Training CNNs with Selective Allocation of Channels

Jongheon Jeong [1] **Jinwoo Shin** [1 2 3]

Abstract

Recent progress in deep convolutional neural networks (CNNs) have enabled a simple paradigm of architecture design: *larger models typically achieve better accuracy*. Due to this, in modern CNN architectures, it becomes more important to design models that generalize well under certain resource constraints, e.g. the number of parameters. In this paper, we propose a simple way to improve the capacity of any CNN model having large-scale features, without adding more parameters. In particular, we modify a standard convolutional layer to have a new functionality of *channel-selectivity*, so that the layer is trained to select important channels to re-distribute their parameters. Our experimental results under various CNN architectures and datasets demonstrate that the proposed new convolutional layer allows new optima that generalize better via efficient resource utilization, compared to the baseline.

1. Introduction

Convolutional neural networks (CNNs) have become one of the most effective approaches for various tasks of machine learning. With a growing interest, there has been a lot of works on designing advanced CNN architectures (Szegedy et al., 2015; Simonyan & Zisserman, 2014; Ioffe & Szegedy, 2015; He et al., 2016a). Although modern CNNs are capable to scale over a thousand of layers (He et al., 2016b) or channels (Huang et al., 2017), deploying them in the real-world becomes increasingly difficult due to computing resource constraints. This has motivated the recent literature such as resource-efficient architectures (Huang et al., 2018b; Sandler et al., 2018; Ma et al., 2018), low-rank factorization (Jaderberg et al., 2014; Novikov et al., 2015), weight quantization (Rastegari et al., 2016; Courbariaux & Bengio, 2016; Chen et al., 2018) and anytime/adaptive networks (Figurnov et al., 2017; Bolukbasi et al., 2017; Huang et al., 2018a).

In order to design a resource-efficient CNN architecture, it is important to process succinct representations of large-scale features. At this point of view, there have been continuous attempts to find an efficient layer for handling such extremely large number of features (Iandola et al., 2016; Ioannou et al., 2017; Sun et al., 2018; Sandler et al., 2018; Ma et al., 2018). However, most prior works assume that the layer is *static*, i.e., the structure in weight connectivity is unchanged during training. Such static layers inevitably have to allocate too many parameters across *homogeneous* features, since it is hard to get prior knowledge on the features before training the network. For instance, one of state-of-the-art models, DenseNet-BC-190 (Huang et al., 2017), devotes 70% of the parameters for just performing dimensionality reduction of pointwise convolutional layers. Such an architectural inefficiency may harm the generalization ability of the model, given a fixed number of parameters.

To alleviate the issue of inefficient allocation of parameters, one can attempt to utilize the posterior information after training, e.g. network pruning (Han et al., 2015; He et al., 2017; Liu et al., 2017), or neural architecture search (Zoph et al., 2018; Real et al., 2018; Luo et al., 2018). A shortcoming of this direction, however, is that it typically requires a time-consuming repetition of training cycles.

Contribution. In this paper, we propose a new way of training CNNs so that each convolutional layer can select channels of importance dynamically during training. As the training progresses, some input channels of a convolutional layer may have almost no contribution to the output, wasting the resources allocated to the channels for the rest of the training. Our method detects such channels, and re-distribute the resources from those channels to another top-K selected channels of importance. Consequently, our training scheme is a process that increases the efficiency of CNN by dynamically pruning or re-wiring its parameters on-the-fly along with learning them. In a sense, our method "imitates" how hippocampus in brain learns, where new neurons are generated and rewired daily under maintenance via neuronal apoptosis or pruning (Sahay et al., 2011a;b).

Our CNN-training method consists of two building blocks. First, we propose the *expected channel damage matrix*

[1]School of Electrical Engineering, KAIST, Daejeon, South Korea [2]Graduate School of AI, KAIST, Daejeon, South Korea [3]AITRICS, Seoul, South Korea. Correspondence to: Jinwoo Shin <jinwoos@kaist.ac.kr>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

(ECDM), which estimates the changes of the output vector given each channel is damaged (or removed). This provides a safe criterion for selecting channels to remove (or to emphasize) during training. Second, we impose *spatial shifting bias* for effective recycling of parameters. It turns out this allows a convolutional layer to "enlarge" the convolutional kernel selectively to important channels only.

We evaluate our method on CIFAR-10/100, Fashion-MNIST, Tiny-ImageNet, and ImageNet classification datasets with a wide range of recent CNN architectures, including ResNet (He et al., 2016a) and DenseNet (Huang et al., 2017). Despite of its simplicity, our experimental results show that training with channel-selectivity consistently improves accuracy over its counterpart across all tested architectures. For example, the proposed selective convolutional layer applied to DenseNet-40 provides 8.01% relative reduction in test error rates for CIFAR-10. Next, we show that our method can also be used for model compression. By applying our method to a highly-efficient CondenseNet (Huang et al., 2018b), we could further improve its efficiency: the resulting model has 25× fewer FLOPs compared to ResNeXt-29 (Xie et al., 2017), while achieving better accuracy.

Compared to the significant interests on pruning parameters during training, i.e., network sparsity learning (Wen et al., 2016; Molchanov et al., 2017; Neklyudov et al., 2017; Louizos et al., 2017; 2018; Dai et al., 2018), the progress is arguably slower on *re-wiring* the pruned parameters to maximize its utility. Han et al. (2016) proposed Dense-Sparse-Dense (DSD) training flow, showing that re-training after re-initialization of the pruned connections can further improve accuracy. Dynamic network surgery (Guo et al., 2016) introduced a method of splicing the pruned connections to recover the possibly mis-pruned ones, showing better compression performances. The recently proposed MorphNet (Gordon et al., 2018) attempts to find an optimal widths of each layer from shirinking and expanding a given DNN through iterative training passes.

Our approach proposes a new way of re-wiring, with several advantages over the existing methods: (a) *generic, easy-to-use*: it can be applied to train any kind of CNN, (b) *single-pass*: it does not require any post-processing or re-training as it is seamlessly integrated into existing training schemes, and (c) *flexibility*: it allows to easily balance between accuracy improvement and model compression on-demand. We believe our work provides a new direction on the important problem of training CNNs more efficiently.

2. Selective Convolution

Our goal is to design a new convolutional layer which can replace any existing one, with improved utilization of network parameters via selecting channels of importance. We call the proposed layer *selective convolution*. We train this

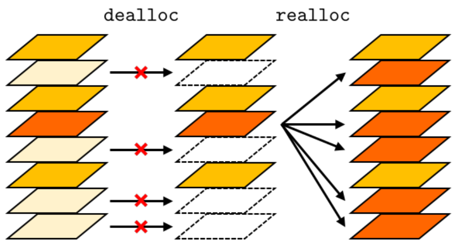


Figure 1. Illustration of channel de-allocation and re-allocation procedures. The higher the saturation of the channel color, the higher the channel importance.

layer via two operations that make a *re-distribution* of the given input channels:

1. *Channel de-allocation* (`dealloc`): Obstruct unnecessary channels from being used in future computations, and release the corresponding parameters.

2. *Channel re-allocation* (`realloc`): Overwrite top-K important channels into the obstructed areas, and recycle the parameters in there.

Figure 1 illustrates the two basic operations. More details of `dealloc` and `realloc` are described in Section 2.2.

During training a neural network with selective convolutional layers, the channel-selectivity is obtained by simply calling `dealloc` and `realloc` for each chosen layer on demand along with the standard stochastic gradient descent (SGD) methods. Repeating `dealloc` and `realloc` alternatively translates the original input to what has only a few important channels, potentially duplicated multiple times. Namely, the parameters originally allocated to handle the entire input now operate on its important subset.

We aim to design `dealloc` and `realloc` to be *function-preserving*, i.e. they do not change the output of the convolution. This allows us to call them anytime during SGD training without damaging the network output. On the other hand, since the resource released from `dealloc` is limited, it is also important for `realloc` to choose channels that will maximize resource utilization. This motivates us to design for those operations a more delicate metric of channel importance than other existing magnitude-based metrics, e.g., weight ℓ^2-norm (Li et al., 2016). To this end, we propose *expected channel damage matrix* (ECDM) in Section 2.1, which leads to an efficient and safe way of identifying channels with low contribution to the output. We provide the architectural description of selective convolution in Section 2.2, and the detailed training scheme using ECDM in Section 2.3.

2.1. Expected Channel Damage Matrix (ECDM)

To begin with, we let $\mathrm{Conv}(\mathbf{X};\mathbf{W})$ to denote a convolutional layer (or function) for its weight $\mathbf{W} \in \mathbb{R}^{I\times O\times K^2}$

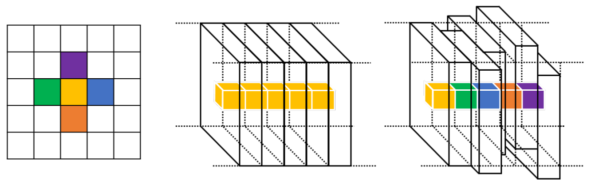


Figure 2. The kernel enlarging effect of spatial shifting. The colored strides indicate a single patch of computation in a convolutional layer assuming the kernel size is 1 for simplicity.

and its input random variable $\mathbf{X} \in \mathbb{R}^{I \times H \times W}$. Here, I and O denote the number of input and output channels, respectively, H and W are the height and width of the input, and K denotes the kernel size.

Expected channel damage matrix (ECDM) is designed for measuring the expected functional difference

$$\mathbb{E}_{\mathbf{X}}[\mathrm{Conv}(\mathbf{X}; \mathbf{W}) - \mathrm{Conv}(\mathbf{X}; \mathbf{W}_{-i})],$$

where $\mathbf{W}_{-i}$ is identical to $\mathbf{W}$ but $\mathbf{W}_{i,:,:}$ is set to 0. In other words, it measures the expected amount of changes in output when i-th channel is "damaged" or "pruned". Remark that this quantity is directly related to the function-preserving property we want to achieve. For $i = 1, \dots, I$, we define $\mathrm{ECDM}(\mathbf{W}; \mathbf{X})_i$ by averaging the expectation over the spatial dimensions:

$$\begin{aligned}&\mathrm{ECDM}(\mathbf{W}; \mathbf{X})_i \in \mathbb{R}^O \\ &:= \frac{1}{HW} \sum_{h,w} \mathbb{E}_{\mathbf{X}}[\mathrm{Conv}(\mathbf{X}; \mathbf{W}) - \mathrm{Conv}(\mathbf{X}; \mathbf{W}_{-i})]_{:,h,w}.\end{aligned}$$

Notice that the above definition requires a marginalization over $\mathbf{X}$. One can estimate it via Monte Carlo sampling using training data, but it can be computationally too expensive if it is used repeatedly during training. Instead, we propose a simple approximation of ECDM utilizing batch normalization (BN) layer (Ioffe & Szegedy, 2015) to infer the current input distribution at any time of training, in what follows.

Consider a hidden neuron x following BN and ReLU nonlinearity (Nair & Hinton, 2010), i.e. $y = \mathrm{ReLU}(\mathrm{BN}(x))$, and suppose one wants to estimate $\mathbb{E}[y]$ *without* sampling. To this end, we exploit the fact that BN already "accumulates" its input statistics continuously throughout training. If we simply assume that $\mathrm{BN}(x) \sim \mathcal{N}(\beta, \gamma^2)$ (i.e. normal distribution), where γ and β are the scaling and shifting parameter of BN, respectively, it is elementary to check:

$$\mathbb{E}[y] = \mathbb{E}[\mathrm{ReLU}(\mathrm{BN}(x))] = |\gamma|\phi_{\mathcal{N}}\left(\frac{\beta}{|\gamma|}\right) + \beta\Phi_{\mathcal{N}}\left(\frac{\beta}{|\gamma|}\right), \tag{1}$$

where $\phi_{\mathcal{N}}$ and $\Phi_{\mathcal{N}}$ denote the p.d.f. and the c.d.f. of the standard normal distribution, respectively.

The idea is directly extended to obtain a closed form approximation of $\mathrm{ECDM}(\mathbf{W}; \mathbf{X})$ when $\mathbf{X}$ is from $\mathrm{ReLU}(\mathrm{BN}(\cdot))$. In practice, this assumption is quite reasonable as many of nowadays CNNs adopt this BN $\to$ ReLU $\to$ Conv as a building block of designing a model (He et al., 2016a; Huang et al., 2017; Xie et al., 2017; Chen et al., 2017). Under assuming that $\mathbf{X}_{i,h,w} \sim \max(\mathcal{N}(\beta_i, \gamma_i^2), 0)$ for all i, h, w, we obtain that for $i = 1, \dots, I$:

$$\begin{aligned}&\mathrm{ECDM}(\mathbf{W}; \mathbf{X})_i \\ &= \underbrace{\left(|\gamma_i|\phi_{\mathcal{N}}\left(\frac{\beta_i}{|\gamma_i|}\right) + \beta_i\Phi_{\mathcal{N}}\left(\frac{\beta_i}{|\gamma_i|}\right)\right)}_{\text{(a)}} \cdot \underbrace{\sum_{k=1}^{K^2} \mathbf{W}_{i,:,k}}_{\text{(b)}},\end{aligned} \tag{2}$$

where the above equality follows from the linearity of convolutional layer, the linearity of expectation, and (1). The detailed derivation is given in the supplementary material.

There are two main terms in (2): (a) measures the overall activity level of the i-th channel from BN statistics, and (b) does the sum of weights related to the channel. Therefore, it allows a way to capture not only "low-magnitude" channels, but also channels of "*low-contribution*" under the distribution of $\mathbf{X}$. On the other hand, existing other magnitude-based metrics (Li et al., 2016; Liu et al., 2017; Neklyudov et al., 2017) typically aim only for the former.

2.2. Selective Convolutional Layer

Any CNN model can have the de/re-allocation mechanism in its training, simply by replacing each convolutional layer $\mathrm{Conv}(\mathbf{X}; \mathbf{W})$ with the proposed *selective convolutional layer* $\mathrm{SelectConv}(\mathbf{X}; \mathbf{W})$. Compared to the standard convolution, SelectConv has an additional layer that rebuilds an input in channel-wise:

$$\mathrm{SelectConv}(\mathbf{X}; \mathbf{W}) := \mathrm{Conv}(\mathrm{SelectChannel}(\mathbf{X}); \mathbf{W}).$$

In essence, SelectChannel requires to perform channel blocking and re-indexing for `dealloc` and `realloc`, respectively. One can implement this layer by:

$$\mathrm{SelectChannel}(\mathbf{X}; \mathbf{g}, \boldsymbol{\pi})_i := g_i \cdot \mathbf{X}_{\pi_i}, \tag{3}$$

for indices $\pi_i \in \{1, 2, \dots, I\}$ and gate variables $g_i \in \{0, 1\}$ for $i = 1, \dots, I$. Here, multiple π_i's can be the same, i.e. a channel is copied multiple times, and $g_i = 0$ means the input channel $\mathbf{X}_{\pi_i}$ is blocked. In the case that a channel is copied N times, the convolution will process the channel with N times more parameters compared to the standard processing. However, naïvely copying a channel in (3) does not give any benefit of using more parameters, due to the linearity of convolutional layer: if two input channels are identical, the corresponding weights are degenerated. To address this issue, we impose *spatial shifting biases* $b_i = (b_i^h, b_i^w) \in \mathbb{R}^2$ for re-allocated channels: we re-define SelectChannel as

$$\mathrm{SelectChannel}(\mathbf{X}; \mathbf{g}, \boldsymbol{\pi}, \mathbf{b})_i := g_i \cdot \mathrm{shift}(\mathbf{X}_{\pi_i}, b_i),$$

where $\text{shift}(\mathbf{X}, b)$ denotes the spatial shifting operation on $\mathbf{X}$. For each pixel (x, y), we define $\text{shift}(\mathbf{X}, b)_{x,y}$ as:

$$\begin{aligned}\text{shift}\left(\mathbf{X}, b\right)_{x,y} := &\sum_{n=1}^{H}\sum_{m=1}^{W} \mathbf{X}_{n,m} \\ &\times \max\left(0, 1 - |x - n + b^h|\right) \\ &\times \max\left(0, 1 - |y - m + b^w|\right)\end{aligned}$$

using a bilinear interpolation kernel. This formulation allows b to be continuous real values, thereby to be learned via SGD with other parameters jointly. We remark that similar spatial shifting operations have recently gained attention in the area of CNN architecture design (Jeon & Kim, 2017; Dai et al., 2017; Wu et al., 2018; Jeon & Kim, 2018), with their efficient implementations. This trick encourages to utilize the re-allocated parameters effectively, as it provides diversity on the copied channels when the convolution is applied. Essentially, as illustrated in Figure 2, it provides an effect of *enlarging* the convolutional kernel, in particular, for the re-allocated channels only. In other words, our method recycles its parameters by *selectively* expanding the kernel of important channels.

2.3. Training Scheme: Channel De/Re-allocation

Given a selective convolutional layer with parameters $\mathbf{S} = (\mathbf{W}, \mathbf{g}, \boldsymbol{\pi}, \mathbf{b})$, we design `dealloc` and `realloc` to train $\mathbf{S}$. For example, once some channels are chosen to be de-allocated, the actual operation can be done by just setting $g_i = 0$ for the channels. We utilize ECDM in order to identify channels to be de/re-allocated. Given a desired damage level $\gamma > 0$, the objective of `dealloc` can be written as the following optimization problem:

$$\begin{aligned}\underset{\mathbf{g}}{\text{minimize}} \quad & \sum_{i=1}^{I} g_i \\ \text{subject to} \quad & \left\| \sum_{i=1}^{I} (1 - g_i) \cdot \text{ECDM}(\mathbf{W}; \mathbf{X})_i \right\|_\infty \leq \gamma, \\ & g_i \in \{0, 1\},\ i = 1, \ldots, I.\end{aligned}$$

However, the above combinatorial optimization is computationally intractable (i.e., NP-hard) in general as it is reduced to the *0-1 multi-dimensional knapsack problem* (MKP) (Kellerer et al., 2004). Although many heuristics for MKP (Vasquez & Vimont, 2005; Raidl & Gottlieb, 2005) can be used for `dealloc`, we consider a simple greedy algorithm. First, we normalize ECDM with respect to the output dimension, namely *normalized-ECDM*[1] (nECDM):

$$\text{nECDM}(\mathbf{W}; \mathbf{X})_{:,j} := \frac{|\text{ECDM}(\mathbf{W}; \mathbf{X})_{:,j}|}{\sum_{i=1}^{I} |\text{ECDM}(\mathbf{W}; \mathbf{X})_{i,j}|},$$

[1]In practice, using nECDM makes the hyperparameter γ to be less sensitive on I, since $\text{nECDM}(\mathbf{W}; \mathbf{X})_i$ represents *relative* contributions across the input channels.

Algorithm 1 Channel de-allocation (`dealloc`)

Input: $\mathbf{S} = (\mathbf{W}, \mathbf{g}, \boldsymbol{\pi}, \mathbf{b})$, $\text{nECDM}(\mathbf{W}; \mathbf{X}) \in \mathbb{R}^{I \times O}$, damage level $\gamma > 0$

```
Initialize $C, C' \leftarrow \emptyset, \emptyset$
repeat
    $C \leftarrow C'$
    $C' \leftarrow C \cup \{\text{argmin}_{i \notin C} \|\text{nECDM}(\mathbf{W}; \mathbf{X})_i\|_\infty\}$
until $\|\sum_{i \in C'} \text{nECDM}(\mathbf{W}; \mathbf{X})_i\|_\infty \leq \gamma$
for all $i \in C$ do
    $g_i \leftarrow 0$
end for
```

Algorithm 2 Channel re-allocation (`realloc`)

Input: $\mathbf{S} = (\mathbf{W}, \mathbf{g}, \boldsymbol{\pi}, \mathbf{b})$, $\text{nECDM}(\mathbf{W}; \mathbf{X}) \in \mathbb{R}^{I \times O}$, candidate size K, maximum re-allocation size $N_{\texttt{max}}$

```
Initialize $C \leftarrow \emptyset$
for $i = 1$ to $I$ do
    $s_i \leftarrow \|\text{nECDM}(\mathbf{W}; \mathbf{X})_i\|_2$
    $N \leftarrow |\{j \in \{1, \ldots, I\} : \pi_j = \pi_i\}|$
    if $N > N_{\texttt{max}}$ then
        $s_i \leftarrow 0$
    end if
end for
$C \leftarrow$ Select top-$K$ indices from s
for $i = 1$ to $I$ do
    if $g_i = 0$ then
        $c \leftarrow$ Select an element from $C$ randomly
        $\pi_i, g_i, \mathbf{W}_{i,:,:} \leftarrow \pi_c, 1, \mathbf{0}$
        Re-initialize $b_i$ randomly
    end if
end for
```

for $j = 1, \ldots, O$. Once nECDM is computed, channels to be de-allocated are determined by the channel of minimum $||\text{nECDM}(\mathbf{W}; \mathbf{X})_i||_\infty$ iteratively, while the ℓ^∞-norm of their vector sum of $\text{nECDM}(\mathbf{W}; \mathbf{X})_i$ is less than γ.

In case of `realloc`, on the other hand, we select top-K largest channels with respect to the ℓ^2-norm of nECDM, i.e. $||\text{nECDM}(\mathbf{W}; \mathbf{X})_i||_2$.[2] The selected top-$K$ channels randomly occupy the channels that are currently de-allocated (i.e., $g_i = 0$). When i-th channel is re-allocated, $\mathbf{W}_{i,:,:}$ are set to zero so that the operation does not harm the training. We also set a maximum reallocation size $N_{\texttt{max}}$ to prevent a feature to be re-allocated too redundantly. Algorithm 1 and 2 summarize the overall procedure of `dealloc` and `realloc`, respectively.

Finally, the training scheme of $\mathbf{S}$ is build upon any exist-

[2]We use ℓ^2-norm for `realloc` to consider features that contribute across many filters more importantly. Nevertheless, we found using ℓ^∞-norm instead also achieves a comparable result.

ing SGD training method, simply by calling `dealloc` or `realloc` additionally on demand. In other words, at any time during training $\mathbf{W}$ via SGD, `dealloc` and `realloc` additionally updates the remaining parameters of $\mathbf{S}$: `dealloc` for $\mathbf{g}$, and `realloc` for $\mathbf{g}$, $\boldsymbol{\pi}$, $\mathbf{b}$ and $\mathbf{W}$.

3. Experiments

We evaluate our method on various image classification tasks: CIFAR-10/100 (Krizhevsky, 2009), Fashion-MNIST (Xiao et al., 2017), Tiny-ImageNet[3], and ImageNet (Russakovsky et al., 2015) datasets. We consider a variety of CNN architectures recently proposed, including ResNet (He et al., 2016a), DenseNet (Huang et al., 2017), and ResNeXt (Xie et al., 2017). Unless otherwise stated, we fix $\gamma = 0.001$, $K = 3$, and $N_{\texttt{max}} = 32$ for training selective convolutional layers. In cases of DenseNet-40 and ResNet-164, we do not use $N_{\texttt{max}}$, i.e. $N_{\texttt{max}} = \infty$, as they handle relatively fewer channels. We did not put much effort for the very best hyperparameters of our method, so there can be better ones depending across datasets. Nevertheless, we found our method has resilience on the given configuration, as we verify from the experiments: it generally yield good performances for all the tested models and datasets, even in the large-scale ImageNet experiments. The more training details, e.g. datasets and model configurations, are given in the supplementary material.

In overall, our results show that training with channel-selectivity consistently improves the model efficiency, mainly demonstrated in two aspects: (a) improved accuracy and (b) model compression. We also perform an ablation study to verify the effectiveness of our main ideas.

3.1. Improved Accuracy with Selective Convolution

We compare classification performance of various CNN models trained with our method against conventional training. For each baseline model, we consider the counterpart *selective* model that every convolutional layer is replaced by the corresponding selective convolutional layer. We train the pair of models for the same number of epochs.

Table 1 and Table 3 summarize the main results. In overall, our method consistently reduces classification error rates across all the tested models compared to the conventional training.[4] As the selective models have almost the same number of parameters with the baseline model, the results confirm that the proposed channel de/re-allocation scheme utilized the given parameters more efficiently, i.e. by enlarging the kernel size of each important channel selectively.

Recall that our training method is compatible on any existing training scheme, since `dealloc` and `realloc` does not affect the loss during training due to their function-preserving property. The training configurations used in our experiments, e.g. weight decay or momentum, are one of the most common choice for training CNNs. Even though not explored in this paper, we believe that the effectiveness of our method can be further improved by using more coarse-grained training schemes, e.g. channel-level regularization (Wen et al., 2016; Liu et al., 2017; Neklyudov et al., 2017), as our method operates in the channel-level as well.

3.2. Model Compression with Selective Convolution

Next, we show that our method can also be used for model compression, when the model is under regime of large-scale features. To this end, we consider two state-of-the-art CNN models, namely DenseNet-BC-190 (Huang et al., 2017) and CondenseNet-182 (Huang et al., 2018b). Here, CondenseNet is a highly-efficient mobile-targeted CNN architecture, outperforming MobileNet (Howard et al., 2017) and ShuffleNet (Zhang et al., 2018). We select these two architectures to compare since they both attempt to design an efficient architecture under extremely large number of features. Here, we apply our ECDM-based channel de-allocation scheme upon these architectures to show that our method can further exploit the inefficiency of large-scale feature regime to improve the model efficiency. Namely, we compare the model efficiency of the models with our counterpart models with selective convolution trained using only `dealloc`, with the intention of maximizing the computational efficiency.

In the case of CondenseNet, the architecture contains a channel pruning mechanism during training, namely *learned group convolution* (LGC) layer, which is similar to `dealloc` in our method. We aim to compare this mechanism with selective convolution trained using only `dealloc`, showing that our de-allocation mechanism with ECDM can further improve the efficiency.[5] To this end, we consider a variant of CondenseNet-182 where only each of the LGC layers inside is replaced by the selective convolutional layer, coined *CondenseNet-SConv-182*. We remark that, unlike LGC, we use neither group convolution nor group-lasso regularization for selective convolutional layers, even if they can further improve the efficiency. The other training details are set identical to the original one by Huang et al. (2018b) for fair comparison.

Table 2 report the result. First, observe that CondenseNet-SConv-182 shows much better efficiency compared to the original CondenseNet-182. Namely, our model achieves

[3]`https://tiny-imagenet.herokuapp.com/`

[4]We remark that the reduction in the ImageNet results (Table 3) is quite non-trivial, e.g. reducing error $23.6\% \rightarrow 23.0\%$ requires to add 51 more layers from ResNet-101 (i.e., ResNet-152), according to the official repository: `https://github.com/KaimingHe/deep-residual-networks`.

[5]For the interested readers, we present the more details of LGC in the supplementary material.

Table 1. Comparison of test error rates on various classification tasks. "SelectConv" indicates our model from the corresponding baseline that is trained with channel-selectivity. We indicate k by the growth rate of DenseNet. All the reported values and error bars are measured by computing mean and standard deviation across 3 trials upon randomly chosen seeds, respectively.

			Error rates (%)			
Model	Params	Method	CIFAR-10	CIFAR-100	Fashion-MNIST	Tiny-ImageNet
DenseNet-40 (bottleneck, $k = 12$)	0.21M	Baseline	6.62±0.15	29.9±0.1	5.03±0.07	45.8±0.2
		SelectConv	**6.09±0.10 (-8.01%)**	**28.8±0.1 (-3.42%)**	**4.73±0.06 (-5.96%)**	**44.4±0.2 (-3.03%)**
DenseNet-100 (bottleneck, $k = 12$)	1.00M	Baseline	4.51±0.04	22.8±0.3	4.70±0.06	41.0±0.1
		SelectConv	**4.29±0.08 (-4.88%)**	**22.2±0.1 (-2.64%)**	**4.58±0.05 (-2.55%)**	**39.9±0.3 (-2.78%)**
ResNet-164 (bottleneck, pre-act)	1.66M	Baseline	4.23±0.15	21.3±0.2	4.53±0.04	37.7±0.4
		SelectConv	**3.92±0.14 (-7.33%)**	**20.9±0.2 (-1.97%)**	**4.37±0.03 (-3.53%)**	**37.5±0.2 (-0.56%)**
ResNeXt-29 ($8 \times 64d$)	33.8M	Baseline	3.62±0.12	18.1±0.1	4.40±0.07	31.7±0.3
		SelectConv	**3.39±0.14 (-6.36%)**	**17.6±0.1 (-2.92%)**	**4.27±0.06 (-2.95%)**	**31.4±0.3 (-0.88%)**

Table 2. Comparison of performance on CIFAR-10 between different CNN models including ours. Models named "X-Pruned" are the results from Network slimming (Liu et al., 2017).

Model	Params	FLOPs	Error rates (%)
ResNet-1001 (He et al., 2016b)	16.1M	2,357M	4.62
WideResNet-28-10 (Zagoruyko & Komodakis, 2016)	36.5M	5,248M	4.17
ResNeXt-29 ($16 \times 64d$) (Xie et al., 2017)	68.1M	10,704M	3.58
VGGNet-Pruned (Liu et al., 2017)	2.30M	391M	6.20
ResNet-164-Pruned (Liu et al., 2017)	1.10M	275M	5.27
DenseNet-40-Pruned (Liu et al., 2017)	0.35M	381M	5.19
DenseNet-BC-190 (Huang et al., 2017)	25.6M	9,388M	3.46
DenseNet-BC-SConv-190 (Ours)	**11.5M (-55.1%)**	**4,287M (-54.3%)**	**3.45 (-0.29%)**
CondenseNet-182 (Huang et al., 2018b)	4.20M	513M	3.76
CondenseNet-SConv-182 (Ours)	**3.24M (-22.9%)**	**422M (-17.7%)**	**3.50 (-6.91%)**

Table 3. Comparison of the single-crop top-1 validation error rates on the ImageNet dataset. "SelectConv" indicates our model from the corresponding baseline that is trained with channel-selectivity. In case of DenseNet-121, γ is adjusted to 0.0005.

Model	Params	Method	Error (%)
DenseNet-121 ($k = 32$)	7.98M	Baseline	24.7
		SelectConv	**24.4**
ResNet-50 (bottleneck)	22.8M	Baseline	23.9
		SelectConv	**23.4**

-22.9% of reduction in parameters, and -6.91% in error rates over the baseline, even the orignal CondenseNet-182 also performs channel pruning on $\frac{3}{4}$ of the input channels. This result confirms the effectiveness of our ECDM-based deallocation scheme over the ℓ^1-based LGC, suggesting a benefit of using our method for both accuracy improvement and model compression. DenseNet-BC-SConv-190, on the other hand, shows more reduction in parameters and FLOPs without loss of accuracy, compared to the original model. This reduction is due to that DenseNet-BC-190 is less optimized in its architecture for efficiency, compared to CondenseNet-182. This shows that our method can discover an inherited inefficiency inside the model as well. We also compare the result with various state-of-the-art CNN models. Remarkably, CondenseNet-SConv-182 achieves even better accuracy than ResNeXt-29, while ours has 25× fewer FLOPs. Compared to *network slimming* (Liu et al., 2017), on the other hand, this model shows significantly better accuracy with similar FLOPs.

3.3. Ablation Study

We also conduct an ablation study on the proposed method, investigating the detailed analysis on our method. Throughout this study, we consider DenseNet-40, which consists 3 *dense blocks*, each of which consists of 6 consecutive *dense units*. Each of the units produces $k = 12$ features, and those features are concatenated over the units. Unlike Huang et al. (2017), we do not place a feature compression layer between the dense blocks for simplicity. All the experiments in this section are performed on CIFAR-10.

Analysis on the selected channels. We train a DenseNet-40 model with channel selectivity, and analyze which channels are de/re-allocated during training. Figure 3 demonstrates channel-indices that are de/re-allocated for each dense unit. The result show that features made at early units (i.e. lower-level features) are de-allocated more than the others, which is consistent with our intuition. Remarkably, one can also find there are some channels which tend to be de-allocated across multiple *consecutive* units, possi-

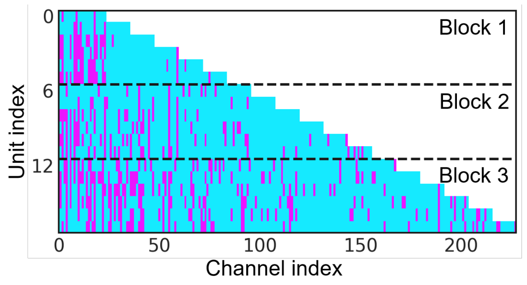


Figure 3. Illustration of channel indices that a channel de/re-allocation is occurred in a DenseNet-40 model for each of dense units. The channels of interest are marked by magenta. Unit indices are divided into three for each dense block.

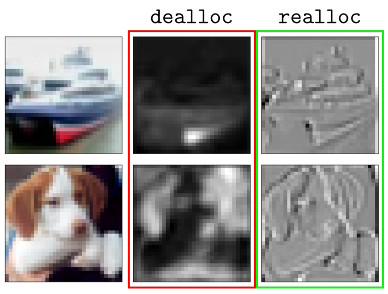


Figure 4. Images from CIFAR-10 and their feature maps at a de-allocated channel index (middle), and at the corresponding re-allocated index (right). The feature maps are taken from the first dense unit of a DenseNet-40 model.

bly across multiple blocks, but apparently used in later units. This tendency found by selective convolution reflects how DenseNet processes features under its architectural benefit: some low-level features may not needed for a long term in the visual pathway. Our method effectively utilizes the redundancy from just "keeping" such of the features.

Figure 4 provides an additional insight from which channel is actually de/re-allocated in the model. We observed that channels containing more information for the given task, e.g. sharp edge information for classification, tend to be re-allocated more, possibly for better processing of the task information. We provide more illustrations about which channels are de/re-allocated in the supplementary material.

Channel re-allocation. Our main motivation to introduce spatial shifting is to force the `realloc` procedure to utilize the re-allocated parameters diversely in input distributions. To evaluate its effect in accuracy, we compare five DenseNet-40 models with different re-allocation scheme:

- *Ours* (+D+R$^{\texttt{zero+shift}}$): If a channel is re-allocated, the corresponding convolutional weights are set to 0, and spatial shifting is imposed correspondingly.
- *Zero re-initialization* (+D+R$^{\texttt{zero}}$): We do not use spatial shifting from the above original configuration, i.e., `realloc` is used, but spatial shifting is not imposed.

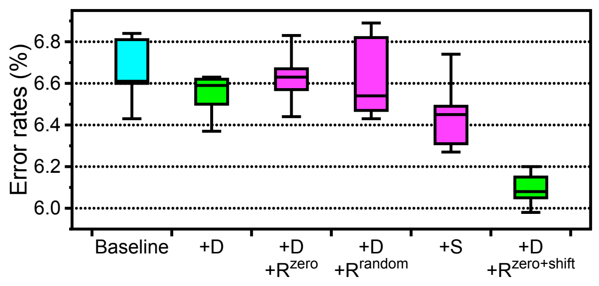


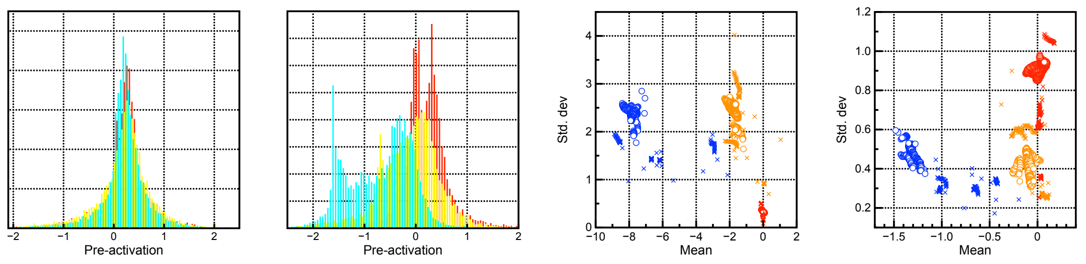
Figure 5. Comparison of error rates on DenseNet-40 models between our model of using spatial shifting (+D+R$^{\texttt{zero+shift}}$), and its ablations (+D+R$^{\texttt{zero}}$ and +D+R$^{\texttt{random}}$). Error rates of the baseline model without channel-selectivity is also provided.

- *Random re-initialization* (+D+R$^{\texttt{random}}$): Now, we modify the initialization, i.e., `realloc` is used without spatial shifting, and the weights are re-initialized following the model initialization scheme.
- *De-allocation only* (+D): Only `dealloc` is used, i.e. `realloc` is not performed during training.
- *Shift only* (+S): Neither de/re-allocation is used, but all channels learn spatial bias from the beginning. This ablation is essentially equivalent to the method proposed by Jeon & Kim (2017).

Figure 5 clearly shows that +D+R$^{\texttt{zero+shift}}$ outperforms the others, while +D+R$^{\texttt{zero}}$ or +D+R$^{\texttt{random}}$ could not statistically improve its accuracy over the baseline and +D even though `realloc` is performed. This confirms that copying a channel naïvely is not enough, and the spatial shifting is an effective trick under our channel de/re-allocation setting. In case of +S, on the other hand, we found a certain gain from the use of shifting biases, but +D+R$^{\texttt{zero+shift}}$ also outperforms it by a large margin. We also emphasize that +D+R$^{\texttt{zero+shift}}$ uses much less shifting, e.g. about 5 times less than +S, as it performs shifting only for the re-allocated channels. This confirms that our de/re-allocation scheme is crucial for the effectiveness.

Figure 7 further compares the models with the testing loss curves. Each of the curves is taken from the model that showed median performance across the trials. One can clearly observe that +D+R$^{\texttt{zero+shift}}$ is converged at much lower testing loss, while the others are stuck at a similar local minima. Recent works show that all sub-optimal local minima in a neural network can be eliminated theoretically by adding a neuron of a certain form (Liang et al., 2018; Kawaguchi & Kaelbling, 2019). In this sense, our training scheme can be thought as a process of continuously adding new neurons into the network during training, along with the spirit of network pruning.

Learned biases from spatial shifting. As explained in Section 2.2, channel re-allocation with spatial shifting has an



(a) Non-boundary pixels (b) Boundary pixels (c) Initialized randomly (d) Trained to converge

Figure 6. Pixel-wise input distributions of the 4th layer (i.e. the middle of the first dense block) in a DenseNet-40 model. (a, b) Empirical distributions of three randomly chosen pixels in a fixed channel of input, which are inferred from CIFAR-10 test dataset. (c, d) Scatter plots between empirical mean and standard deviation of each pixel distributions, plotted for 3 representative channels in the input. Each plot consists 1,024 points, as a channel have 32 × 32 pixels. Pixels in boundaries are specially marked as ×.

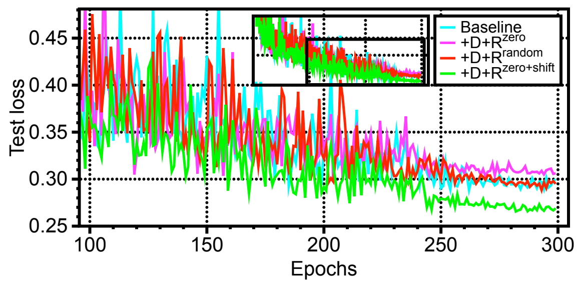


Figure 7. Comparison of test losses of our model (+D+R$^{\texttt{zero+shift}}$) and its ablations on spatial shifting (+D+R$^{\texttt{zero}}$ and +D+R$^{\texttt{random}}$). Each curve denotes cross-entropy loss measured on the CIFAR-10 test set for each training epoch.

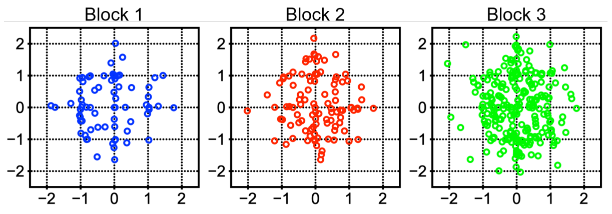


Figure 8. Scatter plots of learned spatial biases in pixels for the re-allocated channels in a DenseNet-40 model. The points are grouped into different plots for each dense block.

effect of enlarging the kernel sizes for the selected channels. Figure 8 illustrates how the spatial biases are actually trained for each dense block in a DenseNet-40 model with channel-selectivity. Interestingly, we found that some biases are converged with a tendency to *align* on the pixel grids, especially at the first dense block. Since the information contained in CIFAR-10 images is given in pixel-wise, biases on exact grids will maximize the amount of new information from the original channel. The observation suggests that optimizing shifting-parameters $\mathbf{b}$ indeed considers such an effect during training. Biases at the later layers, on the other hand, shows less tendency of aligning but a larger diversity on the values, which corresponds to larger kernel sizes.

Empirical support on the ECDM formula. In order to obtain the practical formula for ECDM (2), we assumed that the input $\mathbf{X}$ is of the from $\mathrm{ReLU}(\mathrm{BN}(\mathbf{Y};\boldsymbol{\beta},\boldsymbol{\gamma}))$ for another random variable $\mathbf{Y}$, and approximated $\mathbf{X}_{i,h,w}$ by $\max(\mathcal{N}(\beta_i,\gamma_i^2),0)$ for all i,h,w. Essentially, this approximation imposes two key assumptions on $\mathbf{Y}$ accordingly: (a) $\mathbf{Y}$ follows normal distribution, and (b) for a fixed i, each of $\mathbf{Y}_{i,:,:}$ are *identically* distributed. Here, our question is that how much these assumptions hold in modern CNNs.

To validate this, we calculate hidden inputs $\mathbf{X}^{\texttt{test}}$ at the 4th dense unit of a DenseNet-40 model using CIFAR-10 test images. By analyzing empirical distributions of $\mathbf{X}^{\texttt{test}}_{c,h,w}$ for varying h and w, we found that: (a) for a fixed c, most of the distributions are uni-modal, with exceptions at the boundary pixels (Figure 6(a), 6(b)), and (b) for a large portion of c the means and variances of $\mathbf{X}^{\texttt{test}}_{c,h,w}$'s are concentrated in a cluster (Figure 6(d)). These observations support that the proposed assumptions are reasonable, with some exceptional points, e.g. the boundary pixels. We also found that the trends still exist even the model re-initialized (Figure 6(c)), i.e. they are not "learned", but come from some structural properties of CNN. Two of such properties can be responsible: (a) the central limit theorem from the linear, weighted summing nature of convolution, and (b) equivariance of convolution on spatial dimensions. This observation confirms that ECDM is valid at anytime during training.

4. Conclusion

We address a new fundamental problem of training CNNs given restricted neural resources, where our new approach is exploring pruning and re-wiring on demand via channel-selectivity. Such a dynamic training scheme have been considered difficult in a conventional belief, as it easily makes the training unstable. We overcome this with our novel metric, ECDM, which allows more robust pruning during training, consequently opens a new direction of training CNNs. We expect that the channel-selectivity is also a desirable property for many subjects related to CNNs, e.g. interpretability (Selvaraju et al., 2017), and robustness (Goodfellow et al., 2015), just to name a few.

Acknowledgements

This research was supported by Naver Labs and the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921).

References

Bolukbasi, T., Wang, J., Dekel, O., and Saligrama, V. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning (ICML)*, pp. 527–536, 2017.

Chen, C., Tung, F., Vedula, N., and Mori, G. Constraint-aware deep neural network compression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 400–415, 2018.

Chen, Y., Li, J., Xiao, H., Jin, X., Yan, S., and Feng, J. Dual path networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 4470–4478, 2017.

Courbariaux, M. and Bengio, Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

Dai, B., Zhu, C., Guo, B., and Wipf, D. Compressing neural networks using the variational information bottleneck. In *International Conference on Machine Learning (ICML)*, pp. 1135–1144, 2018.

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., and Wei, Y. Deformable convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 764–773, 2017.

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., and Salakhutdinov, R. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1039–1048, 2017.

Goodfellow, I., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015. URL `http://arxiv.org/abs/1412.6572`.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.-J., and Choi, E. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1379–1387, 2016.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

Han, S., Pool, J., Narang, S., Mao, H., Tang, S., Elsen, E., Catanzaro, B., Tran, J., and Dally, W. J. DSD: regularizing deep neural networks with dense-sparse-dense training flow. *CoRR*, abs/1607.04381, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. IEEE Computer Society, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*, volume 9908 of *Lecture Notes in Computer Science*, pp. 630–645. Springer, 2016b.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1389–1397, 2017.

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

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269. IEEE Computer Society, 2017.

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., and Weinberger, K. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*, 2018a. URL `https://openreview.net/forum?id=Hk2aImxAb`.

Huang, G., Liu, S., Van der Maaten, L., and Weinberger, K. Q. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2752–2761, 2018b.

Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*, abs/1602.07360, 2016.

Ioannou, Y., Robertson, D., Cipolla, R., and Criminisi, A. Deep roots: Improving cnn efficiency with hierarchical filter groups. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1231–1240, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR, 2015.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2014.

Jeon, Y. and Kim, J. Active convolution: Learning the shape of convolution for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4201–4209, 2017.

Jeon, Y. and Kim, J. Constructing fast network through deconstruction of convolution. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, pp. 5955–5965. Curran Associates, Inc., 2018.

Kawaguchi, K. and Kaelbling, L. P. Elimination of all bad local minima in deep learning. *arXiv preprint arXiv:1901.00279*, 2019.

Kellerer, H., Pferschy, U., and Pisinger, D. *Knapsack Problems*. Springer, Berlin, Germany, 2004.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016.

Liang, S., Sun, R., Lee, J. D., and Srikant, R. Adding one neuron can eliminate all bad local minima. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 4355–4365. Curran Associates, Inc., 2018.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 2755–2763. IEEE, 2017.

Louizos, C., Ullrich, K., and Welling, M. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3290–3300, 2017.

Louizos, C., Welling, M., and Kingma, D. P. Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 7827–7838, 2018.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning (ICML)*, pp. 2498–2507, 2017.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, pp. 807–814. Omnipress, 2010.

Neklyudov, K., Molchanov, D., Ashukha, A., and Vetrov, D. P. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 6778–6787, 2017.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 442–450, 2015.

Raidl, G. R. and Gottlieb, J. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4):441–475, 2005.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, pp. 525–542. Springer, 2016.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. URL http://arxiv.org/abs/1802.01548.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Sahay, A., Scobie, K. N., Hill, A. S., O'carroll, C. M., Kheirbek, M. A., Burghardt, N. S., Fenton, A. A., Dranovsky, A., and Hen, R. Increasing adult hippocampal neurogenesis is sufficient to improve pattern separation. *Nature*, 472(7344):466, 2011a.

Sahay, A., Wilson, D. A., and Hen, R. Pattern separation: a common function for new neurons in hippocampus and olfactory bulb. *Neuron*, 70(4):582–588, 2011b.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4510–4520, 2018.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D., et al. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626, 2017.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

Sun, K., Li, M., Liu, D., and Wang, J. IGCV3: Interleaved low-rank group convolutions for efficient deep neural networks. *CoRR*, abs/1806.00178, 2018. URL http://arxiv.org/abs/1806.00178.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9. IEEE Computer Society, 2015.

Vasquez, M. and Vimont, Y. Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81, 2005.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2074–2082, 2016.

Wu, B., Wan, A., Yue, X., Jin, P., Zhao, S., Golmant, N., Gholaminejad, A., Gonzalez, J., and Keutzer, K. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9127–9135, 2018.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995. IEEE Computer Society, 2017.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2016.

Zhang, T., Qi, G.-J., Xiao, B., and Wang, J. Interleaved group convolutions. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 4373–4382, 2017.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8697–8710, 2018.