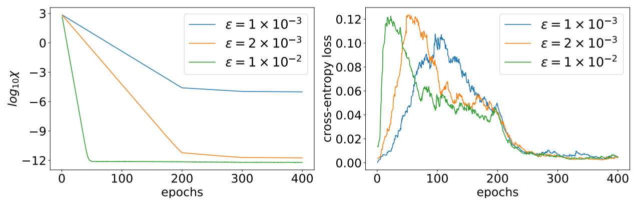
Centripetal SGD for Pruning Very Deep Convolutional Networks with Complicated Structure
Appendix

Xiaohan Ding [1] Guiguang Ding [2] Yuchen Guo [3] Jungong Han [4]
[1,2,3] Tsinghua University [4] Lancaster University
dxh17@mails.tsinghua.edu.cn dinggg@tsinghua.edu.cn {yuchen.w.guo, jungonghan77}@gmail.com



(a) Value of χ. (b) Training loss.

Figure 1: Curves of the sum of squared kernel deviation χ (note the logarithmic scale) and the training loss with different centripetal strength ϵ. The learning rate is decayed at epoch 200 and 300, respectively.

C-SGD is Insensitive to the Hyper-parameter

We perform a set of controlled experiments on ResNet-56 to study the effects of the centripetal strength ϵ by setting ϵ to 1×10^{-3}, 2×10^{-3} and 1×10^{-2}, respectively. Fig. 1 shows that C-SGD is robust to ϵ, as the three models converge in a similar way. Intuitively, when we use C-SGD to produce the redundancy patterns on *every* layer simultaneously, the network undergoes a period of progressive change, which leads to an increasing loss. When this kind of change becomes stable, *i.e.*, when the filters in each cluster have become close enough, the loss starts to decrease. Obviously, the filters in each cluster grow centripetally at a faster rate with a larger ϵ, thus the change is finished earlier.

C-SGD Literally Slims the Network

As a contribution of this paper, we have partly solved an open problem of constrained filter pruning, which several preceding works choose to sidestep. From a holistic perspective we are literally slimming the complicated networks, rather than clipping some layers. More implementation details are presented in Fig. 2 and Fig. 3, where each rectangle represents a filter, and different filters labeled by the same letter are pushed close during C-SGD training.

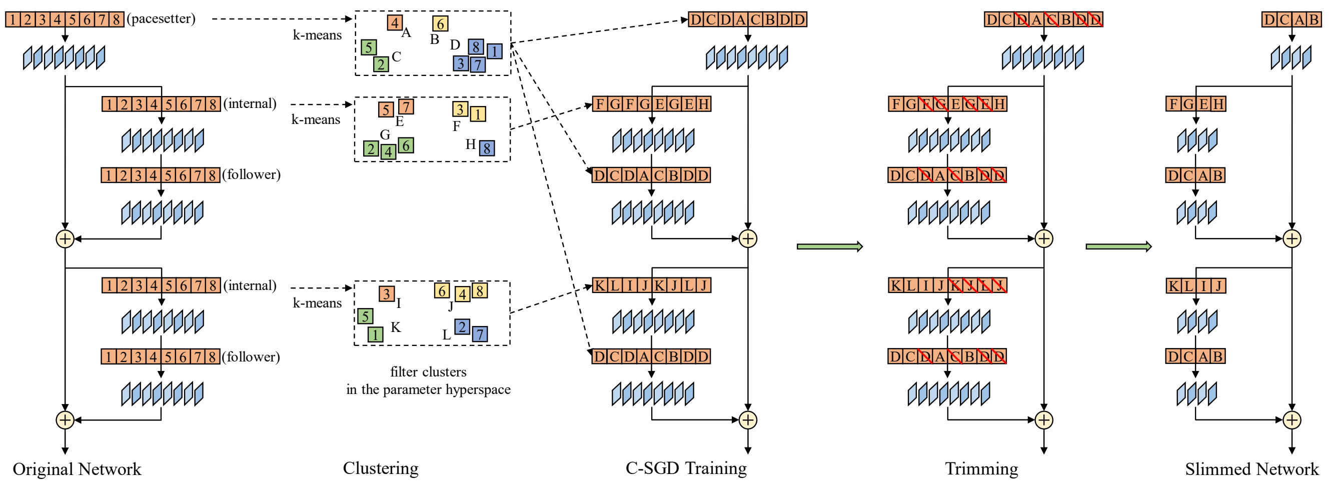


Figure 2: Sketch for slimming ResNets. We take the first stage of a toy ResNet where every layer has 8 filters for example. Since every convolutional layer is directly followed by exactly one batch normalization layer, we view them as a whole. We generate clusters for the pacesetter and internal layers in each stage by k-means for example. Before we start C-SGD training, the clustering result of a pacesetter is assigned to its followers in order to produce the same redundancy pattern.

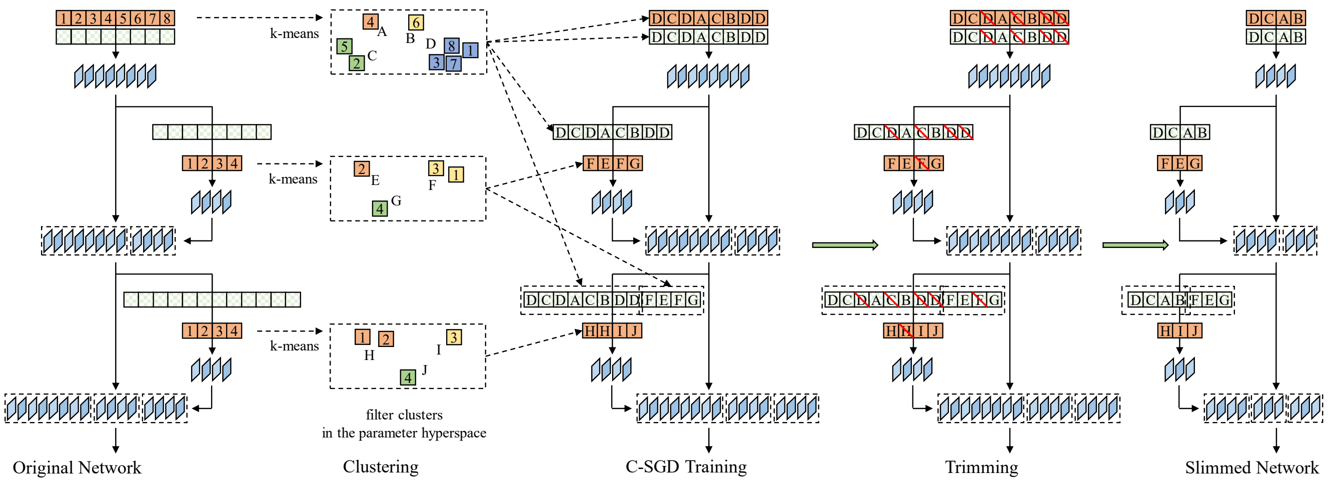


Figure 3: Sketch for slimming DenseNets. We take a toy DenseNet with growth rate 4 for example. Considering the special dense connection and pre-activation structure of DenseNets, we treat the batch normalization layers separately, which are denoted by the rectangles with chessboard-like background. As the output feature map of every convolutional layer serves as the input of one or more batch normalization layers, we generate clusters for every convolutional layer and apply the clustering results $\mathcal{C}$ to every following batch normalization layer *at the corresponding position*, such that the gradients of γ and β are transformed as the preceding convolutional layers.