MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks
Supplementary material

A Inception V2 trained on ImageNet

In this section we provide the technical details regarding the the experiments in Section 5.2 of the paper.

When training with a FLOP regularizer, we used a learning rate of 10^{-3}, and we kept it constant in time. The values of λ that were used to obtain the points displayed in Figure 4 are 0.7, 1.0, 1.3, 2.0 and 3.0, all times 10^{-9}.

Tables 1 and 2 below lists the size of each convolution in Inception V2, for the seed network and for the two MorphNetiterations. The names of the layers are the ones generated by this[1] code. Each column represents a learned DNN structure, obtained from the previous one by applying a FLOP regularizer with $\lambda = 1.3 \cdot 10^{-9}$ and then the width multiplier that was needed to restore the number of FLOPs to the initial value of $3.88 \cdot 10^9$. The width multipliers at iteration 1 and 2 respectively were 1.692 and 1.571.

B MobileNet Training Details

B.1 Training protocol

Our models operate on 128×128 images. The training procedure is a slight variant of running the main MorphNetalgorithm for one iteration. This variability gives better results overall and is crucial for MorphNetto overtake the 50% width-multipler model (see below). The procedure is as follows:

1. The full network (width-multipler of 1.0 on 128×128 image input) was first trained for 2 million steps (which is the typical number of steps for a network's performance to plateau as observed from training models with similar model sizes). Note that training smaller networks (e.g. with a width-multiplier of 0.25) takes significantly more steps, e.g. around 10 millions steps, to converge.

[1] https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/slim/python/slim/nets/inception_v2.py

2. The checkpoint was used to initialize MorphNettraining, which goes on for an additional 10 million steps or until the FLOPs of the active channels converge, whichever is longer. We tried a range of λ values $\in \{3, 4, \ldots, 10, 11\} \times 10^{-9}$ to ensure that the converged FLOPs remain close to the FLOPs of the width-multiplier baselines.

3. We took the converged checkpoint and extracted a pruned network (both structure and weights) that consists of only the active channels.

4. Finally, we fine-tuned the pruned network using a small learning rate (0.0013). This is merely to restore moving average statistics for batch-normalization, and normally takes a negligible number, e.g. $20k$, of steps. While training for longer keeps improving the accuracy, simply training for $20k$ steps suffices to outperform models with width multipliers.

All training steps use the same optimizer, which is discussed below.

B.2 Trainer

We use the same trainer from MobileNet v2 (2), described below. We trained with the RMSProp optimizer (3) implemented in Tensorflow (1) with a batch-size of 96. The initial learning rate was chosen from $\{0.013, 0.045\}$, unless otherwise specified. The learning rate decays by a factor of 0.98 every 2.5 epochs. Training uses 16 workers asynchronously.

B.3 Observations

The total training time for each attempted λ value is around $2 + 10 = 12$ million steps, which is less than twice the number of steps (around 10 million) for training a regular network. Although multiple λ values are required, each one of them contributes to the "optimal" FLOPs-vs-accuracy tradeoff, as shown in figure 1. The "optimality" is defined in a narrow sense that no model is dominated in both FLOP and accuracy by another. By contrast, the 50% width-multipler model is dominated by the MorphNetmodels. Finally, we found that both the learning rate and the λ parameter affects the converged FLOPs, but just the λ parameter by itself suffices to traverse the range of desirable FLOPs.

C ResNet101 on JFT

The FLOP regularizer λ-s used in Figure 5 on JFT were 0.7, 1.0, 1.3 and 2 times 10^{-9}. The size regularizer λ-s were 0.7, 1 and 3 times 10^{-7}. The width multiplier values were 1.0, 0.875, 0.75, 0.625, 0.5, and 0.375. Figure 2 illustrates the structures learned when applying these regularizers on ResNet101.

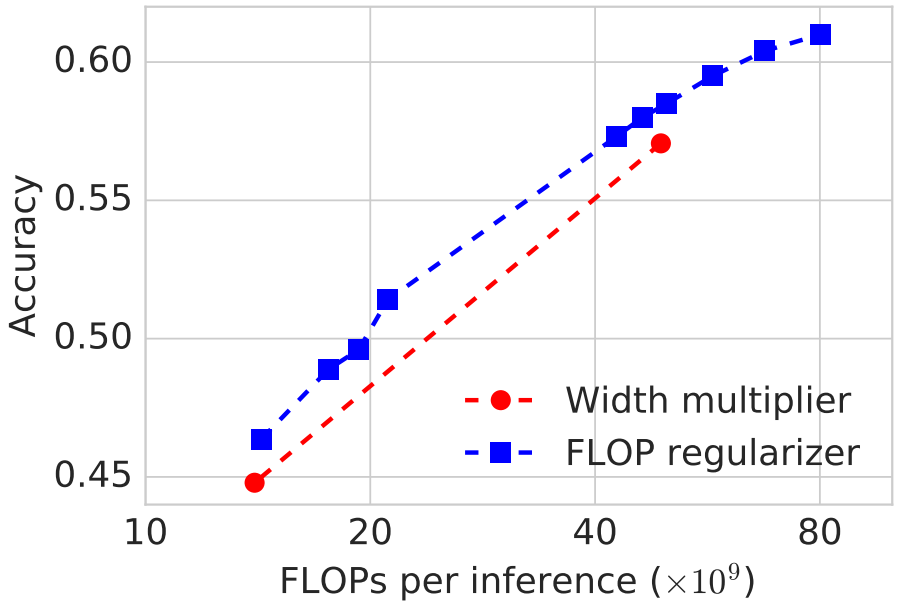

Figure 1: ImageNet evaluation accuracy for various MobileNets on 128×128 images using both a naïve width multiplier (red circles) and a sparsifying FLOP regularizer (blue squares).

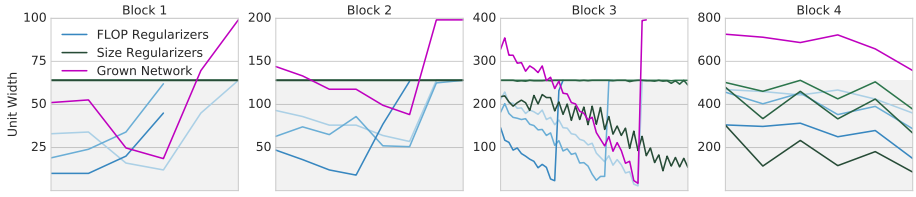

Figure 2: Each of the four figures show the width of units in ResNet101 blocks (1-4). The green (blue) shaded lines represent different strength of the FLOP (size) regularizer. The purple line represents the unit width of a model expanded from a FLOP-regularized ResNet101 model so that the number of FLOPs-matches these of the seed model. One can observe that increasing strengths of the FLOP regularizer (darker blue) remove more and more neurons from all blocks, and remove entire residual units (layers) from all blocks except for Block 4. Increasing the strength of the size regularizer (darker green) mainly removes neurons from Block 4.

D Stability of MorphNet

In this section, we study the stability of MorphNetwith Inception V2 model on the ImageNet dataset. We trained the Inception V2 model regularized by

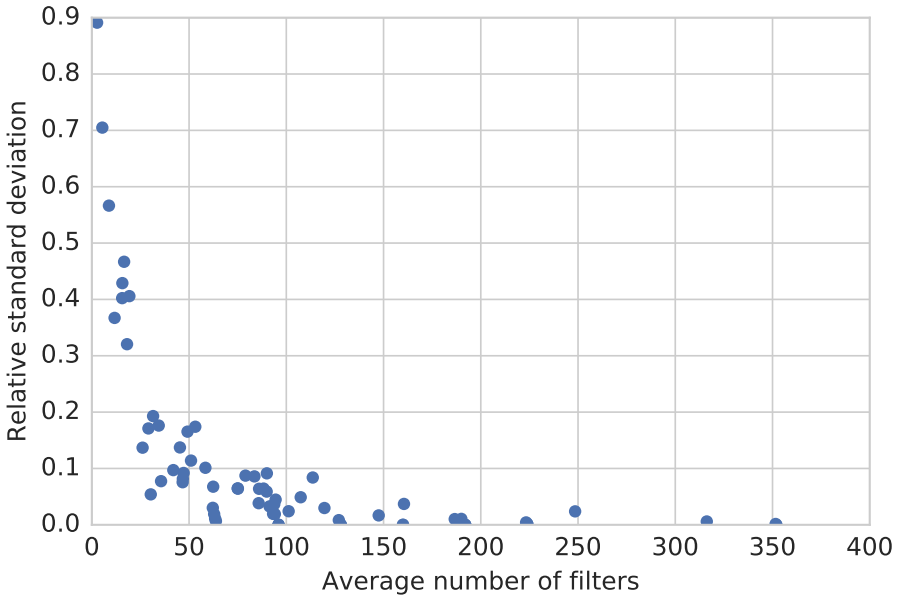

Figure 3: A scatter plot of relative standard deviations v.s. average number of filters of each layer in ImageNet Inception V2 model. The standard deviation was calculated over the results of 10 independent runs of Inception V2 with a FLOP regularizer of $\lambda = 1.3 \cdot 10^{-9}$, using the same hyperparameter configuration.

FLOP regularizer with a constant learning rate of 10^{-3}. We also set the value of λ to be 1.3×10^{-9}. The training procedure was repeated independently for 10 times. We extracted the final architecture, e.g. the number of filters in each layer, generated by MorphNetfrom each run, and computed the relative standard deviations[2] (RSTD) for the number of filters in each layer of the Inception V2 model across the 10 independent runs. Figure 3 shows the scatter plot of RSTD for the ImageNet Inception V2 model. Such results show that the number of filters in most of the layers does not change too much across different runs of MorphNetwith the same parameter configuration. Few of the layers have slightly large RSTD. However the number of filters in these layers is small, which means the absolute changes of the number of filters in these layers are still quite small across independent runs. Figure 4 shows the scatter plot of FLOPs v.s. test accuracy of Inception V2 model retrained over ImageNet dataset with the network architectures generated by the 10 independent runs of MorphNetwith FLOPs regularizer. As we can see from this figure, the FLOPs and test accuracies from different runs all converged to the same region with a relative standard deviation of **1.12%** and **0.208%** respectively, which are

[2]Standard deviation divided by the mean.

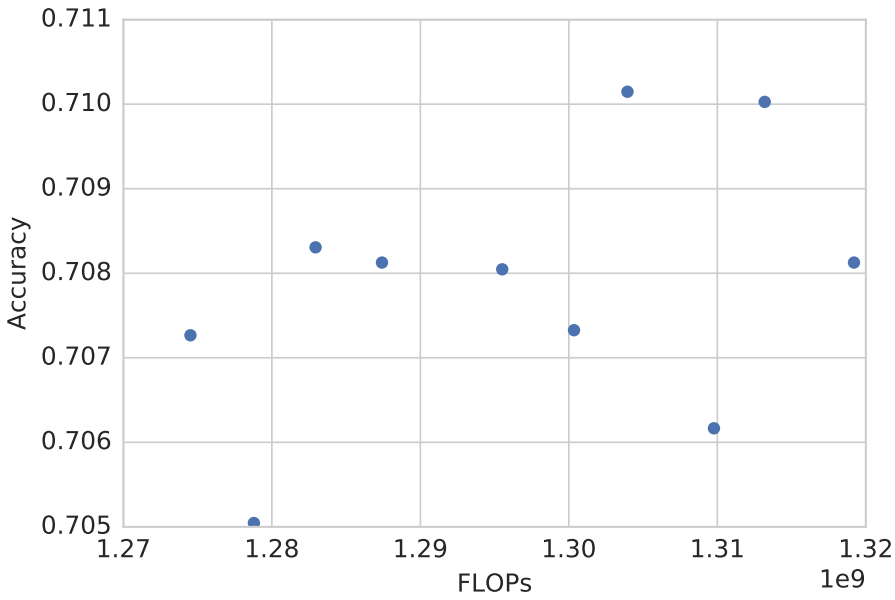

Figure 4: FLOPs v.s. test accuracy for Inception V2 model on the ImageNet dataset. Each point represents an independent run of Inception V2 with a FLOP regularizer of $\lambda = 1.3 \cdot 10^{-9}$, using the same hyperparameter configuration. The differences in the FLOP counts of the resulting architectures and in their test accuracies is shown in the figure. The relative standard deviation for FLOPs and test accuracy across 10 runs are **1.12%** and **0.208%** respectively.

relatively small. All of these results demonstrate that the MorphNetis capable of generating pretty stable DNN architectures under constrained computation resources.

E Extensions of the method

We have restricted the discussion and evaluation in this paper to optimizing only the output widths $O_{1:M}$ of all layers. However, our iterative process of shrinking via a sparsifying regularizer and expanding via a uniform multiplicative factor easily lends itself to optimizing over other aspects of network design.

For example, to determine filter dimensions and network depth, previous work (4) has proposed to leverage Group LASSO and residual connections to induce structured sparsity corresponding to smaller filter dimensions and reduced network depth. This gives us a suitable shrinking mechanism. For expansion, one may reuse the idea of the width multiplier to uniformly expand all filter dimensions and network depth. To avoid a substantially larger network, it may be beneficial to incorporate some rules regarding which filters will be uniformly expanded (*e.g.,*by observing which filters were least affected by the sparsifying regularizer; or more simply by random selection).

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*, 1, 2015.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. In *Computer Vision and Pattern Recognition, 2018. CVPR 2018. IEEE Conference on*. IEEE, 2018.

[3] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[4] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

Layer name	iteration 0	iteration 1	iteration 2
Conv2d_1a_7x7	64	78	86
Conv2d_2b_1x1	64	51	25
Conv2d_2c_3x3	192	217	309
Mixed_3b/Branch_0/Conv2d_0a_1x1	64	108	170
Mixed_3b/Branch_1/Conv2d_0a_1x1	64	81	0
Mixed_3b/Branch_1/Conv2d_0b_3x3	64	73	0
Mixed_3b/Branch_2/Conv2d_0a_1x1	64	73	112
Mixed_3b/Branch_2/Conv2d_0b_3x3	96	42	63
Mixed_3b/Branch_2/Conv2d_0c_3x3	96	61	96
Mixed_3b/Branch_3/Conv2d_0b_1x1	32	52	69
Mixed_3c/Branch_0/Conv2d_0a_1x1	64	108	170
Mixed_3c/Branch_1/Conv2d_0a_1x1	64	15	24
Mixed_3c/Branch_1/Conv2d_0b_3x3	96	8	13
Mixed_3c/Branch_2/Conv2d_0a_1x1	64	19	0
Mixed_3c/Branch_2/Conv2d_0b_3x3	96	0	0
Mixed_3c/Branch_2/Conv2d_0c_3x3	96	17	0
Mixed_3c/Branch_3/Conv2d_0b_1x1	64	108	168
Mixed_4a/Branch_0/Conv2d_0a_1x1	128	130	75
Mixed_4a/Branch_0/Conv2d_1a_3x3	160	154	82
Mixed_4a/Branch_1/Conv2d_0a_1x1	64	54	66
Mixed_4a/Branch_1/Conv2d_0b_3x3	96	86	69
Mixed_4a/Branch_1/Conv2d_1a_3x3	96	154	154
Mixed_4b/Branch_0/Conv2d_0a_1x1	224	377	573
Mixed_4b/Branch_1/Conv2d_0a_1x1	64	108	121
Mixed_4b/Branch_1/Conv2d_0b_3x3	96	159	107
Mixed_4b/Branch_2/Conv2d_0a_1x1	96	161	124
Mixed_4b/Branch_2/Conv2d_0b_3x3	128	178	53
Mixed_4b/Branch_2/Conv2d_0c_3x3	128	181	83
Mixed_4b/Branch_3/Conv2d_0b_1x1	128	201	258
Mixed_4c/Branch_0/Conv2d_0a_1x1	192	325	496
Mixed_4c/Branch_1/Conv2d_0a_1x1	96	134	13
Mixed_4c/Branch_1/Conv2d_0b_3x3	128	147	11
Mixed_4c/Branch_2/Conv2d_0a_1x1	96	144	162
Mixed_4c/Branch_2/Conv2d_0b_3x3	128	154	118

Table 1:

Layer name	iteration 0	iteration 1	iteration 2
Mixed_4c/Branch_2/Conv2d_0c_3x3	128	135	146
Mixed_4c/Branch_3/Conv2d_0b_1x1	128	217	303
Mixed_4d/Branch_0/Conv2d_0a_1x1	160	271	424
Mixed_4d/Branch_1/Conv2d_0a_1x1	128	105	94
Mixed_4d/Branch_1/Conv2d_0b_3x3	160	118	90
Mixed_4d/Branch_2/Conv2d_0a_1x1	128	51	80
Mixed_4d/Branch_2/Conv2d_0b_3x3	160	39	61
Mixed_4d/Branch_2/Conv2d_0c_3x3	160	58	91
Mixed_4d/Branch_3/Conv2d_0b_1x1	96	162	255
Mixed_4e/Branch_0/Conv2d_0a_1x1	96	162	255
Mixed_4e/Branch_1/Conv2d_0a_1x1	128	110	64
Mixed_4e/Branch_1/Conv2d_0b_3x3	192	130	82
Mixed_4e/Branch_2/Conv2d_0a_1x1	160	32	50
Mixed_4e/Branch_2/Conv2d_0b_3x3	192	22	35
Mixed_4e/Branch_2/Conv2d_0c_3x3	192	36	57
Mixed_4e/Branch_3/Conv2d_0b_1x1	96	162	255
Mixed_5a/Branch_0/Conv2d_0a_1x1	128	217	324
Mixed_5a/Branch_0/Conv2d_1a_3x3	192	325	482
Mixed_5a/Branch_1/Conv2d_0a_1x1	192	151	237
Mixed_5a/Branch_1/Conv2d_0b_3x3	256	73	113
Mixed_5a/Branch_1/Conv2d_1a_3x3	256	404	635
Mixed_5b/Branch_0/Conv2d_0a_1x1	352	596	936
Mixed_5b/Branch_1/Conv2d_0a_1x1	192	321	11
Mixed_5b/Branch_1/Conv2d_0b_3x3	320	535	17
Mixed_5b/Branch_2/Conv2d_0a_1x1	160	271	258
Mixed_5b/Branch_2/Conv2d_0b_3x3	224	379	178
Mixed_5b/Branch_2/Conv2d_0c_3x3	224	379	200
Mixed_5b/Branch_3/Conv2d_0b_1x1	128	217	341
Mixed_5c/Branch_0/Conv2d_0a_1x1	352	596	930
Mixed_5c/Branch_1/Conv2d_0a_1x1	192	257	102
Mixed_5c/Branch_1/Conv2d_0b_3x3	320	168	110
Mixed_5c/Branch_2/Conv2d_0a_1x1	192	313	300
Mixed_5c/Branch_2/Conv2d_0b_3x3	224	272	146
Mixed_5c/Branch_2/Conv2d_0c_3x3	224	178	226
Mixed_5c/Branch_3/Conv2d_0b_1x1	128	217	341

Table 2: