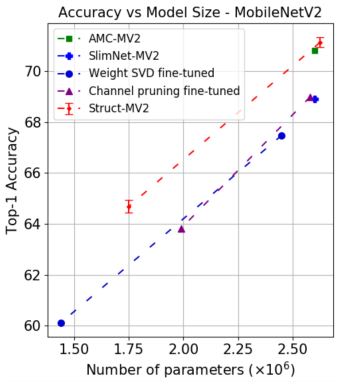
We thank all reviewers for their valuable feedback. We tried our best to address their questions within the page limit.

(R1,R3)(1): We agree that R1's suggested approach (of training structured convolution as a architectural feature) is more direct and does not require computing the regularization. We actually experimented with the suggested 'direct' approach and observed that the regularization based approach always outperformed the direct approach (by 1.5% for Struct-18-A and by 0.9% for Struct-MV2-A). We think this is because the direct approach optimizes the weights in a restricted subspace of $c \times n \times n$ kernels right from the start, whereas the regularization based approach gradually moves the weights from the larger ($C \times N \times N$) subspace to the restricted subspace with gradual imposition of the structure constraints using the regularization loss. We will include this analysis and the results in more detail in the revision. We hope that this also addresses **R3's question "why not directly optimize the weights in the restricted subspace**?"
(R1)(2): We agree with R1 that finetuning using Eq (3) from pretrained weights (whenever they are available) may be a useful alternative, but our proposed method also works well with random initialization.

(R1,R4) Computational cost of regularization term: Although our focus is more on inference efficiency, we measure memory and time-per-iteration for training *with* and *w/o* SR loss, on an NVIDIA V100 GPU. Mathematically, the SR term, $\sum_{l=1}^{L} \frac{\left\|(I - A_l A_l^{+}) W_l\right\|_F}{\|W_l\|_F}$, is independent of the input size. Hence, when using a large batchsize, the SR term's memory and runtime overhead is relatively small as shown in the table. As R1 suggested, we could eliminate the computational cost of the regularization term if we use the 'direct' approach, but that leads to a noticeable loss in accuracy as shown in above answer.

Training costs (batchsize=256)	Struct-18-A		Struct-MV2-A	
	Mem	sec/iter	Mem	sec/iter
With SR loss	9.9GB	0.46s	18.8GB	0.38s
Without SR loss	9.2GB	0.44s	17.9GB	0.37s

(R1) Comments on Fig.6: In Fig. 6, we compare our performance with the widely used *structured compression* approaches [48]-2016, [12]-2017, [35]-2018, since our method belongs to this category. In addition, Tables 1,2,3 present comparisons with some of the most recent methods in efficient architectures (e.g. GhostNet - CVPR 2020, SlimmableNets - ICLR 2019). Following R1's suggestion, we will add error-bars to Fig.6. We show one of the acc v/s parameters plots here for reference. Note that, we provide the **mean** $\pm$ **stddev** values for our method, whereas the compared methods have only provided their '**best**' numbers. As visible, the proposed method outperforms by a considerable margin (0.5% than the highly regarded AMC [11]). Looking beyond performance numbers, we think that our proposal of the structured convolutions offers a new way in efficient model design.



(R1,R4) Inference latency: Following the convention in EfficientNet paper [37], we report the average inference latency over 50 runs measured with batch size 1 on a single core of Intel Xeon CPU W-2123. As discussed on lines 231-235 and also pointed out by R4, our method is designed for recent accelerators [44] that allow efficient sum-pooling operations, thus the theoretical speedups (Tables 1-4) are realizable on such platforms.

ResNet18	0.039s	MobilenetV2	0.088s	EfficientNet-B1	0.114s
Struct-18-A	**0.030s**	Struct-MV2-A	**0.078s**	Struct-EffNet	**0.101s**

(R3,R4) Details on training Steps 1 and 2: To answer R4's question, we first train the model *until convergence* in Step1 which imposes the desired structure and then decompose in Step 2. To answer R3's question, the approximation error comes in Step 2 after decomposition ($\alpha_l = A_l^{+} W_l$). This is because the desired structure is not enforced, but *induced* via the loss function. We do provide performance numbers after 1^{st} step (and 2^{nd} step) for ResNet18 in Sec. E (tables 3,4) of supplementary. We will report these numbers for all other architectures too in our camera-ready version.

(R4) Clarification on Implementation Details: We will add pseudocode and more intuitive diagrams about the training details in the appendix of our camera-ready version. As outlined in Fig. 4 of paper, the implementation steps may be summarized as follows - in Step 1, we train the architecture with original $C \times N \times N$ kernels in place and the regularization loss imposes desired structure on these kernels. Then, in step 2, we decompose the $C \times N \times N$ convolution layer and replace it with a sum-pooling layer followed by a smaller conv layer with kernels of size $c \times n \times n$. The reported parameters and operations (in Tables 1-4) were calculated using https://github.com/sovrasov/flops-counter.pytorch.

(R3) Laplacian filter: A Laplacian filter can be constructed with the basis (β_i's) as shown, hence can be decomposed into just a horizontal and vertical sum-pooling component. This is an interesting analysis question, that we plan to look into as future work.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} = -1 \times \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} + -1 \times \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} + 6 \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

(R2): Simply swapping 3×3 kernels with 2×2's in MobileNetV2 led to a severe drop in accuracy ($\approx 4.5\%$). This, we believe, is due to the loss of receptive field that was being captured by the sum-pooling part of structured convolutions.

(R2,R3) Typos: Thanks R2, R3 for finding the typos. We will correct them in the camera-ready version.