}{T} \big(m(c_i) + \\
\frac{\mu_k}{\nu_k} \mathcal{L}_i(c_{i+1}) + \frac{\gamma}{\nu_k} \mathcal{R}_i(c_i, c_{i+1}) \big)
\end{aligned} \tag{5}$$

where T is a temperature parameter that affects the smoothness of the relaxation. In particular, the outputs of the smoothed Viterbi inference are probabilities p_i for each layer.

Then we can iteratively solve (4) by first choosing a platform k with probability ν_k, and then perform stochastic gradient descent with sandwich rule while using smoothed Viterbi to decide the sampling probabilities of the random configuration.

4 Experiments

4.1 Setups

We perform our experiments on CIFAR-100 with MobileNetV1 [7]. In the search phase, we set the width range to $[0.2, 1.5]$ and consider up to 14 channel choices per layer. We consider 7 platforms each with a target FLOPs equivalent to $0.25\times, 0.375\times, 0.50\times, 0.625\times, 0.75\times, 0.875\times, 1.0\times$ of the original MobileNetV1. Then we use a Chinese restaurant process (CRP) with varying parameters to determine the weights of the remaining 5 platforms. Additionally, we add two additional sets of weights that are less skewed than those generated by CRP to make the distributions more diverse (Table 1), and we have 6 different weight distributions in total.

Platform	#1	#2	#3	#4	#5	#6	#7
Width	0.25×	0.375×	0.50×	0.625×	0.75×	0.875×	1.0×
S	0.25	0.005	0.010	0.045	0.130	**0.310**	0.25
RS	0.25	**0.310**	0.130	0.045	0.010	0.005	0.25
U	0.25	0.100	0.100	0.100	0.100	0.100	0.25
M	0.25	0.025	0.0625	**0.325**	0.0625	0.025	0.25
L	0.25	0.005	0.010	0.020	0.040	**0.425**	0.25
RL	0.25	**0.425**	0.040	0.020	0.010	0.005	0.25
FLOPs (M)	3.35	7.34	12.21	19.02	26.58	36.20	46.45

Table 1: Weights and targets of 7 platforms. Weights of #1 and #7 are all 0.25. S: weights of #2 – #6 are generated by a CRP ($\alpha = 0.3, \theta = 0.15$) and the peak is at #6; RS: weights of #2 – #6 are reversed and the peak is at #2; U: weights of #2 – #6 are the same (uniform distribution); M: the peak is at #4 and does not correspond to a CRP; L: weights of #2 – #6 are generated by a CRP ($\alpha = 0.15, \theta = 0.15$) and the peak is at #6; RL: weights of #2 – #6 are reversed and the peak is at #2.

In the search phase, we first train a slimmable network (with non-uniform scaling) using a constant learning rate of 0.1 for 50 epochs and we choose $k = 1$ to reduce the computational cost. We call the model trained with uniform sampling as OWS and with sampling probabilities decided by smoothed Viterbi as AOWS. For AOWS, we use a warm-up epoch of 10, uniform sampling, and anneal the temperature with a piece-wise linear function. After that, we use Viterbi inference to find the optimal channel numbers for each platform.

In the final re-train phase, we train a slimmable network with widths found during the search phase, as well as a slimmable network with the original configurations, at widths of $[0.25\times, 0.375\times, 0.50\times, 0.625\times, 0.75\times, 0.875\times, 1.0\times]$. Note that all of them have 7 different widths in total and each counterpart has the same number of FLOPs. Inspired by biased training of AOWS, we also experiment with both uniform sampling and sampling platform #2 - #5 according to their weights. We train the model for 300 epochs with a batch size of 128, and use SGD with an initial learning rate of 0.1 which is decayed by 0.985 at each epoch. Nesterov momentum is set to 0.9 and weight decay is 0.0005. A dropout rate of 0.6 is applied only to the maximum configuration [16].

In the sequel we use Slim-U to represent a slimmable network with original configurations and trained with uniform sampling, and Slim-W for some weight distribution, where the random configurations are sampled according to W. The naming convention for OWS and AOWS methods are similar, except that we sometimes add a prefix for AOWS such as RS-AOWS-U, meaning that in the search phase we sample different platforms according to RS to perform smoothed Viterbi, and in the retrain phase, we sample random configurations according to U.

4.2 Main results

The average accuracy weighted by different weight distribution is shown in Table 2. Generally, training a slimmable network with channel numbers found by OWS and AOWS achieves a better result than that with the original channel configurations. For some distributions such as S, RS and L, AOWS beats OWS, but for the others OWS is better.

Interestingly, the original slimmable network, as well as that whose channels are found by AOWS generally benefits from sampling random configurations according to platform weights instead of simple uniform sampling. By contrast, the slimmable network with channel numbers found by OWS have worse results with non-uniform sampling.

4.3 Comparison with the original channel configurations

We first compare channel numbers of platform #7 found by OWS with the original MobileNet-v1

Weight distribution	Slim-U	Slim-W	OWS-U	OWS-W	AOWS-U	AOWS-W
S	65.03	66.14	66.50	66.48	66.30	**66.62**
RS	63.70	63.48	65.22	64.52	65.16	**65.31**
U	64.44	64.44	**65.96**	65.96	65.94	65.94
M	64.57	65.24	**66.14**	65.55	65.48	65.85
L	65.10	65.72	66.55	66.25	66.28	**66.56**
RL	63.48	64.10	**65.03**	64.83	64.39	64.53

Table 2: The **average** top-1 validation accuracy (%) on CIFAR-100 with different weight distributions. Slim-U: a slimmable network with original channel configurations and trained with uniform sampling; Slim-W: sampling according to platform weights; OWS-U: a slimmable network with channels found by Viterbi inference and samples according to uniform distribution; OWS-W: sampling according to platform weights; AOWS-U: a slimmable network with channels found by smoothed Viterbi and samples according to uniform distribution; AOWS-W: sampling according to platform weights.

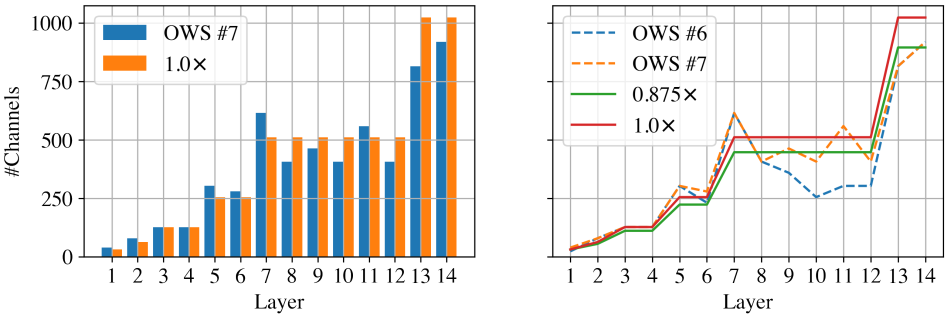

Figure 2: Left: the number of channels of platform #7 found by OWS vs. the original MobileNet-v1; right: the number of channels of platforms #6 and #7 found by OWS vs. the original MobileNet-v1.

(AOWS and other platforms show a similar pattern). It can be seen from Figure 2 that OWS put more channels in shallow layers but it becomes thinner in deep layers. In contrast with [2], their experiments on ImageNet show that OWS and AOWS use more channels in deep layers. We conjecture that it is because CIFAR-100 has much smaller resolution than ImageNet (32×32 vs. 224×224) and using more channels in the shallow layers help to propagate more information to deep ones.

We then investigate the interaction among different platforms. In Figure 2 we show channel numbers found by OWS for platform #6 and #7 vs. the original ones, while we observe a similar behavior for other platforms. Configurations found by OWS are more correlated than the original ones. Indeed, 8 out of 14 layers use the same channel numbers. This might explain why a slimmable network performs better when trained with channel numbers found by OWS - it is not only because the channel configuration of every single platform is superior to the original one, but also when training an ensemble of models with shared weights at the same time, the correlation between child models can help with optimization.

To verify this, we train MobileNet-v1 separately with the original configurations as well as those found by OWS, and then average the performance of these stand-alone models by U. The result is shown in Table 3. The average accuracy for OWS is 0.85% higher than the original configurations, but according to Table 2, when training a group of models as a slimmable network, this difference is 1.52%. This sheds light on neural architecture search (NAS) methods in general: when searching for an ensemble of models that are trained at the same time, a NAS method should not only consider models that perform well when trained alone, but also take the correlation among different models into account.

Configuration	Original	OWS
Average accuracy (%)	63.46	64.31 (**+0.85**)

Table 3: Top-1 validation accuracy (%) weighted **average** by U.

4.4 Comparison of OWS with AOWS

Generally, channel numbers found by OWS and AOWS are very similar. As a result, although AOWS beats OWS in some weight distributions such as S, RS and L, the difference is very small. However, as opposed to OWS whose channel configurations are independent of the weight distribution, AOWS finds a set of channel numbers for each weight distribution. These, as well as its dependence on temperature scheduling, which we have not tuned extensively, make AOWS less attractive.

On the other hand, in contrast with OWS, biased sampling improves AOWS's performance for all weight distributions. We investigate their difference in Figure 3 using RS-AOWS as an example. AOWS sometimes finds more correlated channel configurations than OWS, and uses more channels in the last two layers. We believe it is these large channel numbers that need more time to train, due to this non-uniformity.

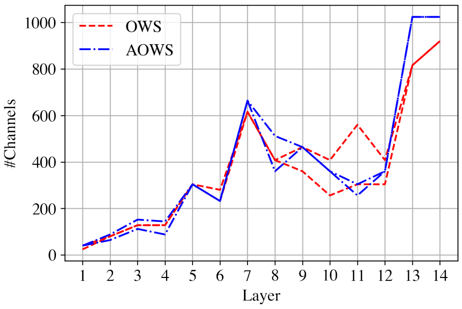

Figure 3: The number of channels of platforms #6 and #7 found by OWS (red) vs. RS-AOWS (blue).

5 Related Work

Platform-aware NAS methods [2, 4, 5, 12, 13, 14] try to optimize a performance-speed trade-off either by enforcing platform-dependent constraint (FLOPS, latency), or incorporating them into the objective and jointly optimizing for accuracy and efficiency. However, for each platform profile, these methods need to repeat the training process to obtain a new set of weights, and therefore are not scalable to scenarios where there are a large number of platforms of different capcities.

To deal with this, Once-for-All (OFA) [3] proposes to train a super-network and then use a so-called progressive shrinking algorithm that generalize the pruning method to depth, width, kernel size and resolution, and the pruned child-models can meet targets of different platforms. BigNAS [16] extends the slimmable networks [15, 17], along with other training tricks, so as to train a single network which is able to operate at different configurations (depth, width, kernel size and resolution) without any extra post-processing.

6 Conclusions

In this paper we focus on optimizing slimmable networks with multiple target platforms. We formalize this problem as maximizing a weighted average accuracy over different platforms and generate these weights with a Chinese restaurant process using our insight from a real distribution. Then we use a pairwise Markov random field framework to efficiently find the optimal channel configurations for each platform, and we also extend AOWS [2] to the multi-platform setting so as to utilize an adaptive sampling strategy. After that, we train a slimmable network with the found channel numbers and evaluate our approach on CIFAR-100 with various weight distributions and sampling strategies. The experiments show that our method improve the average accuracy over different weight distributions.

Acknowledgements

We acknowledge support from the Research Foundation - Flanders (FWO) through project number G0A1319N, and funding from the Flemish Government under the Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen programme.

References

[1] D. Aldous. *Exchangeability and related topics.* Springer, 1985.

[2] M. Berman, L. Pishchulin, N. Xu, M. Blaschko, and G. Medioni. AOWS: Adaptive and Optimal Network Width Search with Latency Constraints. *CVPR*, 2020.

[3] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-All: Train One Network and Specialize it for Efficient Deployment. *ICLR*, 2020.

[4] B. Chen, P. Li, B. Li, C. Lin, C. Li, M. Sun, J. Yan, and W. Ouyang. BN-NAS: Neural Architecture Search with Batch Normalization. *ICCV*, 2021.

[5] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda, and J. E. Gonzalez. FBNetV3: Joint Architecture-Recipe Search using Predictor Pretraining. *CVPR*, 2021.

[6] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le. Searching for MobileNetV3. *ICCV*, pages 1314–1324, 2019.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv*, 2017.

[8] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. *ECCV*, 2018.

[9] A. Mensch and M. Blondel. Differentiable dynamic programming for structured prediction and attention. *ICML*, 2018.

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *CVPR*, pages 4510–4520, 2018.

[11] StatCounter. Mobile vendor market share United States of America. `https://gs.statcounter.com/vendor-market-share/mobile/united-states-of-america`. Accessed: 2020-11-11.

[12] M. Tan and Q. V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *ICML*, 2019.

[13] M. Tan and Q. V. Le. EfficientNetV2: Smaller Models and Faster Training. *ICML*, 2021.

[14] J. Yu and T. Huang. AutoSlim: Towards One-Shot Architecture Search for Channel Numbers. *NeurIPS Workshop*, 8, 2019.

[15] J. Yu and T. S. Huang. Universally slimmable networks and improved training techniques. *ICCV*, pages 1803—1811, 2019.

[16] J. Yu, P. Jin, H. Liu, G. Bender, P.-J. Kindermans, M. Tan, T. Huang, X. Song, R. Pang, and Q. Le. BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models. *ECCV*, 2020.

[17] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang. Slimmable neural networks. *ICLR*, 2019.