Table 1: Complexity analysis on CIFAR-10 for different expansion rates $r$. (The original network is the SmallNet with kernel size 7 used in Section 5 (#Params:150.35K, #MACs: 6.12M, Epoch Time: 4.05s).

| $r$ | 2 | | | 4 | | | 8 | | |
|---|---|---|---|---|---|---|---|---|---|
| | #Params | #MACs | Epoch Time | #Params | #MACs | Epoch Time | #Params | #MACs | Epoch Time |
| FC(Arora18) | 339.40K | 6.30M | 4.09s | 675.91K | 6.64M | 3.94s | 1.74M | 7.70M | 4.02s |
| ExpandNet-CL | 562.95K | 25.16M | 4.13s | 2.17M | 98.39M | 4.61s | 8.58M | 389.35M | 9.39s |
| ExpandNet-CK | 237.72K | 14.38M | 4.10s | 653.25K | 42.64M | 4.12s | 2.07M | 141.10M | 5.50s |

**To R1:** Thank you for the positive feedback. We will incorporate your suggestions in the final version.

**Gradient analysis.** Following [49], we account for all the layers in each network to analyze the gradient confusion.

**Initialization.** The gradients of the initialized ExpandNet before and after contraction are not the same, which indeed contributes to their different training behavior. This was studied in [5] for purely linear FC networks, and remains true for our new convolutional expansion strategies in non-linear CNNs. We will clarify this.

**Computational overheads and wall-clock training times.** The numbers provided in Table 1 above for a SmallNet with kernel size 7 show that our expansion strategies can better leverage GPU computation, thus leading to only moderate wall-clock time increases as $r$ grows, particularly for our CK strategy. We report additional values for MobileNets in Table 2 below and provide a complexity analysis in Section B of the supplementary material.

**Other suggestions:** As a rule of thumb, "compact networks" in practice have < 10% of the parameters of conventional CNNs. There is indeed a bijection between the original and matrix form of a convolution. The padding/stride schemes maintain equivalence between the expanded and compact networks.

**To R2: Training time.** As shown in Table 1 above, the training time only moderately increases with $r$, because our models can better exploit the GPU to make full use of its capacity, particularly for very small networks. For larger ones, the GPU usage saturates, and thus the training time of ExpandNets increases. To nonetheless confirm that our better results are not just due to longer training, we increased the training time of the MobileNets to match that of our ExpandNets-CL. This led to the accuracies highlighted in Table 2 with a †, which remain lower than those of our ExpandNets.

Table 2: Top-1 accuracy (%) of MobileNets vs ExpandNets with $r = 4$ on CIFAR-10 and CIFAR-100.

| Model | #Params | #MACs | Epoch Time | CIFAR-10 | CIFAR-100 |
|---|---|---|---|---|---|
| MobileNet | 3.22M | 47.18M | 13.08s | 89.61 (88.87†) | 67.93 (68.18†) |
| ExpandNet-CL | 3.94M | 101.36M | 22.78s | **91.79** | **69.75** |
| MobileNetV2 | 2.30M | 94.60M | 24.88s | 91.64 (90.85†) | 71.66 (71.41†) |
| ExpandNet-CL | 3.12M | 207.87M | 49.22s | **92.58** | **72.33** |

† Accuracy with the same training time as ExpandNet-CL.
‡ #Params, #MACs and Epoch Time were evaluated on CIFAR-10 on 2 32G TITAN V100 GPUs.

**Network pruning.** Our work tackles a different problem from network pruning: 1) We aim to train a given compact architecture, whereas pruning yields an arbitrary one, which precludes a direct comparison; 2) We show that linear over-parameterization is beneficial for network training; 3) Contraction in our work yields no information loss.

**MobileNetV2 on ImageNet.** Many people have observed that training MobileNetV2 on ImageNet following the experimental setting in [48] does not yield the reported results. In a recent repository[1], heavy hyper-parameter tuning eventually led to 72.20% accuracy. In this setting, our ExpandNet-CL reaches *73.16%*.

**CIFAR baselines.** We focused on compact networks, which may not yield SOTA results, such as the SmallNet ($\sim$80% on CIFAR-10) taken from [37]. Nonetheless, in Table 2, we report the results of *new*, stronger baselines – MobileNets on the CIFAR datasets. These results, which we will include in the final version, show that stronger baselines still benefit from our expansion strategies.

**To R3: Rationale of our method.** Over-parameterization is widely acknowledged as beneficial for network training, and mathematical explanations have been studied in [1, 2, 5, 16, 50, 62] for linear networks under specific assumptions. Here, we contribute novel linear expansion strategies for convolutional networks and confirm the benefits of over-parameterization. Note that, as mentioned in Answer 2 to R1, our expansion schemes modify the network gradients, thus explaining their different training behavior. Importantly, all other reviewers acknowledge the thoroughness and interest of our empirical validation, and argue that our work is significant for the community.

**Backpropagation settings of non-expanded networks.** As mentioned to R2, we used the hyper-parameters tuned for the non-expanded networks to train both the non-expanded networks and our ExpandNets. Despite this, our ExpandNets still reach better solutions. This remains true for our new MobileNetV2 experiments on ImageNet and CIFAR above.

**Felix Wu et al.**. Thank you for the reference, which we will discuss. Wu et al. show that non-linearities can be removed from the *MLP* layers of graph convolution networks. Because they collapse multiple FC layers into one, their work is in fact closer to [5]. In contrast to our work, they do *not* remove the non-linearities of the conv layers and collapse them; do *not* introduce expansion strategies for conv layers; and do *not* study the benefits of linear over-parameterization.

**To R4:** Thank you for the positive feedback. As correctly pointed out by R4, our work differs from matrix factorization in that, via our expansion strategies, we aim to incorporate over-parameterization in a given compact network so as to improve its accuracy. We will highlight this in the final version.

[1]Reference: d-li14, PyTorch Implemention of MobileNetV2, available at https://github.com/d-li14/mobilenetv2.pytorch.