This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

CondenseNet: An Efficient DenseNet using Learned Group Convolutions

Gao Huang*
Cornell University
gh349@cornell.edu

Shichen Liu*
Tsinghua University
liushichen95@gmail.com

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

Abstract

Deep neural networks are increasingly used on mobile devices, where computational resources are limited. In this paper we develop CondenseNet, a novel network architecture with unprecedented efficiency. It combines dense connectivity with a novel module called learned group convolution. The dense connectivity facilitates feature re-use in the network, whereas learned group convolutions remove connections between layers for which this feature re-use is superfluous. At test time, our model can be implemented using standard group convolutions, allowing for efficient computation in practice. Our experiments show that CondenseNets are far more efficient than state-of-the-art compact convolutional networks such as ShuffleNets.

1. Introduction

The high accuracy of convolutional networks (CNNs) in visual recognition tasks, such as image classification [12, 19, 38], has fueled the desire to deploy these networks on platforms with limited computational resources, *e.g.*, in robotics, self-driving cars, and on mobile devices. Unfortunately, the most accurate deep CNNs, such as the winners of the ImageNet [6] and COCO [31] challenges, were designed for scenarios in which computational resources are abundant. As a result, these models cannot be used to perform real-time inference on low-compute devices.

This problem has fueled development of computationally efficient CNNs that, *e.g.*, prune redundant connections [9, 11, 27, 29, 32], use low-precision or quantized weights [4, 21, 36], or use more efficient network architectures [5, 12, 16, 19, 22, 47]. These efforts have lead to substantial improvements: to achieve comparable accuracy as VGG [38] on ImageNet, ResNets [12] reduce the amount of computation by a factor $5\times$, DenseNets [19] by a factor of $10\times$, and MobileNets [16] and ShuffleNets [47] by a factor of $25\times$. A typical set-up for deep learning on mobile devices is one where CNNs are trained on multi-GPU machines but deployed on devices with limited compute. Therefore, a good network architecture allows for fast parallelization during training, but is compact at test-time.

Recent work [4, 20] shows that there is a lot of redundancy in CNNs. The layer-by-layer connectivity pattern forces networks to replicate features from earlier layers throughout the network. The DenseNet architecture [19] alleviates the need for feature replication by directly connecting each layer with all layers before it, which induces feature re-use. Although more efficient, we hypothesize that dense connectivity introduces redundancies when early features are not needed in later layers. We propose a novel method to prune such redundant connections between layers and then introduce a more efficient architecture. In contrast to prior pruning methods, our approach learns a sparsified network automatically during the training process, and produces a regular connectivity pattern that can be implemented efficiently using group convolutions. Specifically, we split the filters of a layer into multiple groups, and gradually remove the connections to less important features *per group* during training. Importantly, the groups of incoming features are not predefined, but *learned*. The resulting model, named CondenseNet, can be trained efficiently on GPUs, and has high inference speed on mobile devices.

Our image-classification experiments show that CondenseNets consistently outperform alternative network architectures. Compared to DenseNets, CondenseNets use only $1/10$ of the computation at comparable accuracy levels. On the ImageNet dataset [6], a CondenseNet with 275 million FLOPs[1] achieved a 29% top-1 error, which is comparable to the error of a MobileNet that requires twice as much compute.

*Both authors contributed equally.

[1]Throughout the paper, *FLOPs* refers to the number of multiplication-addition operations.

2. Related Work and Background

We first review related work on model compression and efficient network architectures, which inspire our work. Next, we review the DenseNets and group convolutions that form the basis for CondenseNet.

2.1. Related Work

Weights pruning and quantization. CondenseNets are closely related to approaches that improve the inference efficiency of (convolutional) networks via weight pruning [11, 14, 27, 29, 32] and/or weight quantization [21, 36]. These approaches are effective because deep networks often have a substantial number of redundant weights that can be pruned or quantized without sacrificing (and sometimes even improving) accuracy. For convolutional networks, different pruning techniques may lead to different levels of granularity [34]. Fine-grained pruning, *e.g.*, independent weight pruning [10, 27], generally achieves a high degree of sparsity. However, it requires storing a large number of indices, and relies on special hardware/software accelerators. In contrast, coarse-grained pruning methods such as filter-level pruning [1, 14, 29, 32] achieve a lower degree of sparsity, but the resulting networks are much more regular, which facilitates efficient implementations.

CondenseNets also rely on a pruning technique, but differ from prior approaches in two main ways: First, the weight pruning is initiated in the early stages of training, which is substantially more effective and efficient than using L_1 regularization throughout. Second, CondenseNets have a higher degree of sparsity than filter-level pruning, yet generate highly efficient group convolution—reaching a sweet spot between sparsity and regularity.

Efficient network architectures. A range of recent studies has explored efficient convolutional networks that can be trained end-to-end [16, 19, 22, 46, 47, 48, 49]. Three prominent examples of networks that are sufficiently efficient to be deployed on mobile devices are MobileNet [16], ShuffleNet [47], and Neural Architecture Search (NAS) networks [49]. All these networks use depth-wise separable convolutions, which greatly reduce computational requirements without significantly reducing accuracy. A practical downside of these networks is depth-wise separable convolutions are not (yet) efficiently implemented in most deep-learning platforms. By contrast, CondenseNet uses the well-supported group convolution operation [25], leading to better computational efficiency in practice.

Architecture-agnostic efficient inference has also been explored by several prior studies. For example, knowledge distillation [3, 15] trains small "student" networks to reproduce the output of large "teacher" networks to reduce test-time costs. Dynamic inference methods [2, 7, 8, 17] adapt the inference to each specific test example, skipping units or even entire layers to reduce computation. We do not explore such approaches here, but believe they can be used in conjunction with CondenseNets.

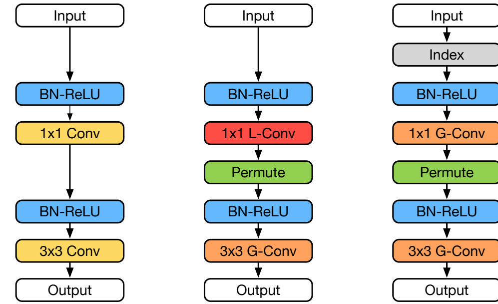

Figure 1. The transformations within a *layer* in DenseNets *(left)*, and CondenseNets at training time *(middle)* and at test time *(right)*. The *Index* and *Permute* operations are explained in Section 3.1 and 4.1, respectively. (L-Conv: learned group convolution; G-Conv: group convolution)

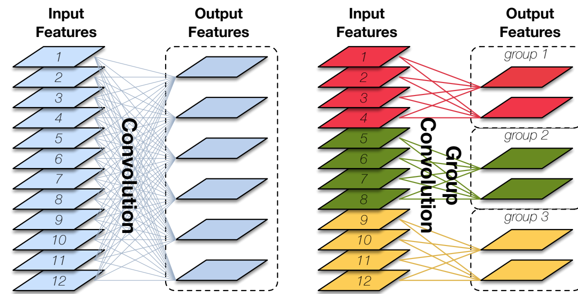

Figure 2. Standard convolution *(left)* and group convolution *(right)*. The latter enforces a sparsity pattern by partitioning the inputs (and outputs) into disjoint groups.

2.2. DenseNet

Densely connected networks (DenseNets; [19]) consist of multiple *dense blocks*, each of which consists of multiple *layers*. Each layer produces k features, where k is referred to as the *growth rate* of the network. The distinguishing property of DenseNets is that the input of each layer is a concatenation of all feature maps generated by *all* preceding layers within the same dense block. Each layer performs a sequence of consecutive transformations, as shown in the left part of Figure 1. The first transformation (*BN-ReLU*, blue) is a composition of batch normalization [23] and rectified linear units [35]. The first convolutional layer in the sequence reduces the number of channels to save computational cost by using the 1×1 filters. The output is followed by another BN-ReLU transformation and is then reduced to the final k output features through a 3×3 convolution.

2.3. Group Convolution

Group convolution is a special case of a sparsely connected convolution, as illustrated in Figure 2. It was first used in the AlexNet architecture [25], and has more recently been popularized by their successful application in ResNeXt [43]. Standard convolutional layers (left illustration in Figure 2) generate O output features by applying

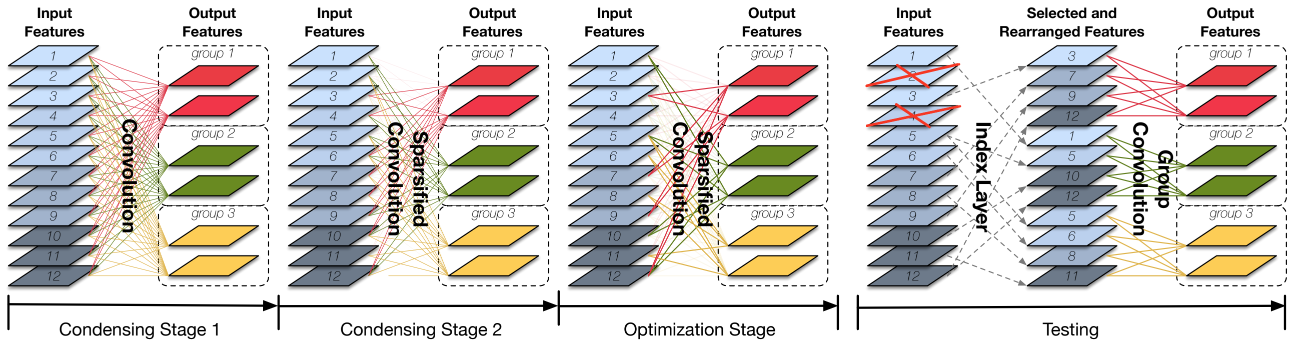

Figure 3. Illustration of learned group convolutions with $G=3$ groups and a condensation factor of $C=3$. During training a fraction of $(C-1)/C$ connections are removed after each of the $C-1$ condensing stages. Filters from the same group use the same set of features, and during test-time the *index layer* rearranges the features to allow the resulting model to be implemented as standard group convolutions.

a convolutional filter (one per output) over *all* R input features, leading to a computational cost of $R\times O$. In comparison, group convolution (right illustration) reduces this computational cost by partitioning the input features into G mutually exclusive groups, each producing its own outputs—reducing the computational cost by a factor G to $\frac{R\times O}{G}$.

3. CondenseNets

Group convolution works well with many deep neural network architectures [43, 46, 47] that are connected in a layer-by-layer fashion. For dense architectures group convolution can be used in the 3×3 convolutional layer (see Figure 1, left). However, preliminary experiments show that a naïve adaptation of group convolutions in the 1×1 convolutional layer leads to drastic reductions in accuracy. We surmise that this is caused by the fact that the inputs to the 1×1 convolutional layer are concatenations of feature maps generated by preceding layers. Therefore, they differ in two ways from typical inputs to convolutional layers: 1. they have an intrinsic order; and 2. they are far more diverse. The hard assignment of these features to disjoint groups hinders effective re-use of features in the network. Experiments in which we randomly permute input feature maps in each layer before performing the group convolution show that this reduces the negative impact on accuracy — but even with the random permutation, group convolution in the 1×1 convolutional layer makes DenseNets less accurate than for example smaller DenseNets with equivalent computational cost.

It is shown in [19] that making early features available as inputs to later layers is important for efficient feature re-use. Although not all prior features are needed at every subsequent layer, it is hard to predict which features should be utilized at what point. To address this problem, we develop an approach that *learns* the input feature groupings automatically during training. Learning the group structure allows each filter group to select its own set of most relevant inputs. Further, we allow multiple groups to share input features and also allow features to be ignored by all groups. Note that in a DenseNEt, even if an input feature is ignored by all groups in a specific layer, it can still be utilized by some groups At different layers. To differentiate it from regular group convolutions, we refer to our approach as *learned group convolution*.

3.1. Learned Group Convolution

We learn group convolutions through a multi-stage process, illustrated in Figures 3 and 4. The first half of the training iterations comprises of *condensing* stages. Here, we repeatedly train the network with sparsity inducing regularization for a fixed number of iterations and subsequently prune away unimportant filters with low magnitude weights. The second half of the training consists of the *optimization* stage, in which we learn the filters after the groupings are fixed. When performing the pruning, we ensure that filters from the same group share the same sparsity pattern. As a result, the sparsified layer can be implemented using a standard group convolution once training is completed (*testing* stage). Because group convolutions are efficiently implemented by many deep-learning libraries, this leads to high computational savings both in theory and in practice. We present details on our approach below.

Filter Groups. We start with a standard convolution of which filter weights form a 4D tensor of size $O\times R\times W\times H$, where O, R, W, and H denote the number of output channels, the number of input channels, and the width and the height of the filter kernels, respectively. As we are focusing on the 1×1 convolutional layer in DenseNets, the 4D tensor reduces to an $O\times R$ matrix $\mathbf{F}$. We consider the simplified case in this paper. But our procedure can readily be used with larger convolutional kernels. Before training, we first split the filters (or, equivalently, the output features) into G groups of equal size. We denote the filter weights for these groups by $\mathbf{F}^1,\ldots,\mathbf{F}^G$; each $\mathbf{F}^g$ has size $\frac{O}{G}\times R$ and $\mathbf{F}^g_{ij}$ corresponds to the weight of the jth input for the ith output

within group g. Because the output features do not have an implicit ordering, this random grouping does not negatively affect the quality of the layer.

Condensation Criterion. During the training process we gradually screen out subsets of less important input features for each group. The importance of the jth incoming feature map for the filter group g is evaluated by the averaged absolute value of weights between them across all outputs within the group, *i.e.*, by $\sum_{i=1}^{O/G} |\mathbf{F}_{i,j}^g|$. In other words, we remove columns in $\mathbf{F}^g$ (by zeroing them out) if their L_1-norm is small compared to the L_1-norm of other columns. This results in a convolutional layer that is structurally sparse: filters from the same group always receive the same set of features as input.

Group Lasso. To reduce the negative effects on accuracy introduced by weight pruning, L_1 regularization is commonly used to induce sparsity [29, 32]. In CondenseNets, we encourage convolutional filters from the same group to use the same subset of incoming features, *i.e.*, we induce group-level sparsity instead. To this end, we use the following group-lasso regularizer [44] during training:

$$\sum_{g=1}^{G} \sum_{j=1}^{R} \sqrt{\sum_{i=1}^{O/G} \mathbf{F}_{i,j}^{g\;2}}.$$

The group-lasso regularizer simultaneously pushes all the elements of a column of $\mathbf{F}^g$ to zero, because the term in the square root is dominated by the largest elements in that column. This induces the group-level sparsity we aim for.

Condensation Factor. In addition to the fact that learned group convolutions are able to automatically discover good connectivity patterns, they are also more flexible than standard group convolutions. In particular, the proportion of feature maps used by a group does not necessarily need to be $\frac{1}{G}$. We define a *condensation factor* C, which may differ from G, and allow each group to select $\lfloor \frac{R}{C} \rfloor$ of inputs.

Condensation Procedure. In contrast to approaches that prune weights in pre-trained networks, our weight pruning process is integrated into the training procedure. As illustrated in Figure 3 (which uses $C = 3$), at the end of each $C-1$ *condensing* stages we prune $\frac{1}{C}$ of the filter weights. By the end of training, only $\frac{1}{C}$ of the weights remain in each filter group. In all our experiments we set the number of training epochs of the condensing stages to $\frac{M}{2(C-1)}$, where M denotes the total number of training epochs—such that the first half of the training epochs is used for condensing. In the second half of the training process, the *Optimization* stage, we train the sparsified model.[2]

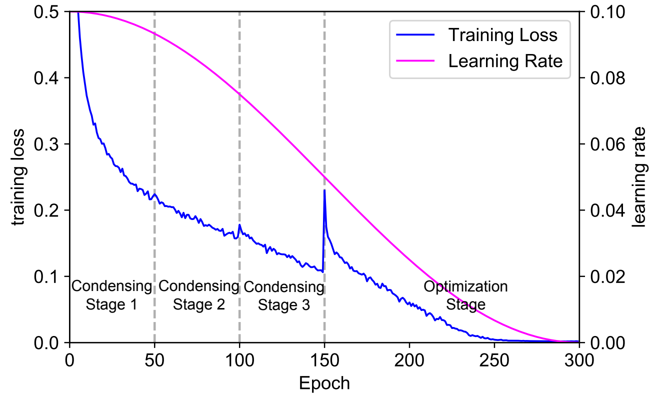

Figure 4. The cosine shape learning rate and a typical training loss curve with a condensation factor of $C = 4$.

Learning rate. We adopt the *cosine shape learning rate* schedule of Loshchilov et al. [33], which smoothly anneals the learning rate, and usually leads to improved accuracy [18, 49]. Figure 4 visualizes the learning rate as a function of training epoch (in magenta), and the corresponding training loss (blue curve) of a CondenseNet trained on the CIFAR-10 dataset [24]. The abrupt increase in the loss at epoch 150 is causes by the final condensation operation, which removes half of the remaining weights. However, the plot shows that the model gradually recovers from this pruning step in the optimization stage.

Index Layer. After training we remove the pruned weights and convert the sparsified model into a network with a regular connectivity pattern that can be efficiently deployed on devices with limited computational power. For this reason we introduce an *index layer* that implements the feature selection and rearrangement operation (see Figure 3, right). The convolutional filters in the output of the index layer are rearranged to be amenable to existing (and highly optimized) implementations of regular group convolution. Figure 1 shows the transformations of the CondenseNet layers during training (middle) and during testing (right). During training the 1×1 convolution is a learned group convolution (L-Conv), but during testing, with the help of the *index* layer, it becomes a standard group convolution (G-Conv).

3.2. Architecture Design

In addition to the use of learned group convolutions introduced above, we make two changes to the regular DenseNet architecture. These changes are designed to further simplify the architecture and improve its computational efficiency. Figure 5 illustrates the two changes that we made to the DenseNet architecture.

Exponentially increasing growth rate. The original DenseNet design adds k new feature maps at each layer,

[2] In our implementation of the *training* procedure we do not actually remove the pruned weights, but instead mask the filter $\mathbf{F}$ by a binary tensor $\mathbf{M}$ of the same size using an element-wise product. The mask is initialized with only ones, and elements corresponding to pruned weights are set to zero. This implementation via masking is more efficient on GPUs, as it does not require sparse matrix operations. In practice, the pruning hardly increases the wall time needed to perform a forward-backward pass during training.

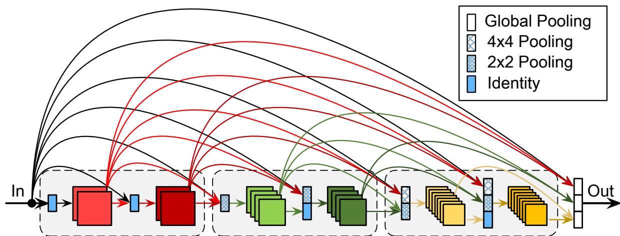

Figure 5. The proposed DenseNet variant. It differs from the original DenseNet in two ways: (1) layers with different resolution feature maps are also directly connected; (2) the growth rate doubles whenever the feature map size shrinks (far more features are generated in the third, yellow, dense block than in the first).

where k is a constant referred to as the *growth rate*. As shown in [19], deeper layers in a DenseNet tend to rely on high-level features more than on low-level features. This motivates us to improve the network by strengthening *short-range* connections. We found that this can be achieved by gradually increasing the growth rate as the depth grows. This increases the proportion of features coming from later layers relative to those from earlier layers. For simplicity, we set the growth rate to $k = 2^{m-1}k_0$, where m is the index of the dense block, and k_0 is a constant. This way of setting the growth rate does not introduce any additional hyper-parameters. The "increasing growth rate" (IGR) strategy places a larger proportion of parameters in the later layers of the model. This increases the computational efficiency substantially but may decrease the parameter efficiency in some cases. Depending on the specific hardware limitations it may be advantageous to trade-off one for the other [22].

Fully dense connectivity. To encourage feature re-use even more than the original DenseNet architecture does already, we connect input layers to *all* subsequent layers in the network, even if these layers are located in different dense blocks (see Figure 5). As dense blocks have different feature resolutions, we downsample feature maps with higher resolutions when we use them as inputs into lower-resolution layers using average pooling.

4. Experiments

We evaluate CondenseNets on the CIFAR-10, CIFAR-100 [24], and the ImageNet (ILSVRC 2012; [6]) image-classification datasets. The models and code reproducing our experiments are publicly available at `https://github.com/ShichenLiu/CondenseNet`.

Datasets. The CIFAR-10 and CIFAR-100 datasets consist of RGB images of size 32×32 pixels, corresponding to 10 and 100 classes, respectively. Both datasets contain 50,000 training images and 10,000 test images. We use a standard data-augmentation scheme [20, 26, 28, 30, 37, 39, 41], in which the images are zero-padded with 4 pixels on each side, randomly cropped to produce 32×32 images, and horizontally mirrored with probability 0.5.

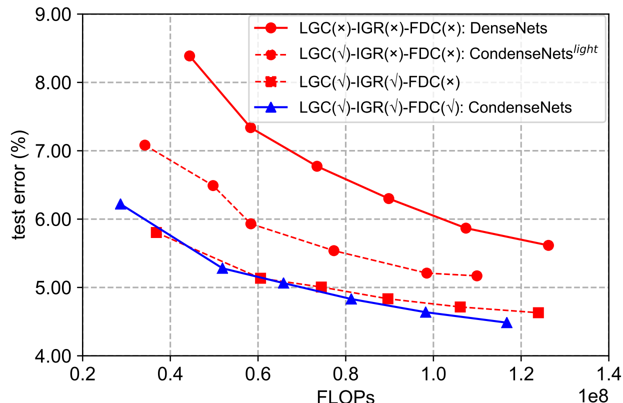

Figure 6. Ablation study on CIFAR-10 to investigate the efficiency gains obtained by the various components of CondenseNet.

The ImageNet dataset comprises 1000 visual classes, and contains a total of 1.2 million training images and 50,000 validation images. We adopt the data-augmentation scheme of [12] at training time, and perform a rescaling to 256×256 followed by a 224×224 center crop at test time before feeding the input image into the networks.

4.1. Results on CIFAR

We first perform a set of experiments on CIFAR-10 and CIFAR-100 to validate the effectiveness of learned group convolutions and the proposed CondenseNet architecture.

Model configurations. Unless otherwise specified, we use the following network configurations in all experiments on the CIFAR datasets. The standard DenseNet has a constant growth rate of $k = 12$ following [19]; our proposed architecture uses growth rates $k_0 \in \{8,16,32\}$ to ensure that the growth rate is divisable by the number of groups. The learned group convolution is only applied to the first convolutional layer (with filter size 1×1, see Figure 1) of each basic layer, with a condensation factor of $C = 4$, *i.e.*, 75% of filter weights are gradually pruned during training with a step of 25%. The 3×3 convolutional layers are replaced by standard group convolution (without applying learned group convolution) with four groups. Following [46, 47], we permute the output channels of the first 1×1 learned group convolutional layer, such that the features generated by each of its groups are evenly used by all the groups of the subsequent 3×3 group convolutional layer .

Training details. We train all models with stochastic gradient descent (SGD) using similar optimization hyper-parameters as in [12, 19]. Specifically, we adopt Nesterov momentum with a momentum weight of 0.9 without dampening, and use a weight decay of 10^{-4}. All models are trained with mini-batch size 64 for 300 epochs, unless otherwise specified. We use a cosine shape learning rate which

starts from 0.1 and gradually reduces to 0. Dropout [40] with a drop rate of 0.1 was applied to train CondenseNets with >3 million parameters (shown in Table 1).

Component analysis. Figure 6 compares the computational efficiency gains obtained by each component of CondenseNet: learned group convolution (LGR), exponentially increasing learning rate (IGR), full dense connectivity (FDC). Specifically, the figure plots the test error as a function of the number of FLOPs (*i.e.*, multiply-addition operations). The large gap between the two red curves with dot markers shows that learned group convolution significantly improves the efficiency of our models. Compared to DenseNets, CondenseNet^{light} only requires half the number of FLOPs to achieve comparable accuracy. Further, we observe that the exponentially increasing growth rate, yields even further efficiency. Full dense connectivity does not boost the efficiency significantly on CIFAR-10, but there does appear to be a trend that as models getting larger, full connectivity starts to help. We opt to include this architecture change in the CondenseNet model, as it does lead to substantial improvements on ImageNet (see later).

Comparison with state-of-the-art efficient CNNs. In Table 1, we show the results of experiments comparing a 160-layer CondenseNet^{light} and a 182-layer CondenseNet with alternative state-of-the-art CNN architectures. Following [49], our models were trained for 600 epochs. From the results, we observe that CondenseNet requires approximately $8\times$ fewer parameters and FLOPs to achieve a comparable accuracy to DenseNet-190. CondenseNet seems to be less parameter-efficient than CondenseNet^{light}, but is more compute-efficient. Somewhat surprisingly, our CondenseNet^{light} model performs on par with the NASNet-A, an architecture that was obtained using an automated search procedure over $20,000$ candidate architectures composed of a rich set of components, and is thus carefully tuned on the CIFAR-10 dataset [49]. Moreover, CondenseNet (or CondenseNet^{light}) does not use depth-wise separable convolutions, and only use simple convolutional filters with size 1×1 and 3×3. It may be possible to include CondenseNet as a meta-architecture in the procedure of [49] to obtain even more efficient networks.

Comparison with existing pruning techniques. In Table 2, we compare our CondenseNets and CondenseNets^{light} with models that are obtained by state-of-the-art *filter-level* weight pruning techniques [14, 29, 32]. The results show that, in general, CondenseNet is about $3\times$ more efficient in terms of FLOPs than ResNets or DenseNets pruned by the method introduced in [32]. The advantage over the other pruning techniques is even more pronounced. We also report the results for CondenseNet^{light} in the second last row of Table 2. It uses only half the number of parameters to achieve comparable performance as the most competitive baseline, the 40-layer DenseNet described by [32].

Model	Params	FLOPs	C-10	C-100
ResNet-1001 [13]	16.1M	2,357M	4.62	22.71
Stochastic-Depth-1202 [20]	19.4M	2,840M	4.91	-
Wide-ResNet-28 [45]	36.5M	5,248M	4.00	19.25
ResNeXt-29 [43]	68.1M	10,704M	3.58	17.31
DenseNet-190 [19]	25.6M	9,388M	3.46	17.18
NASNet-A* [49]	3.3M	-	3.41	-
CondenseNet^{light}-160*	3.1M	1,084M	3.46	17.55
CondenseNet-182*	4.2M	513M	3.76	18.47

Table 1. Comparison of classification error rate (%) with other convolutional networks on the CIFAR-10(C-10) and CIFAR-100(C-100) datasets. * indicates models that are trained with cosine shape learning rate for 600 epochs.

Model	FLOPs	Params	C-10	C-100
VGG-16-pruned [29]	206M	5.40M	6.60	25.28
VGG-19-pruned [32]	195M	2.30M	6.20	-
VGG-19-pruned [32]	250M	5.00M	-	26.52
ResNet-56-pruned [14]	62M		8.20	-
ResNet-56-pruned [29]	90M	0.73M	6.94	-
ResNet-110-pruned [29]	213M	1.68M	6.45	-
ResNet-164-B-pruned [32]	124M	1.21M	5.27	23.91
DenseNet-40-pruned [32]	190M	0.66M	5.19	25.28
CondenseNet^{light}-94	122M	0.33M	5.00	24.08
CondenseNet-86	65M	0.52M	5.00	23.64

Table 2. Comparison of classification error rate (%) on CIFAR-10 (C-10) and CIFAR-100 (C-100) with state-of-the-art filter-level weight pruning methods.

CondenseNet	Feature map size
3×3 Conv (stride 2)	112×112
$\begin{bmatrix}1\times1 \text{ L-Conv}\\3\times3 \text{ G-Conv}\end{bmatrix}\times4 \quad (k=8)$	112×112
2×2 average pool, stride 2	56×56
$\begin{bmatrix}1\times1 \text{ L-Conv}\\3\times3 \text{ G-Conv}\end{bmatrix}\times6 \quad (k=16)$	56×56
2×2 average pool, stride 2	28×28
$\begin{bmatrix}1\times1 \text{ L-Conv}\\3\times3 \text{ G-Conv}\end{bmatrix}\times8 \quad (k=32)$	28×28
2×2 average pool, stride 2	14×14
$\begin{bmatrix}1\times1 \text{ L-Conv}\\3\times3 \text{ G-Conv}\end{bmatrix}\times10 \quad (k=64)$	14×14
2×2 average pool, stride 2	7×7
$\begin{bmatrix}1\times1 \text{ L-Conv}\\3\times3 \text{ G-Conv}\end{bmatrix}\times8 \quad (k=128)$	7×7
7×7 global average pool	1×1
1000-dim fully-connected, softmax	

Table 3. CondenseNet architectures for ImageNet.

4.2. Results on ImageNet

In a second set of experiments, we test CondenseNet on the ImageNet dataset.

Model configurations. Detailed network configurations are shown in Table 3. To reduce the number of parameters, we prune 50% of weights from the fully connected (FC) layer at epoch 60 in a way similar to the learned group convolution, but with $G=1$ (as the FC layer could not be split

Model	FLOPs	Params	Top-1	Top-5
Inception V1 [42]	1,448M	6.6M	30.2	10.1
1.0 MobileNet-224 [16]	569M	4.2M	29.4	10.5
ShuffleNet 2x [47]	524M	5.3M	29.1	10.2
NASNet-A (N=4) [49]	564M	5.3M	26.0	8.4
NASNet-B (N=4) [49]	488M	5.3M	27.2	8.7
NASNet-C (N=3) [49]	558M	4.9M	27.5	9.0
CondenseNet ($G=C=8$)	274M	2.9M	29.0	10.0
CondenseNet ($G=C=4$)	529M	4.8M	26.2	8.3

Table 4. Comparison of *Top-1* and *Top-5* classification error rate (%) with other state-of-the-art compact models on ImageNet.

into multiple groups) and $C=2$. Similar to prior studies on MobileNets and ShuffleNets, we focus on training relatively small models that require less than 600 million FLOPs to perform inference on a single image.

Training details. We train all models using stochastic gradient descent (SGD) with a batch size of 256. As before, we adopt Nesterov momentum with a momentum weight of 0.9 without dampening, and a weight decay of 10^{-4}. All models are trained for 120 epochs, with a cosine shape learning rate which starts from 0.1 and gradually reduces to 0. We use group lasso regularization in all experiments on ImageNet; the regularization parameter is set to 10^{-5}.

Comparison with state-of-the-art efficient CNNs. Table 4 shows the results of CondenseNets and several state-of-the-art, efficient models on the ImageNet dataset. We observe that a CondenseNet with 274 million FLOPs obtains a 29.0% Top-1 error, which is comparable to the accuracy achieved by MobileNets and ShuffleNets that require twice as much compute. A CondenseNet with 529 million FLOPs produces to a 3% absolute reduction in top-1 error compared to a MobileNet and a ShuffleNet of comparable size. Our CondenseNet even achieves a the same accuracy with slightly fewer FLOPs and parameters than the most competitive NASNet-A, despite the fact that we only trained a very small number of models (as opposed to the study that lead to the NASNet-A model).

Actual inference time. Table 5 shows the actual inference time on an ARM processor for different models. The wall-time to inference an image sized at 224×224 is highly correlated with the number of FLOPs of the model. Compared to the recently proposed MobileNet, our CondenseNet ($G=C=8$) with 274 million FLOPs inferences an image $2\times$ faster, while without sacrificing accuracy.

4.3. Ablation Study

We perform an ablation study on CIFAR-10 in which we investigate the effect of (1) the pruning strategy, (2) the number of groups, and (3) the condensation factor. We also investigate the stability of our weight pruning procedure.

Pruning strategy. The left panel of Figure 7 compares our *on-the-fly* pruning method with the more common approach

Model	FLOPs	Top-1	Time(s)
VGG-16	15,300M	28.5	354
ResNet-18	1,818M	30.2	8.14
1.0 MobileNet-224 [16]	569M	29.4	1.96
CondenseNet ($G=C=4$)	529M	26.2	1.89
CondenseNet ($G=C=8$)	274M	29.0	0.99

Table 5. Actual inference time of different models on an ARM processor. All models are trained on ImageNet, and accept input with resolution 224×224.

of pruning weights of *fully converged* models. We use a DenseNet with 50 layers as the basis for this experiment. We implement a "traditional" pruning method in which the weights are pruned in the same way as in as in CondenseNets, but the pruning is only done once after training has completed (for 300 epochs). Following [32], we fine-tune the resulting sparsely connected network for another 300 epochs with the same cosine shape learning rate that we use for training CondenseNets. We compare the traditional pruning approach with the CondenseNet approach, setting the number of groups G is set to 4. In both settings, we vary the condensation factor C between 2 and 8.

The results in Figure 7 show that pruning weights gradually during training outperforms pruning weights on fully trained models. Moreover, gradual weight pruning reduces the training time: the "traditional pruning" models were trained for 600 epochs, whereas the CondenseNets were trained for 300 epochs. The results also show that removing 50% the weights (by setting $C=2$) from the 1×1 convolutional layers in a DenseNet incurs hardly any loss in accuracy.

Number of groups. In the middle panel of Figure 7, we compare four CondenseNets with exactly the same network architecture, but a number of groups, G, that varies between 1 and 8. We fix the condensation factor, C, to 8 for all the models, which implies all models have the same number of parameters after training has completed. In CondenseNets with a single group, we discard entire filters in the same way that is common in filter-pruning techniques [29, 32]. The results presented in the figure demonstrate that test errors tends to decrease as the number of groups increases. This result is in line with our analysis in Section 3, in particular, it suggests that grouping filters gives the training algorithm more flexibility to remove redundant weights.

Effect of the condensation factor. In the right panel of Figure 7, we compare CondenseNets with varying condensation factors. Specifically, we set the condensation factor C to 1, 2, 4, or 8; this corresponds to removing 0%, 50%, 75%, or 87.5% of the weights from each of the 1×1 convolutional layers, respectively. A condensation factor $C=1$ corresponds to a baseline model without weight pruning. The number of groups, G, is set to 4 for all the networks. The results show that a condensation factors C larger than 1 con-

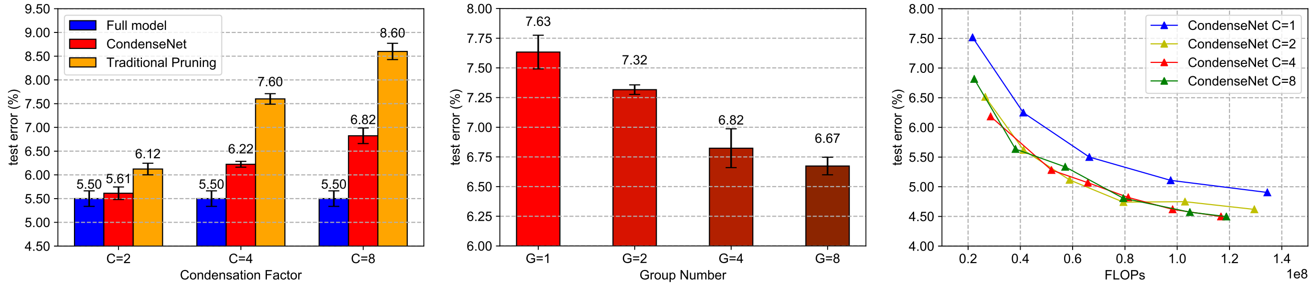

Figure 7. Classification error rate (%) on CIFAR-10. *Left*: Comparison between our condense method with traditional pruning approach, under varying condensation factors. *Middle*: CondenseNets with different number of groups for the 1×1 learned group convolution. All the models have the same number of parameters. *Right*: CondenseNets with different condensation factors.

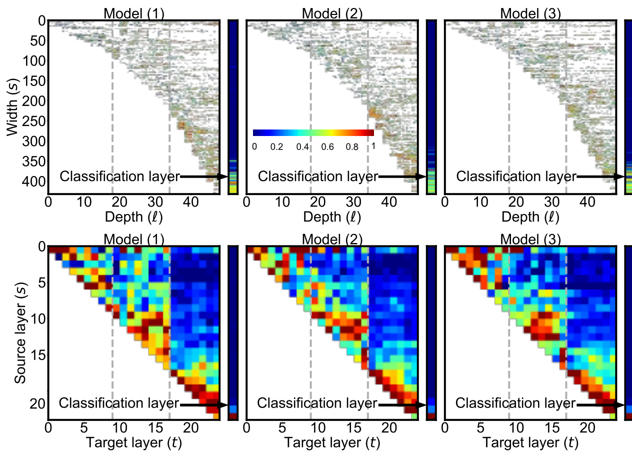

Figure 8. Norm of weights between layers of a CIFAR-10 CondenseNet per filter group (*top*) and per filter block (*bottom*). The three columns correspond to independent training runs.

sistently lead to improved efficiency, which underlines the effectiveness of our method. Interestingly, models with condensation factors 2, 4 and 8 perform comparably in terms of classification error as a function of FLOPs. This suggests that whilst pruning more weights yields smaller models, it also leads to a proportional loss in accuracy.

Stability. As our method removes redundant weights in early stages of the training process, a natural question is whether this will introduce extra variance into the training. Does early pruning remove some of the weights simply because they were initialized with small values?

To investigate this question, Figure 8 visualizes the learned weights and connections for three independently trained CondenseNets on CIFAR-10 (using different random seeds). The top row shows detailed weight strengths (averaged absolute value of non-pruned weights) between a filter group of a certain layer (corresponding to a column in the figure) and an input feature map (corresponding to a row in the figure). For each layer there are four filter groups (consecutive columns). A white pixel in the top-right corner indicates that a particular input feature was pruned by that layer and group. Following [19], the bottom row of Figure fig:learned-weights-stablity shows the overall connection strength between two layers in the condensed network. The vertical bars correspond to the linear classification layer on top of the CondenseNet. The gray vertical dotted lines correspond to pooling layers that decrease the feature resolution.

The results in the figure suggest that while there are differences in learned connectivity at the filter-group level (top row), the overall information flow between layers (bottom row) is similar for all three models. This suggests that the three training runs learn similar global connectivity patterns, despite starting from different random initializations. Later layers tend to prefer more recently generated features, do however utilize some features from very early layers.

5. Conclusion

In this paper, we introduced CondenseNet: an efficient convolutional network architecture that encourages feature re-use via dense connectivity and prunes filters associated with superfluous feature re-use via learned group convolutions. To make inference efficient, the pruned network can be converted into a network with regular group convolutions, which are implemented efficiently in most deep-learning libraries. Our pruning method is simple to implement, and adds only limited computational costs to the training process. In our experiments, CondenseNets outperform recently proposed MobileNets and ShuffleNets in terms of computational efficiency at the same accuracy level. CondenseNet even slightly outperforms a network architecture that was discovered by empirically trying tens of thousands of convolutional network architectures, and with a much simpler structure.

Acknowledgements. The authors are supported in part by grants from the National Science Foundation (III-1525919, IIS-1550179, IIS-1618134, S&AS 1724282, and CCF-1740822), the Office of Naval Research DOD (N00014-17-1-2175), and the Bill and Melinda Gates Foundation. We thank Xu Zou, Weijia Chen, Danlu Chen for helpful discussions.

References

[1] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[2] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. Adaptive neural networks for fast test-time prediction. In *ICML*, 2017.

[3] C. Bucilua, R. Caruana, and A. Niculescu-Mizil. Model compression. In *ACM SIGKDD*, pages 535–541, 2006.

[4] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *ICML*, pages 2285–2294, 2015.

[5] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[7] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. *arXiv preprint arXiv:1612.02297*, 2016.

[8] A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[9] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[10] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, pages 1135–1143, 2015.

[11] B. Hassibi, D. G. Stork, and G. J. Wolff. Optimal brain surgeon and general network pruning. In *IJCNN*, pages 293–299, 1993.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[14] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. 2017.

[15] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*, 2014.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[17] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense networks for resource efficient image classification. In *ICLR*, 2018.

[18] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. In *ICLR*, 2017.

[19] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[20] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *NIPS*, pages 4107–4115, 2016.

[22] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[23] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[24] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. In *Tech Report*, 2009.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[26] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[27] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPS*, volume 2, pages 598–605, 1989.

[28] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *AISTATS*, 2015.

[29] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[30] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014.

[31] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014.

[32] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[33] I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with restarts. In *ICLR*, 2017.

[34] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally. Exploring the Regularity of Sparse Structure in Convolutional Neural Networks.

[35] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML-10*, pages 807–814, 2010.

[36] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542. Springer, 2016.

[37] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015.

[38] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[39] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[40] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[41] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *NIPS*, 2015.

[42] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[43] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[44] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[45] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[46] T. Zhang, G.-J. Qi, B. Xiao, and J. Wang. Interleaved group convolutions for deep neural networks. In *ICCV*, 2017.

[47] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

[48] L. Zhao, J. Wang, X. Li, Z. Tu, and W. Zeng. Deep convolutional neural networks with merge-and-run mappings. 2016.

[49] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.