# WIDTH TRANSFER: ON THE (IN)VARIANCE OF WIDTH OPTIMIZATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Optimizing the channel counts for different layers of a convolutional neural network (CNN) to achieve better accuracy without increasing the number of floating-point operations (FLOPs) required during the forward pass at test time is known as CNN width optimization. Prior work on width optimization has cast it as a hyper-parameter optimization problem, which introduces large computational overhead (*e.g.*, an additional $2\times$ FLOPs of standard training). Minimizing this overhead could therefore significantly speed up training. With that in mind, this paper sets out to empirically understand width optimization by sensitivity analysis. Specifically, we consider the following research question: "*Do similar training configurations for a width optimization algorithm also share similar optimized widths?*" If this in fact is the case, it suggests that one can find a proxy training configuration requiring fewer FLOPs to reduce the width optimization overhead. Scientifically, it also suggests that similar training configurations share common architectural structure, which may be harnessed to build better methods. To this end, we control the training configurations, *i.e.*, network architectures and training data, for three existing width optimization algorithms and find that the optimized widths are largely transferable across settings. Per our analysis, we can achieve up to $320\times$ reduction in width optimization overhead without compromising the top-1 accuracy on ImageNet. 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 (Guo et al., 2020; Gordon et al., 2018; Yu & Huang, 2019). However, designing the width 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, designing the width for each layer based on computational methods has received growing interests. Examples include using reinforcement learning (He et al., 2018b), evolutionary algorithms (Liu et al., 2019; Chin et al., 2020b), and differentiable parameterization (Guo et al., 2020; Dong & Yang, 2019; Ning et al., 2020) to optimize for layer widths. However, these methods often add a large computational overhead for the width optimization procedure. Concretely, even for efficient methods that use differentiable parameterization (Guo et al., 2020), 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 for training a ResNet50 on the ImageNet dataset (Radosavovic et al., 2020). That is, it takes 300 GPU hours for both width optimization using differentiable methods (Guo et al., 2020) 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 (Chin et al., 2020b). 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 its limited understanding by the community. Without assuming or understanding the structure of the problem,

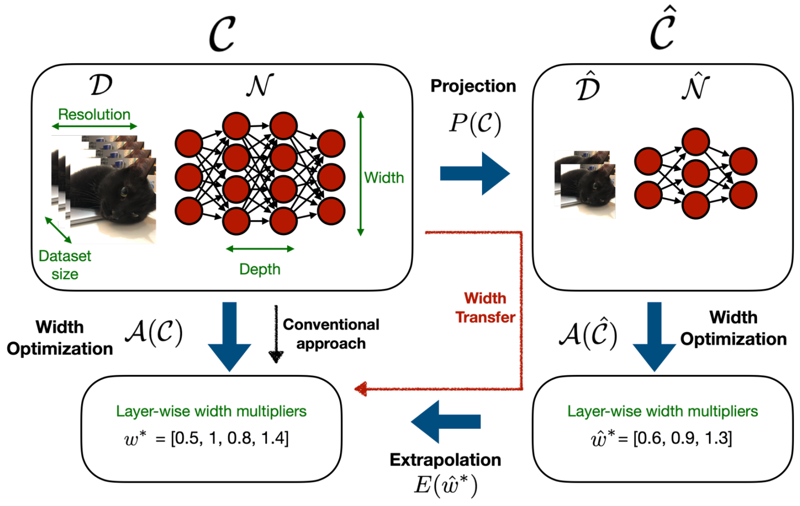


Figure 1: A schematic view of width transfer. $\mathcal{N}$ denotes the network to be optimized, $\mathcal{D}$ denotes the training dataset, $\hat{\mathcal{N}}$ and $\hat{\mathcal{D}}$ denote the projected network and dataset. $P(\cdot)$ is a projection function parameterized by tunable knobs $\boldsymbol{s}$. $\mathcal{A}$ is the width-optimization algorithm and $E(\cdot)$ is a simple extrapolation function to match the dimensions of $\hat{w}^*$ to $w^*$.

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. Such sensitivity analysis techniques have been used in various contexts in the deep learning literature for empirically unveiling the black-box of deep learning (Morcos et al., 2018; Tan & Le, 2019; Li et al., 2018). Similarly, we manipulate the network architectures and dataset properties to aid in our understanding of the (in)variances of width optimization.

A width optimization algorithm, $\mathcal{A}$, takes in a training configuration, $\mathcal{C} = (\mathcal{D}, \mathcal{N})$, and outputs a set of optimized widths, $w^*$, which maxmizes the validation accuracy without additional test-time FLOPs. $\mathcal{A}$ can be seen as neural architecture search algorithms that search for layer-wise channel counts. $\mathcal{C}$ consists of initial network, $\mathcal{N}$, and training dataset, $\mathcal{D}$. In this paper, we systematically analyze how similar $\mathcal{C}$ affects $w^*$.

If similar inputs to the width optimization algorithms result 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 shown schematically in Figure 1. 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. In addition to the potential efficiency benefits from understanding the structure of the width optimization problem, such an exploration can also have scientific benefits. Specifically, via sensitivity analysis, we can provide quantitative results to the following question: Does the level of overparameterization, number of training samples, and the resolution of input images affect width optimization? And if so, by how much?

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

- We find that there exist shared structures in the width optimization problem across a wide variety of network and dataset configurations. Specifically, the outputs of width optimization algorithms are largely robust to perturbation of the network's depth and width via the multiplier method and to perturbation of the dataset's sample size and resolution.

- We demonstrate a practical implication of the previous finding by showing that one can achieve 320× reduction in width optimization overhead for a scaled-up MobileNetV2 and

ResNet18 on ImageNet without hurting the accuracy improvements brought by width optimization.

- We find that, for ResNet18 on ImageNet, width optimization has limited benefits for very deep models.

## 2 Related work

### 2.1 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 (Howard et al., 2017) was introduced in MobileNet to arrive at models with different FLOPs and accuracy profiles and has been widely adopted in many papers (He et al., 2018a; Tan & Le, 2019; Chin et al., 2020a).

Besides simply scaling the width to arrive at CNNs with different FLOPs, width (or channel) optimization has received growing interest recently as a means to improve the efficiency of deployed deep CNNs. To optimize the width of a CNN, one approach is *Prune-then-Grow*, which uses channel pruning methods to arrive at a down-sized CNN with non-trivial layerwise channel counts followed by re-growing the CNN to its original FLOPs using the width multiplier method (Gordon et al., 2018). Another approach is *Grow-then-Prune*, which uses the width multiplier method to enlarge the CNN followed by channel pruning methods to trim down channels to match its pre-grown FLOPs (Yu & Huang, 2019; Liu et al., 2019; Guo et al., 2020). The aim of both of these methods is to improve performance while maintaining a given FLOPs count. The schematic view of the two approaches is visualized in the left panel of Figure 2.

While there are many papers on channel pruning (Li et al., 2016; Molchanov et al., 2019), they mostly focus their analysis on down-sizing the pre-trained models whereas we focus on improving the classification accuracy of a network by optimizing its width without affecting test-time FLOPs. 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 unoptimized 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.

### 2.2 Empirical sensitivity analysis for understanding deep learning

Controlling the components of deep learning to further shed light on understanding and optimization is an important direction complementing theoretical understanding for deep learning. While our work is the first that focuses on empirically understanding the sensitivity of width optimization in deep CNNs, we discuss efforts in empirically understanding deep learning more generally.

Empirical analysis to gain insight into hyperparameter selection for training deep neural networks has received great interest. Goyal et al. (2017) have shown that the accuracy can be retained across a wide range of batch sizes when the number of epochs is held fixed while the batch size is scaled linearly. Shallue et al. (2019) have conducted a comprehensive analysis that sheds light on the relationship among batch size, training iterations, and learning rate. Tan & Le (2019) have empirically controlled the network's architecture to identify a more cost-efficient way of scaling deep models with high classification accuracy. Radosavovic et al. (2020) have empirically manipulated the architectural choices for CNNs and identified a parameterizable relationship among depth, channel counts, and group size for ResNets.

There are also papers using analysis as a tool for better understanding deep learning phenomena. Morcos et al. (2018) have used Canonical Correlation Analysis to empirically understand if networks of different architectures and optimization behavior are of different clusters. Li et al. (2018) have controlled the network's architecture and observed that networks of different architectures have different empirical loss landscapes. Frankle et al. (2019) altered the winning ticket generation procedures in the lottery ticket hypothesis (Frankle & Carbin, 2018) and observed that an empirical stability measure predicts well the success of winning tickets. Morcos et al. (2019) have empirically controlled the training configuration for the winning ticket generation in the lottery ticket hypothesis and discovered the transferability of winning tickets.

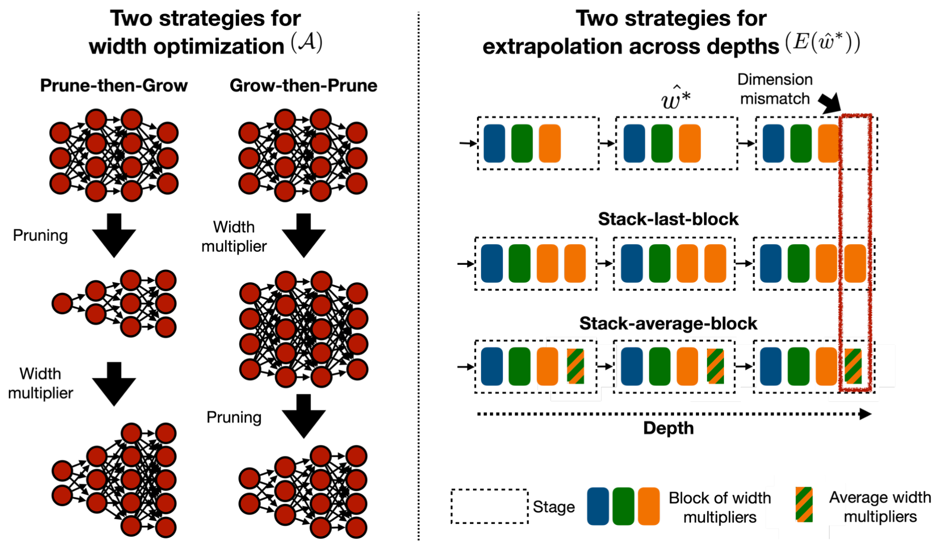

Two strategies for width optimization $(\mathcal{A})$
Prune-then-Grow
Grow-then-Prune
Pruning
Width multiplier
Width multiplier
Pruning
Two strategies for extrapolation across depths $(E(\hat{w}^*))$
$\hat{w}^*$
Dimension mismatch
Stack-last-block
Stack-average-block
Depth
Stage
Block of width multipliers
Average width multipliers

Figure 2: A schematic view of different width optimization strategies in existing algorithms (left) and a cartoon illustration of the two extrapolation strategies for scaling the optimized width multipliers from lower dimensions to higher dimensions (right). Best view in color.

## 3 Approach

### 3.1 Notation

We use $[n]$ to represent a set $\{1, 2, ..., n\}$. A width optimization algorithm, $\mathcal{A}$, is a function that takes a training configuration, $\mathcal{C} = (\mathcal{D}, \mathcal{N})$, which consists of a training dataset, $\mathcal{D}$, and a network to be optimized, $\mathcal{N}$. The output of $\mathcal{A}$ is a vector of width multipliers, $\boldsymbol{w}^*$, whose dimension is $L$ (the number of layers). Let $F_i$ denote the channel counts for layer $i$, it is expected that a network with optimized channel counts $\{w_i F_i \ \forall\, i \in [L]\}$ has the same FLOPs as a network with the original channel counts $\{F_i \ \forall i \in [L]\}$ but has better test accuracy when trained with $\mathcal{D}$. Following common terminologies, a CNN is divided into stages where the convolutional blocks in each stage share the same output resolutions. Within each stage, several convolutional blocks are repeated where a convolutional block consists of several convolutional layers such as the bottleneck block in ResNet (He et al., 2016) and the inverted residual block in MobileNetV2 (Sandler et al., 2018). We use the concept of stage and block for describing extrapolation mechanisms in Section 3.3.

### 3.2 Width optimization methods

Theoretically, we only care about algorithms $\mathcal{A}$ that "*solve*" the width optimization problem. However, the width optimization problem is inherently combinatorially hard. As a result, we use state-of-the-art width optimization algorithms as probes to understand them further. Specifically, we consider MorphNet (Gordon et al., 2018), AutoSlim (Yu & Huang, 2019), and DMCP (Guo et al., 2020). Here, we explicitly consider papers that have analyzed width optimization, *i.e.*, improving the accuracy while maintaining the test time FLOP requirements. We also limited our investigation to methods with publicly available code to ensure correctness of implementation.

### 3.3 Projection and extrapolation

We consider various projection to project a large-scale training setting $\mathcal{C} = (\mathcal{D}, \mathcal{N})$ into a small-scale proxy training setting $\hat{\mathcal{C}} = (\hat{\mathcal{D}}, \hat{\mathcal{N}})$ and this allows us to understand the structure about $\mathcal{A}$ for all three considered algorithms. More specifically, we consider projecting the network $\mathcal{N}$ down to narrower networks via the width multiplier method and shallower networks via the depth multiplier method (Tan & Le, 2019). For projection on the dataset $\mathcal{D}$, we consider subsampling the number of training images and changing the input resolutions.

The extrapolation function involves two stages: matching dimensions and matching test-time FLOPs between $\boldsymbol{w}^*$ and $\hat{\boldsymbol{w}}^*$, *i.e.*, $E(\hat{\boldsymbol{w}}) \overset{\text{def}}{=} \alpha M_{\text{dim}}(\hat{\boldsymbol{w}})$ where $M_{\text{dim}}(\hat{\boldsymbol{w}})$ is a function that matches dimensions and $\alpha$ is responsible for matching test-time FLOPs. If the depth multiplier method is involved during projection, $\hat{\boldsymbol{w}}^*$ and $\boldsymbol{w}^*$ will have different dimensions. As a result, we propose the following two $M_{\text{dim}}$ to extrapolate the found width multipliers to higher dimensions.

- **Stack-last-block**: Stack the width multipliers of the last block of each stage until the desired depth is met.
- **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 $M_{\text{dim}}$ will have the same results when applied to networks with un-optimized widths. After the dimension is matched, we apply a multiplier $\alpha$ to it such that the resulting network has a test-time FLOPs similar to that induced by $\boldsymbol{w}^*$. The schematic views of extrapolation is shown in the right panel of Figure 2.

## 3.4 Experimental Setup

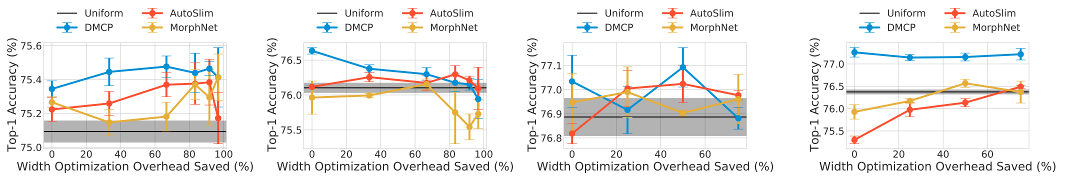
We used the ImageNet dataset (Deng et al., 2009) throughout the experiments. Unless stated otherwise, we use 224 input resolution. For CNNs, we considered both ResNet18 (He et al., 2016) and MobileNetV2 (Sandler et al., 2018). Models were each trained on a single machine with 8 V100 GPUs for all the 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 $\boldsymbol{w}^*$ or $E(\hat{\boldsymbol{w}}^*)$ we trained the corresponding network from scratch using the same hyperparameters to analyze their performance. The training hyperparameters are detailed in Appendix A. We repeated each experiment three times with different random seeds and reported the mean and standard deviation.

## 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 sub-sampling and resolution sub-sampling for dataset projection. 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 induced by $\boldsymbol{w}^*$ and $E(\hat{\boldsymbol{w}}^*)$ across training configurations 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 $\boldsymbol{w}^*$ from a smaller FLOP configuation. 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 Projection: width

Here, we focus on answering the following question: "*Do networks with different initial width (varied by the width multiplier method) share common structures in their optimized widths?*" The answer to this question is unclear from existing literature as the current practice is to re-run the optimization across different networks (Guo et al., 2020; Gordon et al., 2018; Liu et al., 2019). If the optimized widths are similar across different initial widths, it 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, it suggests that not only the direction of the vector of channel counts is important, but also its magnitude. That is, for different magnitudes, we need different orientations. Practically, it suggests that existing practice, though costly, is empirically proved to be necessary.



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

Figure 3: We consider transferring the optimized widths obtained from smaller models and extrapolate that to the largest model (1.732× wide or 4× deep). We plot the ImageNet top-1 accuracy for uniform baseline, transferred width $E(\hat{\boldsymbol{w}}^*)$, and direct optimization $\boldsymbol{w}^*$ (the leftmost points). On the x-axis, we plot the width optimization overhead saved by using width transfer.

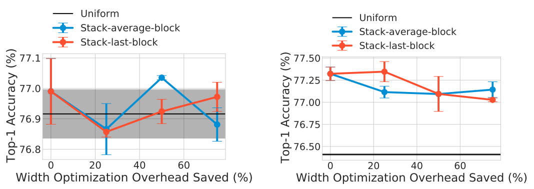
To empirically study the aforementioned question, we considered the source width multipliers of $\{0.312, 0.5, 0.707, 1, 1.414, 1.732\}$ for $\hat{\mathcal{N}}$ and the target width multiplier of 1.732 for $\mathcal{N}$. The set is chosen based on square roots of width optimization overhead. We analyzed the similarity between $E(\hat{\boldsymbol{w}}^*)$ and $\boldsymbol{w}^*$ in the accuracy space. In Figure 3a and 3b, we plot the ImageNet top-1 accuracy for the baseline (1.732× wide network) and networks induced by $E(\hat{\boldsymbol{w}}^*)$ and $\boldsymbol{w}^*$. 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. Since $E(\hat{\boldsymbol{w}}^*)$ in this case is just applying $\alpha\hat{\boldsymbol{w}}^*$ to $\mathcal{N}$ where $\alpha$ makes the resulting network have the same FLOPs as the network induced by $\boldsymbol{w}^*$, 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 Appendix (Figure A1).

### 4.2 Projection: depth

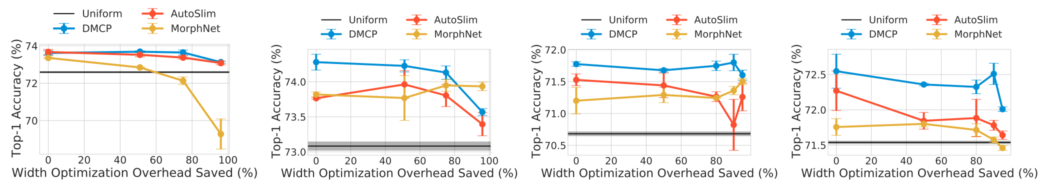
Next, we asked whether networks with different initial depths share common structure in their optimized widths. Because making a network deeper will add new layers with no corresponding optimized width, we will need a mechanism to map the vector optimized widths, $\boldsymbol{w}^*$, to a vector with far more elements. We empirically investigate two methods for aligning across depth, which are detailed in Section 3.3. We considered $\{1, 2, 3, 4\}$ as the depth multipliers for $\hat{\mathcal{N}}$ and use 4 for $\mathcal{N}$. Similar to the analysis done in Section 4.1, we analyzed the similarity in the accuracy space.



(a) ResNet18 (b) MobileNetV2

Figure 4: We compare the two extrapolation strategies using DMCP for both ResNet18 and MobileNetV2. We can observe that both stack-average-block and stack-last-block perform similarly.

Here, we first compared the two extrapolation methods proposed in Section 3.3 using DMCP for ResNet18 and MobileNetV2. As shown in Figure 4, both strategies perform similarly. We focus on the stack-average-block strategy for the following experiments.



(a) Res18, resolution (b) MBv2, resolution (c) Res18, dataset size (d) MBv2, dataset size

Figure 5: We consider transferring the optimized widths obtained from smaller training data configurations to the largest training data configuration ($320 \times 320$ input resolution or full data size). We plot the ImageNet top-1 accuracy for uniform baseline, transferred width $E(\hat{\boldsymbol{w}}^*)$, and direct optimization $\boldsymbol{w}^*$ (the leftmost points). On the x-axis, we plot the width optimization overhead saved by using width transfer.

As shown in Figure 3c and 3d, we find that the optimized widths stay competitive via simple extrapolation 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 observe that width optimization itself, even when optimized directly for that configuration, has limited benefits (for all three algorithms) for much deeper ResNet18 models, which suggests that width optimization may be less useful when the network to be optimized is heavily over-parameterized.

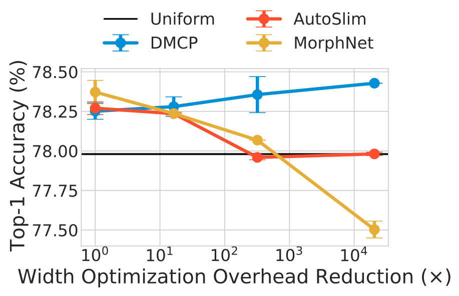
### 4.3 PROJECTION: RESOLUTION

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 (Tan & Le, 2019). As a result, it is not clear if width optimization algorithms are sensitive to input resolution. We therefore asked whether networks trained on different input resolutions also share structure in their optimized widths. If the optimized widths are indeed similar, 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.

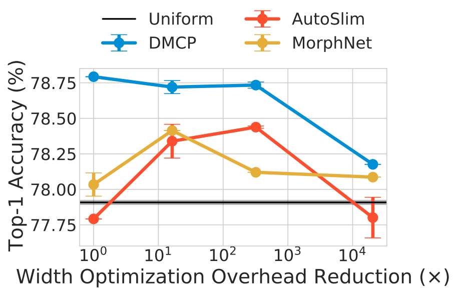
To empirically study the aforementioned question, we considered the input resolution for $\hat{\mathcal{D}}$ to be $\{64, 160, 224, 320\}$ and choose a $\mathcal{D}$ of 320. As shown in Figure 5a and 5b, we find that except for MorphNet targeting ResNet18, all other algorithm and network combinations 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 A2 in Appendix.

### 4.4 PROJECTION: DATASET SIZE

The dataset size is often critical for understanding the generalization performance of a learning algorithm. Here, we would like to understand how width optimization algorithms are affected by the size of training data. We considered sub-sampling the ImageNet dataset to result in $\{5\%, 10\%, 20\%, 50\%, 100\%\}$ 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 5c and 5d, widths optimized for smaller dataset sizes transfer well to large dataset sizes. That is, 95% width optimization overhead can be saved and still outperform the uniform baseline for both networks. On the other hand, 90% width optimization overhead can be saved and still match the performance of direct optimization for



(a) ResNet18



(b) MobileNetV2

Figure 6: Width transfer with compound projection. By using width transfer, we can achieve 320× width optimization overhead reduction for width optimization while remain competitive compared to direct optimization for the top-performing algorithm, DMCP.

DMCP. This suggests that the amount of training data barely affects width optimization, especially for DMCP, which is surprising.

We further conducted experiments using CIFAR-100 and have two findings. First, the optimized widths are similar across these two datasets for DMCP. Second, width optimization results in overfitting for CIFAR-100 and calls for cross validation techniques in width optimization. The supporting materials for these two findings are in Appendix C.

### 4.5 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, depth, resolution, dataset size) denote a training configuration. We considered $\hat{\mathcal{C}}$ to be $\{(0.312,1,64,5\%), (0.707,1,160,10\%), (1,1,224,50\%), (1.414,2,320,100\%)\}$ and $\mathcal{C}$ to be $(1.414,2,320,100\%)$. As shown in Figure 6, the optimized width is transferable across compound projection. Specifically, we can achieve up to 320× 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.

## 5 Discussion

In this paper, we take a first step in understanding the transferability of the optimized widths across different width optimization algorithms and projection dimensions. This investigation sheds light on the width optimization problem, which is often regarded as a black-box problem and tackled with general optimization methods (Liu et al., 2019; Guo et al., 2020; Gordon et al., 2018). More specifically, we show that there are common structures in the optimized widths obtained across a wide range of settings such that one can successfully transfer the optimized width to various settings with competitive performance. Per our analysis, we can achieve up to 320× reduction in width optimization overhead without compromising the top-1 accuracy on ImageNet. Our findings not only suggest an efficient alternative to conduct width optimization, but also imply that width optimization can be done for lower dimensional inputs, which can be beneficial since it allows a more effective traversal of the design space.

While encouraging, our study also presents some limitations. Specifically, we empirically consider two types of CNNs: ResNet18 and MobileNetV2. While these networks are popular currently in our community, it is not clear if such encouraging characteristics hold for other CNNs. Additionally, extending this work beyond convolutional neural networks is an interesting direction going forward.

## REFERENCES

Ting-Wu Chin, Pierce I-Jen Chuang, Vikas Chandra, and Diana Marculescu. One weight bitwidth to rule them all. *arXiv preprint arXiv:2008.09916*, 2020a.

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*, pp. 1518–1528, 2020b.

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*, pp. 702–703, 2020.

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*, pp. 248–255. Ieee, 2009.

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

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. *arXiv preprint arXiv:1912.05671*, 2019.

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*, pp. 1586–1595, 2018.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

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*, pp. 1539–1547, 2020.

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*, pp. 770–778, 2016.

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*, 2018a.

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)*, pp. 784–800, 2018b.

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.

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

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pp. 6389–6399, 2018.

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*, pp. 3296–3305, 2019.

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*, pp. 11264–11272, 2019.

Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems*, pp. 5727–5736, 2018.

Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*, pp. 4932–4942, 2019.

Xuefei Ning, Tianchen Zhao, Wenshuo Li, Peng Lei, Yu Wang, and Huazhong Yang. Dsa: More efficient budgeted pruning via differentiable sparsity allocation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

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*, pp. 8026–8037, 2019.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10428–10436, 2020.

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*, pp. 4510–4520, 2018.

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019. URL `http://jmlr.org/papers/v20/18-789.html`.

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

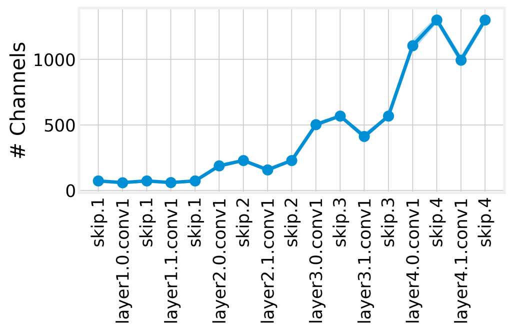
Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.

## A Training hyperparameters

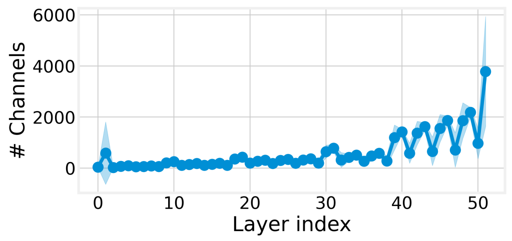
We use PyTorch (Paszke et al., 2019) as our deep learning framework. We largely follow Radosavovic et al. (2020) for training hyperparameters. Specifically, learning rate grows linearly from 0 to $0.2s$ within the first 5 epochs from 0 where $s$ depend on batch size $B$, *i.e.*, $s = \frac{B}{256}$. We use batch size of 1024 and distributed training over 8 GPUs and we have not used synchronized batch normalization layers. We set the training epochs to be 100. For optimizers, we use stochastic gradient descent (SGD) with 0.9 Nesterov momentum. As for data augmentation, we have adopted 0.1 label smoothing, random resize crop, random horizontal flops, and RandAugment (Cubuk et al., 2020) with parameter $N = 2$ and $M = 9$ following common practice in popular repository[1]. Note that these training hyperparameters are fixed for all experiments. In other words, we always train for 100 epochs regardless of the dataset size when we conduct the dataset projection in Section 4.4

For hyperparameters specific to width optimization algorithms, we largely follow the hyperparameters used in respective methods. Specifically, we use 40 epochs to search for optimized width for all three algorithms. We enlarge the network by $1.5\times$ for DMCP and AutoSlim. Since MorphNet has FLOPs-aware regualrization, we normalize the FLOPs for each network and use $\lambda = 1$ for all experiments.

## B Width visualization



(a) ResNet18



(b) MobileNetV2

Figure A1: The average optimized width for ResNet18 and MobileNetV2. They are averaged across the optimized widths in Section 4.1. We plot the mean in solid line with shaded area representing standard deviation.

## C CIFAR-100 experiments

In this section, we are interested in understanding how dependent is width optimization on the training data. To do so, we use CIFAR-100 as the dataset and sweep the network configuration to cover both depth and width multipilers. Specifically, we consider width $\{0.312, 0.5, 0.707, 1, 1.414, 1.732\}$ and depth $\{1, 2, 3, 4\}$ for ResNet18 using DMCP. To allow using the same architecutres as the experiments for ImageNet, we alter the input resolution for CIFAR-100 from $32 \times 32$ to $64 \times 64$. Finally we compare the pairwise cosine similarity between the architectures searched across the two different datasets, *i.e.*, ImageNet and CIFAR-100. As shown in Table 1, we find that the channel counts searched on these two datasets are more similar than comparing the optimized width with uniform baselines.

At first glance, one might conclude that the optimal channel configurations are similar across these two datasets. However, we find that the optimized widths perform worse than uniform (un-optimized widths) on CIFAR-100 due to over-fitting. This suggests that a cross validation method should be used for width optimization algorithms.

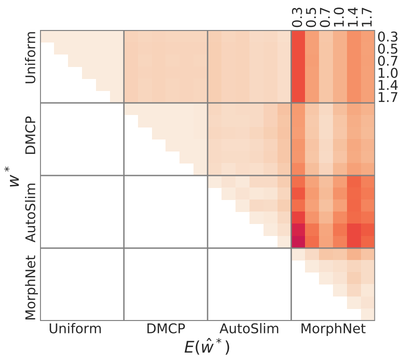
[1]`https://github.com/rwightman/pytorch-image-models` (We use their implementation for RandAugment and use 'rand-m9-mstd0.5' as the value for the 'aa' flag.

| | | 0.3w,1d | 0.5w,1d | 0.7w,1d | 1w,1d | 1.4w,1d | 1.7w,1d | 1w,2d | 1w,3d | 1w,4d |
|---|---|---|---|---|---|---|---|---|---|---|
| CIFAR-100 | Uniform | 61.1 | 64.6 | 66.5 | 67.5 | 68.8 | 69.4 | 68.7 | 69.0 | 69.9 |
| | DMCP | 60.8 | 64.5 | 66.4 | 67.4 | 68.4 | 69.2 | 68.5 | 69.0 | 69.8 |
| ImageNet | Uniform | 53.1 | 61.7 | 66.8 | 70.7 | 73.4 | 75.0 | 74.4 | 76.2 | 77.0 |
| | DMCP | 56.2 | 64.1 | 68.2 | 71.8 | 74.3 | 75.3 | 74.7 | 76.0 | 77.0 |
| cosine(CIFAR,ImageNet) | | 0.96 | 0.95 | 0.95 | 0.94 | 0.94 | 0.95 | 0.91 | 0.91 | 0.92 |
| cosine(CIFAR,uniform) | | 0.93 | 0.92 | 0.92 | 0.92 | 0.91 | 0.92 | 0.88 | 0.88 | 0.88 |

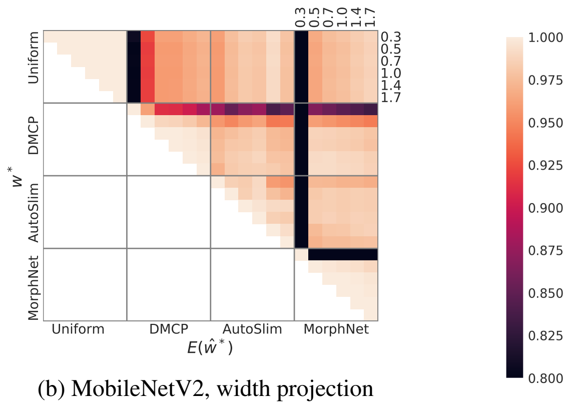
Table 1: Comparing the optimized widths found using ImageNet and CIFAR-100 datasets. The first two sections are the top-1 accuracy on respective datasets while the last section is the cosine similarity among the widths.

## D Similarity among width multipliers

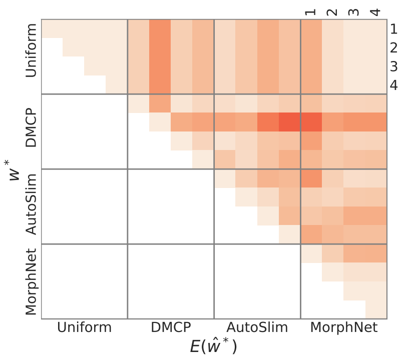
In Section 4, we have analyzed the similarity between $\boldsymbol{w}^*$ and $\hat{\boldsymbol{w}}^*$ in the accuracy space. Here, we show that $\boldsymbol{w}^*$ and $\hat{\boldsymbol{w}}^*$ are in fact similar in the vector space using cosine similarity.



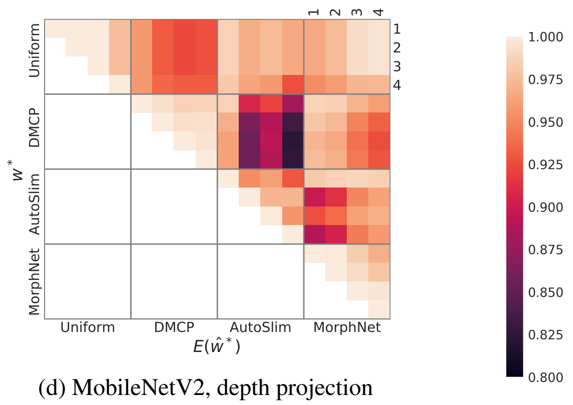
(a) ResNet18, width projection



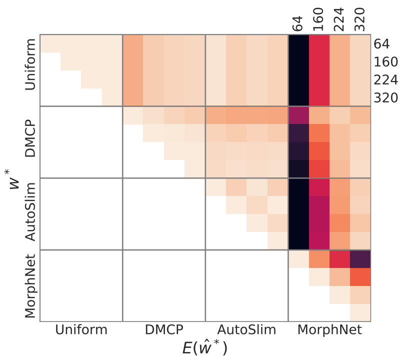
(b) MobileNetV2, width projection



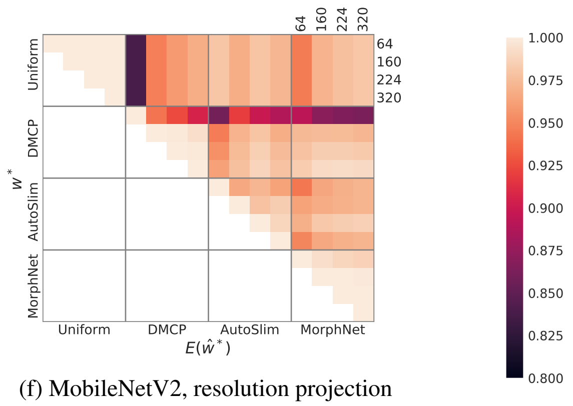
(c) ResNet18, depth projection



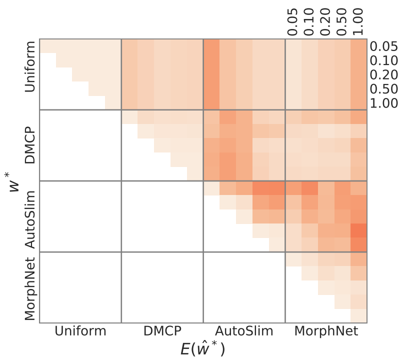
(d) MobileNetV2, depth projection



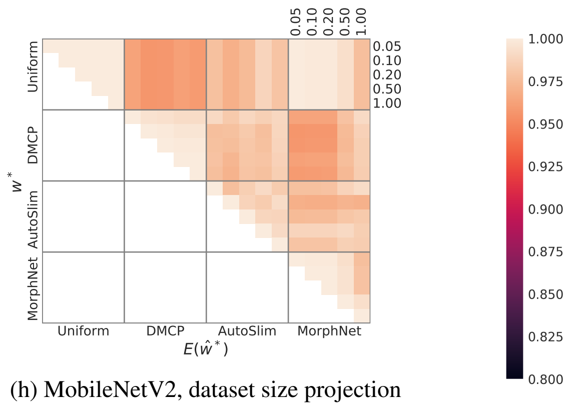
(e) ResNet18, resolution projection



(f) MobileNetV2, resolution projection



(g) ResNet18, dataset size projection



(h) MobileNetV2, dataset size projection

Figure A2: Pairwise cosine similarity between $\boldsymbol{w}^*$ and $E(\hat{\boldsymbol{w}}^*)$ for different width optimization algorithms and projection strategies. Within each methods (diagonal blocks), $\boldsymbol{w}^*$ and $E(\hat{\boldsymbol{w}}^*)$ are generally similar.