Width transfer: on the (in)variance of width optimization

Ting-Wu Chin[1]*, Diana Marculescu[12], Ari S. Morcos[3]

Carnegie Mellon University[1], The University of Texas at Austin[2], Facebook AI Research[3]

tingwuc@andrew.cmu.edu, dianam@utexas.edu, arimorcos@fb.com

Abstract

Optimizing the channel counts for different layers of a CNN has shown great promise in improving the efficiency of CNNs at test-time. However, these methods often introduce large computational overhead (e.g., an additional $2\times$ FLOPs of standard training). Minimizing this overhead could therefore significantly speed up training. In this work, we propose width transfer, a technique that harnesses the assumptions that the optimized widths (or channel counts) are regular across sizes and depths. We show that width transfer works well across various width optimization algorithms and networks. Specifically, we can achieve up to $320\times$ reduction in width optimization overhead without compromising the top-1 accuracy on ImageNet, making the additional cost of width optimization negligible relative to initial training. Our findings not only suggest an efficient way to conduct width optimization, but also highlight that the widths that lead to better accuracy are invariant to various aspects of network architectures and training data.

1. Introduction

Better designs for the number of channels for each layer of a convolutional neural network (CNN) can lead to improved test performance for image classification without requiring additional floating-point operations (FLOPs) during the forward pass at test time. Specifically, by optimizing the channel widths, improvements of up to 2% top-1 accuracy for image classification on ImageNet can be achieved without additional FLOPs [10, 9, 30, 4, 17]. However, designing the layer by layer width multipliers for efficient CNNs is a non-trivial task that often requires intuition and domain expertise together with trial-and-error to do well. To alleviate the labor-intensive trial-and-error procedure, width optimization algorithms have been proposed [20, 13, 10, 8, 9] to automatically determine the width of a convolutional neural network. A width optimization algorithm takes as input an initial network and a training dataset, and outputs

a set of optimized widths for each layer. When these optimized widths are applied to the initial network and the network trained from scratch, one can achieve better validation accuracy compared to training a network of the original widths without incurring additional test-time FLOPs. Such algorithms can be seen as neural architecture search algorithms that search for layer-wise channel counts to maximize validation accuracy subject to test-time FLOPs constraints. In contrast to channel pruning, which seeks to reduce FLOPs without reducing accuracy, width optimization instead aims to improve the accuracy while maintaining the same number of FLOPs.

However, these methods often add a large computational overhead necessary for the width optimization procedure. Concretely, even for efficient methods that use differentiable parameterization [10], width optimization takes an additional $2\times$ the training time. To contextualize this overhead, using distributed training on 8 V100 GPUs, it takes approximately 100 GPU hours to train a ResNet50 on the ImageNet dataset [26]. Including the width optimization overhead, it therefore takes 300 GPU hours for both width optimization using differentiable methods [10] and training the optimized ResNet50. Additionally, width optimization algorithms are often parameterized by some target test-time resource constraints, *e.g.*, FLOPs. As a result, the computational overhead scales linearly with the number of target constraint levels considered, which can be exceedingly time-consuming for optimizing CNNs for embodied AI applications [3]. Reducing the overhead for width optimization, therefore, would have material practical benefits.

Fundamentally, one of the key reasons why width optimization is so costly is due to its limited understanding by the community. Without assuming or understanding the structure of the problem, the best hope is to conduct black-box optimization whenever training configurations, datasets, or architectures are changed. In this work, we take the first step to empirically understand the structure underlying the width optimization problem by changing network architectures and the properties of training datasets, and observing how they affect width optimization.

If similar inputs to the width optimization algorithms re-

---

*This work is mostly done when Ting-Wu works as a research intern at Facebook AI Research.
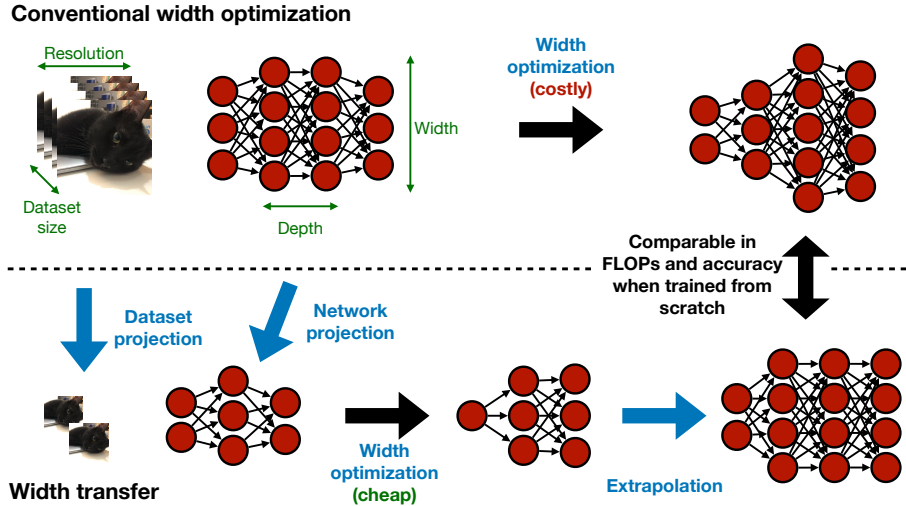
Figure 1: The top row shows the conventional width optimization approach, which takes a training dataset and a seed network, and outputs a network with optimized widths. The bottom row depicts our idea of width transfer, where width optimization operates on the down-scaled dataset and seed network. We then use a simple function to extrapolate the optimized architecture to match the original network. Compared to direct width optimization, our empirical findings suggest that width transfer has similar performance, but has the benefit of drastically lower overhead.

sult in similar outputs, one can exploit this commonality to reduce the width optimization overhead, especially if the two input configurations have markedly different FLOPs requirements. As a concrete example, if optimizing the widths of a wide CNN (high FLOPS) and a narrow CNN (low FLOPs) results in widths that differ only by a multiplier, one can reduce the computational overhead of width optimization by computing widths for the low FLOPS, narrow CNN and adjusting them to accommodate the high FLOPs, wide CNN. To study the aforementioned commonality, we propose *width transfer*, a novel paradigm for efficient width optimization. In width transfer, one first projects the network and the dataset to their smaller counterparts, then one executes width optimization with the smaller network and dataset, and finally one extrapolates the optimized result back as shown schematically in Figure 1.

In other words, we would like to understand if the following invariances hold for optimized widths:

1. **Sample size**: The optimized widths are minimally affected by the size of the dataset when the dataset's distribution is approximately identical (*i.e.*, uniform down-sampling in a class-balanced fashion).

2. **Spatial resolution**: The optimized widths are merely affected by the image resolutions.

3. **Channel magnitude**: The ratios between the optimized widths and the un-optimized ones are roughly constant regardless of the absolute channel counts of the un-optimized network.

4. **Within-stage channel counts**: The optimized widths are similar when they belong to the same stage of a network where stage is defined by grouping the blocks with the same input resolution [11].

Based on a comprehensive empirical analysis, we provide the following contributions:

- We propose *width transfer*, a novel paradigm for efficient width optimization. Additionally, we propose two novel layer-stacking methods to transfer width across networks with different layer counts.

- We find that the optimized widths are highly transferable across network's initial width and depth, and across dataset's sample size and resolution.

- We demonstrate a practical implication of the previous finding by showing that one can achieve $320\times$ reduction in width optimization overhead for a scaled-up MobileNetV2 and ResNet18 on ImageNet with similar accuracy improvements, effectively making the cost of width optimization negligible relative to initial model training.

- With controlled hyperparameters over multiple random seeds on a large-scale image classification dataset, we verify the effectiveness of width optimization methods proposed in prior art. This is in contrast with prior work which borrows numbers from other papers that might not have the same training hyperparameters. However, we also find that, for a deeper ResNet

on ImageNet, width optimization has limited benefits (Fig. 4c).

2. Width optimization

The layer-by-layer widths of a deep CNN are often regarded as a hyperparameter optimized to improve classification accuracy. Specifically, the width multiplier method [14] was introduced in MobileNet to arrive at models with different FLOPs and accuracy profiles and has been widely adopted in many papers [12, 28, 2, 7].

Besides simply scaling the width uniformly for all the layers to arrive at CNNs with different FLOPs, optimizing the width for each layer has received growing interest recently as a means to improve the efficiency of deployed deep CNNs. A width optimization algorithm, $\mathcal{A}$, takes in a training configuration, $\mathcal{C} = (\mathcal{D}, \mathcal{N})$, and outputs a set of optimized widths, $\boldsymbol{w}^*$. $\mathcal{C}$ consists of initial network, $\mathcal{N}$, and training dataset, $\mathcal{D}$. Concretely, the goal of $\mathcal{A}$ is to solve the following optimization problem:

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} Acc_{\text{val}} \left(\theta(\mathcal{N} \times \boldsymbol{w}, \mathcal{D}), \mathcal{N} \times \boldsymbol{w} \right)$$
$$\text{s.t. FLOPs}(\mathcal{N}) = \text{FLOPs}(\mathcal{N} \times \boldsymbol{w}), \quad (1)$$

where $\boldsymbol{w}$ is a width vector with L dimensions, where L is the number of convolutional layers. $\mathcal{N} \times \boldsymbol{w}$ is applying width $\boldsymbol{w}$ to a network $\mathcal{N}$ by scaling the number of channels for layer i from F_i to $w_i F_i$. $\theta()$ is the standard training procedure that takes in a CNN and a training dataset and outputs trained weights, *e.g.*, stochastic gradient descent with a fixed training epoch. Lastly, Acc_{val} is a function that evaluates the validation accuracy given the trained weights and the CNN architecture.
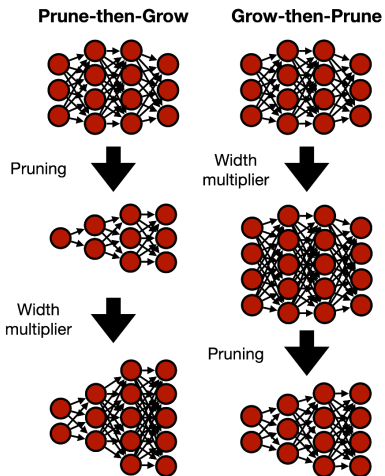

Figure 2: The two width optimization strategies proposed in prior art.

To optimize the width of a CNN, there are in general two approaches proposed in prior art: *Prune-then-Grow* and *Grow-then-Prune*. *Prune-then-Grow* uses channel pruning methods to arrive at a down-sized CNN with non-trivial layer-wise channel counts followed by re-growing the CNN to its original FLOPs using the width multiplier method [9]. On the other hand, *Grow-then-Prune* first uses the width multiplier method to enlarge the CNN followed by channel pruning methods to trim down channels to match its pre-grown FLOPs [30, 10]. The aim of both of these methods is to improve accuracy of the network while maintaining a given FLOPs count. The schematic view of the two approaches is visualized in Figure 2.

While there are many papers on channel pruning [15, 23], they mostly focus their analysis on decreasing the inference-time FLOPs of pre-trained models whereas we focus on improving the classification accuracy of a network by optimizing its width while holding inference-time FLOPs constant. While one can use either the *Prune-then-Grow* or *Grow-then-Prune* strategies to arrive at a CNN of equivalent FLOPs, it is not clear if such strategies generally improve performance over the un-optimized baseline as it is not verified in most channel pruning papers. As a result, in this paper, we focus on analyzing algorithms that have demonstrated the effectiveness over the baseline (uniform) width configurations in either *Prune-then-Grow* or *Grow-then-Prune* settings.

3. Approach

3.1. Width optimization methods

Theoretically, we only care about algorithms $\mathcal{A}$ that "*solve*" the width optimization problem (equation 1). However, the problem is inherently combinatorially hard. As a result, we use state-of-the-art width optimization algorithms as probes to understand the width optimization problem. More specifically, we consider methods that have reported improved accuracy compared to the un-optimized network given the same test-time FLOPs, and have publicly available code to ensure correctness of implementation. With these criteria, we consider MorphNet [9], AutoSlim [30], and DMCP [10]. Note that we use these algorithms to find the "network architecture" which will be trained from scratch using the normal training configurations with randomly initialized weights.

3.2. Projection and extrapolation

Projection In projection, there are two aspects: network projection and dataset projection. For network projection, we propose to use the width multiplier to uniformly shrink the channel counts for all the layers to arrive at a narrower model. Additionally, we also propose to use the depth multiplier to uniformly shrink the block counts per each stage of a neural network to arrive at a shallower model. For dataset projection, on the one hand, we propose to sub-sample the
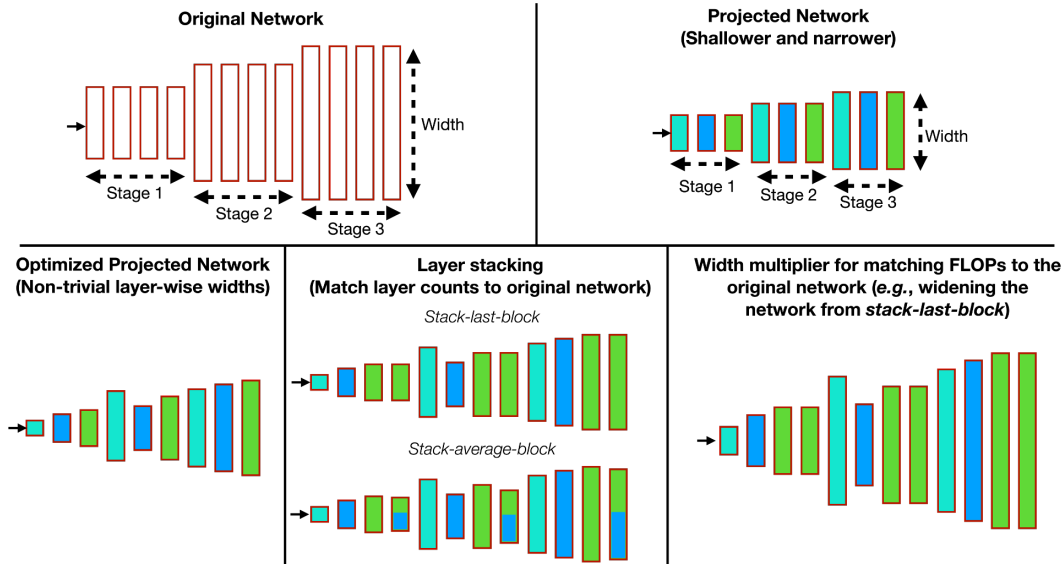
Figure 3: An example for extrapolation. The projected network has fewer layers and channel counts per layer compared to the original network. After width optimization on the projected network, we propose two methods, *i.e.*, stack-last-block and stack-average-block, to match the layer counts to the original network. Finally, we match the FLOPs to the original network with a width multiplier.

training sample in a dataset in a class-balanced way. On the other hand, we propose to sub-sample the spatial dimension of the training images to arrive at images with lower resolutions. When keeping the width optimization algorithm fixed, *i.e.*, training the input network with a certain batch size and training epochs using the input dataset, all the aforementioned projections immediately result in width optimization overhead reduction.

Extrapolation We consider two aspects for extrapolation: dimension-matching and FLOPs-matching, which are schematically shown in Figure 3. First, we want the extrapolated network to have the same number of layers as the original network. This is particularly crucial when the original network is projected in the depth dimension, in which case we propose two layer stacking strategies:

- **Stack-last-block**: Stack the width multipliers of the last block of each stage until the desired depth is met. A stage includes convolutional blocks with the same output resolution in the original network. A convolutional block consists of several convolutional layers such as the bottleneck block in ResNet [11] and the inverted residual block in MobileNetV2 [27].

- **Stack-average-block**: To avoid mismatches among residual connection, we exclude the first block of each stage and compute the average of the width multipliers across all the rest blocks in a stage, then stack the average width multipliers until the desired depth is met.

Note that since existing network designs share the same channel widths for all the blocks in each stage, the above two layer stacking strategies will have the same results when applied to networks with un-optimized widths.

Second, we want to extrapolate the optimized projected network to a larger one such that it has the FLOPs of the original network. To do so, we propose to use the width multiplier to widen the optimized width. This procedure basically assumes that what determines the optimized widths is the ratio among layers and we show that this assumption is largely correct in Section 4 as the optimized widths are largely transferable across network's initial widths.

4. Experiments

In this section, we empirically investigate the transferability of the optimized widths across different projection and extrapolation strategies. Specifically, we study projection across architectures by evaluating different widths and depths as well as across dataset properties by dataset sub-sampling and resolution sub-sampling. In addition to analyzing each of these four settings independently, we also investigate a compound projection that involves all four jointly. To measure the transferability, we plot the ImageNet top-1 accuracy of network obtained by direct optimization and width transfer across different projection scales that have different width optimization overhead. Width optimization overhead refers to the FLOPs needed to carry out width optimization. If transferable, we should observe

a horizontal line across different width optimization overheads, suggesting that performance is not compromised by deriving the optimized widths from a smaller FLOP configuration. Moreover, we also plot the ImageNet top-1 accuracy for the un-optimized baseline to characterize whether width optimization or width transfer is even useful for some configurations.

4.1. Experimental Setup

We used the ImageNet dataset [6] throughout the experiments. Unless stated otherwise, we use 224 input resolution. For CNNs, we considered the meta-architecture of ResNet18 [11] and MobileNetV2 [27]. Note that we adjusted the depth and width of ResNet-18 and MobileNetV2 to arrive at a wide variety of models for our width transfer study. Models were each trained on a single machine with 8 V100 GPUs for all the experiments. The width multiplier method applies to all the layers in the CNNs while the depth-multiplier excludes the first and the last stage of MobileNetV2 as there is only one block for each of them. After we obtained the optimized architecture, we trained the corresponding network from scratch with random initialization using the same hyperparameters to analyze their performance. We repeated each experiment three times with different random seeds and reported the mean and standard deviation. Other hyperparameters are detailed in Appendix A.

4.2. Projection: width

Here, we focus on answering the following question: "*Do optimized widths obey the channel magnitude invariance?*" The answer to this question is unclear from existing literature as the current practice is to re-run the optimization across different networks [10, 9, 20]. If the optimized widths are similar across different initial widths, this suggests that the quality of the vector of channel counts are scale-invariant given the current practice of training deep CNNs and the dataset. Additionally, it also has practical benefits where one can use width transfer to reduce the overhead incurred in width optimization. On the other hand, if the optimized widths are dissimilar, this suggests that not only the direction of the vector of channel counts is important, but also its magnitude. That is, for different magnitudes, we may need different orientations. In other words, this suggests that existing practice, though costly, is empirically proved to be necessary.

To empirically study the aforementioned question, we considered the source width multipliers of {0.312, 0.5, 0.707, 1, 1.414, 1.732} and the target width multiplier of 1.732, and we analyzed if the source optimized architecture transfers to the target architecture. The set of initial width multipliers is chosen based on square roots of width optimization overhead. We analyzed the transferability in the accuracy space. In Figure 4a and 4b, we plot the Ima-

(a) Res18, width   (b) MBv2, width
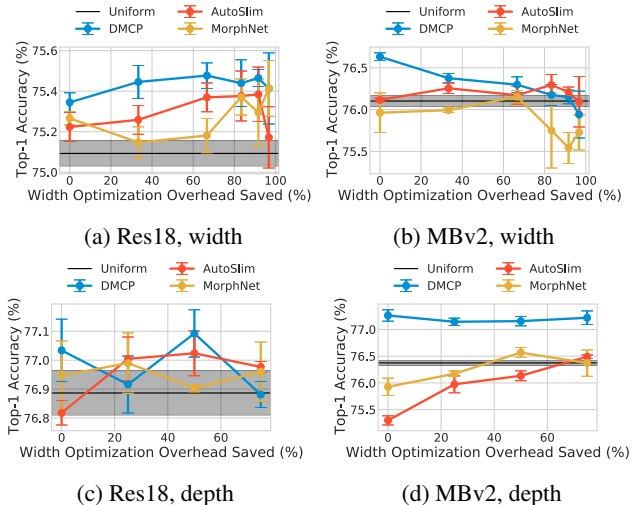
(c) Res18, depth   (d) MBv2, depth

Figure 4: Experiments for width transfer under network projection. We plot the ImageNet top-1 accuracy for uniform baseline, width transfer, and direct optimization (the leftmost points). On the x-axis, we plot the width optimization overhead saved by using width transfer.

geNet top-1 accuracy for the baseline (a $1.732\times$ wide network) and networks obtained by direct optimization and width transfer. For ResNet18, the width optimization overhead can be saved by up to 96% for all three algorithms without compromising the accuracy improvements gained by the width optimization.

On MobileNetV2, AutoSlim and MorphNet can transfer well and save up to 80% width optimization overhead. While DMCP for MobileNetV2 results in 0.4% top-1 accuracy loss when using width transfer, the transferred width can still outperform the uniform baseline, which is encouraging for applications that allow such accuracy degradation in exchange for 83% width optimization overhead savings. More specifically, that would reduce compute time from 160 GPU-hours all the way to 30 GPU-hours for MobileNetV2 measured using a batch size of 1024, a major saving. Our results suggest that a good orientation for the optimized channel vector continues to be suitable across a wide range of magnitudes.

Since the optimized widths are highly transferable, we are interested in the resulting widths for both CNNs. We find that the later layers tend to increase a lot compared to the un-optimized ones. Concretely, in un-optimized networks, ResNet18 has 512 channels in the last layer and MobileNetV2 has 1280 channels in the last layer. In contrast, the average optimized width has 1300 channels for ResNet18 and 3785 channels for MobileNetV2. We visualize the average widths for ResNet18 and MobileNetV2 (average across optimized widths) in Figure 5.

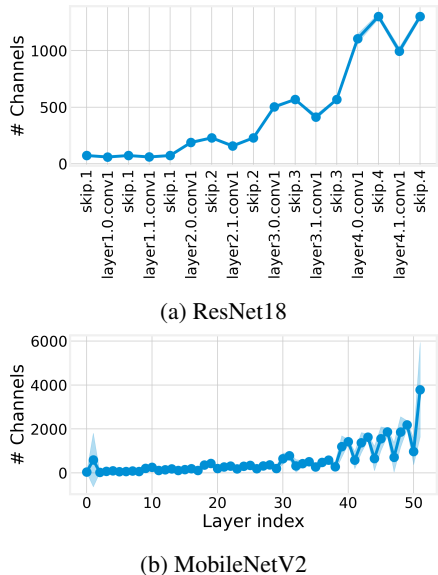(a) ResNet18

(b) MobileNetV2

Figure 5: The average optimized width for ResNet18 and MobileNetV2. They are averaged across the optimized widths. We plot the mean in solid line with shaded area representing standard deviation.

4.3. Projection: depth

Next, we asked whether networks with different initial depths share common structure in their optimized widths. Specifically, "*Do the optimized widths obey the within-stage channel counts invariance?*" Because making a network deeper will add new layers with no corresponding optimized width, we will need a mechanism to map the vector optimized widths to a vector with far more elements. Here, we first compared the two layer-stacking methods proposed in Section 3.2 using DMCP for ResNet18 and MobileNetV2. As shown in Figure 6, both *stack-last-block* and *stack-average-block* layer stacking strategies perform similarly. As a result, we use *stack-average-block* for all other experiments. We considered $\{1, 2, 3, 4\}$ as the source depth multipliers and use 4 as the target depth multiplier. Similar to the analysis done in Section 4.2, we analyzed the similarity in the accuracy space.

As shown in Figure 4c and 4d, we find that the optimized widths stay competitive via simple layer stacking methods and up to 75% width optimization overhead can be saved if we were to optimize the width using width transfer for all three algorithms and two networks. This finding also suggests that the relative values of optimized widths share common structure across networks that differ in depth. In other words, the pattern of width multipliers across depth is scale-invariant. Interestingly, we find that width transfer *improves* direct optimization in terms of accuracy when it comes to AutoSlim and MorphNet as we see a positive slope
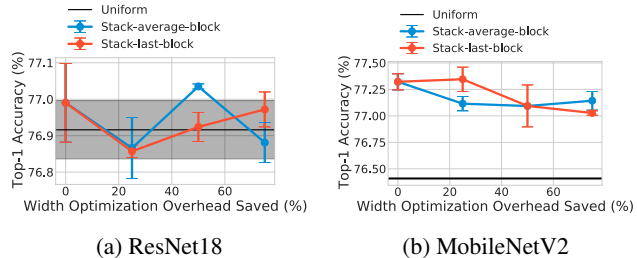

(a) ResNet18   (b) MobileNetV2

Figure 6: We compare the two layer-stacking strategies using DMCP for both ResNet18 and MobileNetV2. We can observe that both stack-average-block and stack-last-block perform similarly.

for these two methods on both networks. We conjecture that this is due to both AutoSlim and MorphNet are affected more by the dimensionality[1] of the problem (the number of widths to be learned), and that the within-stage channel invariance largely holds.

4.4. Projection: resolution

The input resolution and the channel counts of a CNN are known to be related when it comes to the test accuracy of a CNN. As an example, it is known empirically that a wider CNN can benefit from inputs with a higher resolution than a narrower net can [28]. As a result, it is not clear *if the optimized widths obey the spatial resolution invariance.* If the optimized widths indeed obey the spatial resolution invariance, this suggests that although wider networks benefit more from a higher resolution inputs, the non-uniform widths that result in better performance are similar. On the other hand, if the optimized widths are different, it suggests that, when it comes to the test accuracy, the relationship between channel counts and input resolution is more involved than the level of over-parameterization.

To study the aforementioned question, we considered the input resolution for the source to be $\{64, 160, 224, 320\}$ and choose a target of 320. As shown in Figure 7a and 7b, we find that except for MorphNet targeting ResNet18, all other algorithm and network combinations can achieve up to 96% width optimization overhead savings with the optimized widths that are still better than the uniform baseline. By saving 75% width optimization overhead, we can stay close to the performance obtained via direct optimization. Interestingly, we find that MorphNet had a very different optimized widths when transferred from resolution 64 for ResNet18, which leads to the worse performance for ResNet18 compared to direct optimization. The similarity among the optimized widths are detailed in Figure 10 in Appendix.

---

[1]Width depth projection, we effectively reduce the dimensionality of the search problem.

(a) Res18, resolution   (b) MBv2, resolution

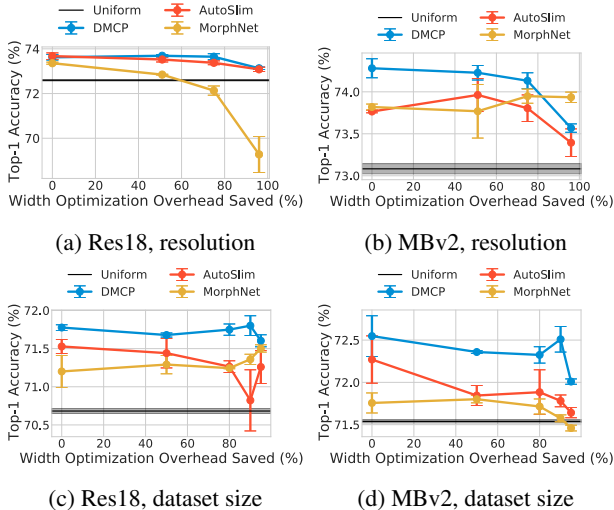(c) Res18, dataset size   (d) MBv2, dataset size

Figure 7: Experiments for width transfer under dataset projection. We plot the ImageNet top-1 accuracy for uniform baseline, width transfer, and direct optimization (the leftmost points). On the x-axis, we plot the width optimization overhead saved by using width transfer.

4.5. Projection: dataset size

The dataset size is often critical for understanding the generalization performance of a learning algorithm. Here, we would like to understand *if the optimized widths obey the sample size invariance*. We considered subsampling the ImageNet dataset to result in a source of $\{5\%, 10\%, 20\%, 50\%, 100\%\}$ of the original training data. Similar to previous analysis, we tried to transfer the optimized widths obtained using the smaller configurations to the largest configuration, *i.e.*, 100% of the original training data. As shown in Figure 7c and 7d, widths optimized for smaller dataset sizes transfer well to large dataset sizes. That is, 95% width optimization overhead can be saved and still outperforms the uniform baseline for both networks. On the other hand, 90% width optimization overhead can be saved and can still match the performance of direct optimization for DMCP. This suggests that the amount of training data barely affects width optimization, especially for DMCP, which is surprising.

4.6. Compound projection

From previous analyses, we find that the optimized widths are largely transferable across various projection methods independently. Here, we further empirically analyzed if the optimized width can be transferable across compound projection. To do so, we considered linearly interpolating all four projection methods and analyzed if the width optimized using cost-efficient settings can transfer to the most costly setting. Specifically, let a tuple (width,
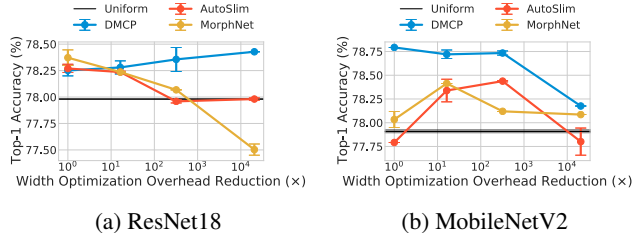

(a) ResNet18   (b) MobileNetV2

Figure 8: Width transfer with compound projection.

depth, resolution, dataset size) denote a training configuration. We considered the source to be $\{(0.312,1,64,5\%), (0.707,1,160,10\%), (1,1,224,50\%), (1.414,2,320,100\%)\}$ and the target to be $(1.414,2,320,100\%)$. As shown in Figure 8, the optimized width is transferable across compound projection. Specifically, we can achieve up to $320\times$ width optimization overhead reduction with width transfer for the best performing algorithm, DMCP. Additionally, it also suggests that the four projection dimensions are not tightly coupled for width optimization.

Applications to other target CNNs We considered using compound width transfer for ResNet50, ResNet101, and EfficientNetB3. For projection, we consider the width, depth, resolution and dataset size to be 0.707, 0.5, 160, and 20%, respectively. As shown in Table 1, up to $30\times$ wall-clock time reduction is achieved with less than 0.2% top-1 accuracy degradation. Considering a scenario where one wants to optimize the width of a network and train such a network for deployment. Width optimization reduce the overall training cost from $3\times$ to $1.06\times$. Such a huge optimization cost reduction can enable fast exploration for the benefits of width optimization for large models without paying the considerable costs.

4.7. Comparing to cheap pruning methods

While adopting state-of-the-art channel optimization directly can be costly, one may consider using cheap pruning methods and adopt a Prune-then-Grow strategy to carry out width optimization. We compare to magnitude-based channel pruning: network slimming (NS) [19] that prunes channels based on the magnitude of γ of the batch normalization layer. NS-xw-ye follows a three-step procedure: train an $x\times$ wider network for y epochs, prune the network with global γ ranking, and re-train the pruned network using full training schedule. The induced overhead for width optimization lies in the first step. The comparisons with NS for ResNet18 on ImageNet is shown in Fig. 9. Using DMCP directly is indeed the most expensive one, but it has the best performance. Our width transfer achieves similar performance compared to DMCP with overhead lower than magnitude-based pruning.

Network	Baseline	DMCP	Width transfer	Overhead (direct→width transfer)
ResNet50	77.97 ± 0.09	78.23 ± 0.11	78.07 ± 0.12	$37.4 \to 1.3$
ResNet101	79.43 ± 0.07	79.70 ± 0.05	79.54 ± 0.07	$66.7 \to 2.7$
EfficientNetB3	80.02 ± 0.09	80.24 ± 0.02	80.19 ± 0.10	$80 \to 3$

Table 1: Compound width transfer for other CNNs. Width optimization overhead measured with 8 NVIDIA V100 GPUS on a single machine.
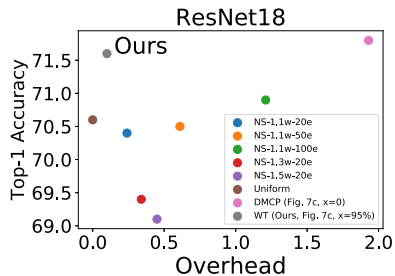

Figure 9: Comparing the proposed using DMCP with width transfer, DMCP, and network slimming.

5. Related work

Neural architecture search Our work is tightly connected to understanding the transferability of the searched results from neural architecture search (NAS) algorithms where the search space is determined by the layer-wise channel counts of a seed network. The transferability of NAS has been recently explored in several papers considering different search spaces and perspectives. Zoph *et al.*[32] have proposed to search the best cell on a small dataset and use the searched cell on a large dataset. Panda *et al.*[24] have analyzed the transferability of the solutions of various NAS algorithms in the DARTS search space [18] and have concluded that the design of the proxy datasets for the search has a great impact on the transferability of the searched result. Critically however, prior work has neglected to include the channel width multipliers in the search space, instead only focusing on proxy datasets [24, 32]. Consequentially, the relationship between optimized widths across different architectures has not been examined previously. Others [29, 1, 22] have analyzed the transferability of the search processes as opposed to the searched solutions. The key difference among the two is that the transfer of the search processes is algorithm-dependent while the transfer of the searched solutions is not. Given width optimization is a fast-developing field, we study the transferability in the solution space to have a more general result.

EcoNAS [31] is closely related to our work as it also systematically investigated several proxy training configurations for neural architecture search. However, the crucial difference between EcoNAS and ours is that, in our study, the number of channels is not only a projection dimension, but also the optimization variable. As a result, the extrapolation step is necessary for our study but not for EcoNAS. Consequentially, the proposed width transfer has a dimensionality reduction effect for hyperparameter optimization (as noted in Section 4.3) while EcoNAS does not. Lastly, the search space for EcoNAS is the cell-based search space while our channel search space is orthogonal to theirs and can be applied to any network for architectural fine-tuning.

Channel pruning Channel pruning is an active research topic for efficient network design. More specifically, channel pruning determines how we can prune an existing network in the channel dimension so as to retain the most accuracy [3, 12, 13, 15, 23, 16]. A channel pruning procedure often has the weights and the optimized channel counts coupled together. Inspired by Liu *et al.* [21], who had empirically shown the importance of the optimized channel counts, we take a step further by understanding the transferability of the searched channel counts across different input network and dataset transformations.

6. Conclusion

In this paper, we take a first step in understanding the transferability of the optimized widths across different width optimization algorithms and invariance dimensions. Our empirical analysis sheds light on the structure of the width optimization problem, which can be used to design better optimization methods. More specifically, by exploiting the channel magnitude and within-stage channel counts invariances, we not only can reduce the computational cost needed to width optimization, but also reduce the dimension of the optimization variables[2]. Per our analysis, we can achieve up to $320\times$ reduction in width optimization overhead without compromising the top-1 accuracy on ImageNet.

Acknowledgement

---

[2]In the channel search space, we have one optimization variable for each layer.

References

[1] Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. Probabilistic neural architecture search. *arXiv preprint arXiv:1902.05116*, 2019. 8

[2] Ting-Wu Chin, Pierce I-Jen Chuang, Vikas Chandra, and Diana Marculescu. One weight bitwidth to rule them all. In Adrien Bartoli and Andrea Fusiello, editors, *Computer Vision – ECCV 2020 Workshops*, pages 85–103, Cham, 2020. Springer International Publishing. 3

[3] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1518–1528, 2020. 1, 8

[4] Ting-Wu Chin, Ari S Morcos, and Diana Marculescu. Pareco: Pareto-aware channel optimization for slimmable neural networks. *arXiv preprint arXiv:2007.11752*, 2020. 1

[5] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020. 11

[6] 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. 5

[7] Ruizhou Ding, Ting-Wu Chin, Zeye Liu, and Diana Marculescu. Regularizing activation distribution for training binarized deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11408–11417, 2019. 3

[8] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pages 760–771, 2019. 1

[9] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1586–1595, 2018. 1, 3, 5

[10] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1539–1547, 2020. 1, 3, 5

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2, 4, 5

[12] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018. 3, 8

[13] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018. 1, 8

[14] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 3

[15] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 3, 8

[16] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. Dhp: Differentiable meta pruning via hypernetworks. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 608–624, Cham, 2020. Springer International Publishing. 8

[17] Yawei Li, Wen Li, Martin Danelljan, Kai Zhang, Shuhang Gu, Luc Van Gool, and Radu Timofte. The heterogeneity hypothesis: Finding layer-wise dissimilated network architecture. *arXiv preprint arXiv:2006.16242*, 2020. 1

[18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 8

[19] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017. 7

[20] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3296–3305, 2019. 1, 5

[21] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018. 8

[22] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. *arXiv preprint arXiv:2007.10396*, 2020. 8

[23] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019. 3, 8

[24] Rameswar Panda, Michele Merler, Mayoore Jaiswal, Hui Wu, Kandan Ramakrishnan, Ulrich Finkler, Chun-Fu Chen, Minsik Cho, David Kung, Rogerio Feris, et al. Nastransfer: Analyzing architecture transferability in large scale neural architecture search. *arXiv preprint arXiv:2006.13314*, 2020. 8

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019. 11

[26] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollar. Designing network design

spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020. 1, 11

[27] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 4, 5

[28] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 3, 6

[29] Catherine Wong, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo. Transfer learning with neural automl. In *Advances in Neural Information Processing Systems*, pages 8356–8365, 2018. 8

[30] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019. 1, 3

[31] Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11396–11404, 2020. 8

[32] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 8