MuffNet: Multi-Layer Feature Federation for Mobile Deep Learning

Hesen Chen
Alibaba Group
Beijing, China
hesen.chs@alibaba-inc.com

Ming Lin
Alibaba Group
Bellevue, USA
ming.l@alibaba-inc.com

Xiuyu Sun
Alibaba Group
Beijing, China
xiuyu.sxy@alibaba-inc.com

Qian Qi
Alibaba Group
Bellevue, USA
qi.qian@alibaba-inc.com

Hao Li
Alibaba Group
Beijing, China
lihao.lh@alibaba-inc.com

Rong Jin
Alibaba Group
Bellevue, USA
jinrong.jr@alibaba-inc.com

Abstract

The increasing industrial demands to deploy deep neural networks on resource constrained mobile device motivates recent researches of efficient structure for deep learning. One popular approach is to densify network connectivity by sharing feature maps between layers. A side effect of this approach is that the volume of the feature maps and the convolution computation will exponentially blow up. In this work, we propose a novel structure, named Multi-Layer Feature Federation Network (MuffNet), to address this issue. The MuffNet is a densely connected network but consumes much less memory and computation at inference. The key idea of the MuffNet is to elaborately split the feature maps of one layer to different groups. Each feature map group is then shared only once with the other layer. In this way we maintain the network computation within budget while keeping the topology density of the network. On the theoretical side, we show that under the same computational budget, MuffNet is a better universal approximator for functions containing high frequency components. We validate the superiority of MuffNet on popular image classification and object detection benchmark datasets. The extensive experiments show that MuffNet is more efficient especially for small models under 45 MFLOPs.

1. Introduction

Deep convolutional neural networks have achieved great success in various challenging tasks including image classification [10, 52], object detection [46, 9], speech recognition [7] and video analysis [40]. Blessed by the internet scale data, it is possible to train very deep networks with terabytes of training data on highly optimized hardware such as Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU). Powered by modern GPU clusters, training a 50 layer Residual Network [10] can be done in a few minutes [19, 6, 63].

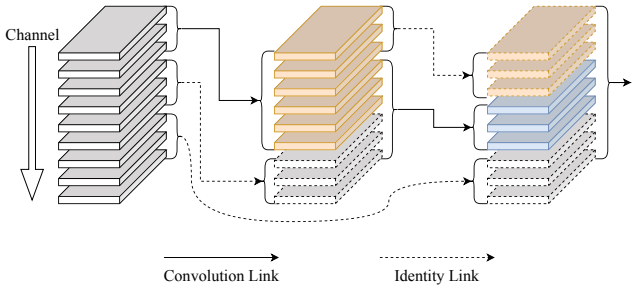


Figure 1. MuffNet structure. Layers are pair-wise densely connected. Channels are split into groups and each channel group is connected to one other layer via convolution or identity mapping.

Although larger and deeper models keep refreshing the state-of-the-art performance, they require huge memory space to store the model parameters and powerful hardware to do inference. For example, the VGGNet (vgg-vd-19) [52] has over 528 million parameters. When the input image size is 224×224, its takes 20 GFLOPs (Floating Point Operations) to do inference. The model size of ResNet-50 [10] is 98 MB and its inference cost is 4 GFLOPs per image. The DenseNet-121 [16] achieves better accuracy at the cost of 3 GFLOPs. It is difficult to deploy these popular deep models on resource constrained mobile device such as smartphones or in the scenario where low latency is enforced such as auto-pilot. The expense of hosting deep neural network is another non-negligible cost to consider in a large-scale industrial deployment. A moderate modern GPU could cost a few hundred dollars with power consumption over 150 Watt. Therefore, it is urgent to develop efficient deep neural network models for mobile device with compact model size

and less computation.

Recent researches of developing efficient deep neural networks could be categorized into five directions, namely, weight pruning [27, 25], low-rank approximation [48, 41], quantization [23, 66, 58], efficient structure [13, 65] and network architecture search [45, 51]. In this work, we concern about the structural efficiency of densely connected network. It was observed that reusing feature maps from low level layers will improve the network accuracy [16]. However, dense network topology will result in computational explosion. This could be alleviated by sparsifying the network topology [67, 14] but the model accuracy after sparsification is usually degraded at the same time. In this work, we try to keep the dense topology of the network while maintaining the computational cost within a small budget. To this end, we innovate a network structure named *MUlti-layer Feature Federation* Network (MuffNet). In the MuffNet, the output channels of each convolutional layer are split into non-overlapped groups. The input channels are concatenated from channel groups of multiple previous layers. Each output channel is served as the input of a higher layer only once. The topology is illustrated in Figure 5. In this way, we guarantee that the computational cost will not blow up by the dense connection.

Comparing to previous works that sparsify network links between layers, the MuffNet does not compromise the link density for computational efficiency. We argue that our solution has theoretical advantages. We study the approximation ability of network in the frequency domain. It is proved in [39] that a deep convolutional network behaves like a low-pass filter. The MuffNet ensembles convolutional sub-networks of all depths such that both low and high frequencies could pass through the structure. This makes the MuffNet a better universal functional approximator under the given computational budget. From this viewpoint, the link-sparsification approach does not improve the network spectrum in proportion since it only ensembles a few sub-networks of the selected depths.

Based on the above observation, we conduct extensive numerical experiments to find the optimal configurations of the MuffNet at 40, 140, and 300 MFLOPs. We find that the optimal feature map sharing pattern of the MuffNet exhibits three stages. In the low-level stage, it is better to share feature maps uniformly. In the mid-level stage, an exponentially decayed sharing ratio is preferred. In high-level stage, no feature map sharing is the best choice. Please check Section 3 for more details.

The remainder of this work is organized as following. In Section 2, we review related works of recent development in efficient deep learning. Section 3 introduces the proposed multi-layer feature federation structure. Theoretical analysis of the MuffNet is presented in Section 4. We describe our experiment settings and report numerical results in Section 5. Section 6 encloses this work.

2. Related Work

In this section, we briefly review recent advances in efficient deep neural network researches.

The research of neural network acceleration dates back to 1989 when LeCun *et al.* proposed to prune unimportant neurons according to the Hessian matrix [62]. Since then various sparsification methods are proposed to prune the weight tensors of the convolution kernel [27, 25]. However, the sparse convolution is difficult to be accelerated on modern hardware [33] which limits its practical value. A more efficient strategy is to enforce the structured sparsity in the network [15, 21, 24, 38, 59, 2, 4, 12, 29, 30, 68]. Figurnov *et al.* [37] show that the spatial redundancy of the network is also considerable and we do not need to do convolutions pixel-by-pixel. Ren *et al.* [36] further discover that block-wise sparse convolution is a more efficient way to reduce the spatial redundancy. Another way to reduce the computation is to approximate the weight tensors by their low-rank decompositions [48, 34, 41, 61, 53]. Vadim Lebedev *et al.* [56] propose to use the CP-decomposition followed by fine-tuning to compensate the accuracy loss caused by low-rank approximation.

Lots of recent researches develop data-dependent methods to find novel efficient network structure. The Network Architecture Search (NAS) searches network structures with high accuracy [50, 3, 18, 45, 51, 42]. Some NAS systems consider the resource constraints and the network accuracy simultaneously during the search [8, 60, 57, 55, 44]. Courbariaux *et al.* [35] compress networks to binary weights. It is later discovered that the ternary weight network is more accurate while nearly as efficient as binary ones [23, 66, 58]. The LQ-Net [64] uses learned dictionary to quantize networks for better accuracy. An ADMM solver is developed in [22] for general quantized network optimization.

In addition, the efficiency of the network could be improved by more efficient network structure. A popular approach is to replace full convolution with sequence of depth separable convolution, point-wise convolution and/or group convolution. In SqueezeNet [17] the dense convolution is replaced by the cost-efficient 1×1 and 3×3 filters to compress the model with fewer parameters. MobileNet [13] directly adopts depth separable convolutions instead of 3×3 convolutions. MobileNetV2 [49] proposes inverted residuals to capture more spatial information and linear bottlenecks to maintain the representational power. ShuffleNet [65, 65] shuffles channels after the depth separable convolution so that the information will propagate through channels. IGCV [53] eliminates the redundancy in convolution kernels by interleaved group convolutions. X-Net [43] randomly connects channels to form sparse convolution. The

innovation of DenseNet shows that a more accurate model is achievable by connecting layers densely. To alleviate the computational cost of dense links, SparseNet [67] and Log-DenseNet [14] propose to connect layers in a sparse pattern. The CondenseNet [5] uses learned group convolutions on dense links to reduce the convolution computation. Comparing to SparseNet and Log-DenseNet, the MuffNet has a dense topology. The data-dependent structure search methods could be combined with MuffNet for better accuracy. Therefore, this work is complementary to those data-dependent methods.

Computational Metric The FLOP is widely adopted in previous works to measure the computational cost of convolutional neural network. Some works find that smaller FLOP does not imply faster inference [60, 1] since the hardware implementation various from device to device. In this work we suggest to use FLOP due to three reasons: 1) In most cases FLOP is positively related to the final performance on hardware; 2) The hardware-dependent metrics cannot be fairly compared across different types of devices; 3) Testing network performance on hardware usually requires engineering efforts not affordable to everyone. Therefore, using FLOP is a more meaningful way to compare performance among different methods on different devices.

3. Multi-Layer Feature Federation

In this section, we first introduce the feedforward inference process of the MuffNet. Then we describe the convolutional blocks we used in each layer. Finally, we give the complete network implementation.

3.1. Feedforward Inference

Considering a MuffNet of L layers of feature maps. Each feature map layer except the first layer is obtained by a convolutional block such as a combination of group convolution and channel shuffling. We will describe the details of the convolutional block in the next subsection. The MuffNet splits the output channels of each layer into non-overlapped groups. The output channel groups from different layers are then re-grouped as the input of the consecutive layers.

For sake of clarity, it is better to introduce a matrix M to parameterize the splitting and re-grouping scheme. For an L-layer MuffNet, M is an $L \times L$ up-triangular matrix. The first layer is the input feature maps of the network and the last layer is the output feature maps of the network. The input layer has $\sum_{j=1}^{L} M_{1,j}$ channels and the output layer has $\sum_{i=1}^{L} M_{i,L}$ channels. Without causing confusion we index the output channel groups by their group size $M_{i,j}$. For any $j > i$, the channel group $M_{i,j}$ is served as part of the input feature maps of the convolutional block generating the j-th feature map layer. Again, without causing confusion we index the convolutional blocks by their output layers. The diagonal element $M_{i,i}$ is the number of channels generated by the i-th convolutional block and is fed into the $(i+1)$-th convolutional block. For the i-th convolutional block, its input feature maps are aggregated from $\{M_{1,i-1}, M_{2,i-1}, \cdots, M_{i-1,i-1}\}$ groups and its output channels are split into $L-i+1$ groups of size $\{M_{i,i}, M_{i,i+1}, \cdots, M_{i,L}\}$.

Figure 2 demonstrates the step-by-step feedforward inference of a MuffNet parameterized by an M matrix. The network has four feature map layers generated by three convolutional blocks. The first layer is the input feature map layer with $M_{1,1}$ channels. In Figure 2(a), the convolutional block ConvUnit2 is activated to convolute $M_{1,1}$ and generates $M_{2,2:4}$. Please note that $M_{2,1} = 0$ since M is up-triangular. In Figure 2(b), the $M_{2,2:4}$ are assigned to the corresponding layers for later re-grouping. In Figure 2(c), ConvUnit3 is activated. The input of ConvUnit3 is $M_{2,2}$. The convolution output is split into two channel groups of size $M_{3,3}$ and $M_{3,4}$. In Figure 2(d), similarly $M_{3,4}$ is moved to the fourth layer. ConvUnit4 is then activated. We re-group $M_{2,3}$ and $M_{3,3}$ as input of ConvUnit4. The output of ConvUnit4 is $M_{4,4}$. Finally, we group $\{M_{2,4}, M_{3,4}, M_{4,4}\}$ as the output of the network.

3.2. Convolutional Block

The implementation of the convolutional block in MuffNet is flexible. Inspired by the previous researches [17, 13], we adopt the implementation described in Figure 3 for a good balance between computational cost and model accuracy.

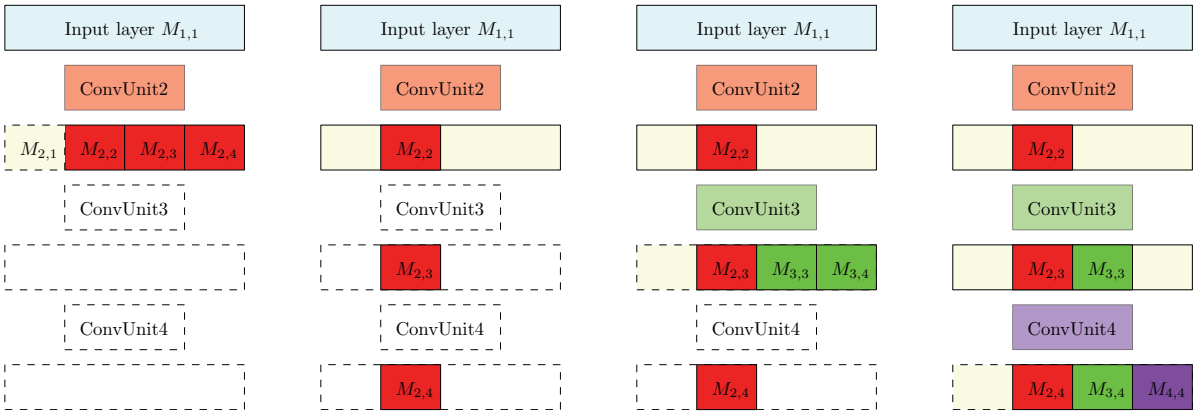
The MuffNet uses three types of convolutional blocks. When the input and output layer are of the same shape, we use normal block in Figure 3(a) or normal block with residual link (residual block) in Figure 3(b). If the input and output layer have different height and width, we use the reduction block in Figure 3(c). Following the convention, we always downsample the feature map size by factor 2.

Structure FLOPs Given $M_{i,j}$, we could derive its FLOPs as following. Suppose the feature map size of the input layer is $s_i \times s_i$ and that of the output layer is $s_j \times s_j$. Define m_i as the number of input channels and m_j as the number of output channels. For the normal block and the residual block,

$$\text{FLOP}_i = s_i^2(m_i^2 + 9m_i + m_i m_j) \ .$$

For the reduction block,

$$\text{FLOP}_i = s_i^2(\frac{1}{2}m_i^2 + \frac{1}{2}m_i m_j) + s_j^2(\frac{75}{2}m_i^2 + \frac{53}{2}m_i m_j + 13m_j^2) \ .$$



(a) forward ConvUnit2 (b) split and re-group (c) forward ConvUnit3 (d) forward ConvUnit4

Figure 2. Feedforward inference of MuffNet parameterized by M matrix.

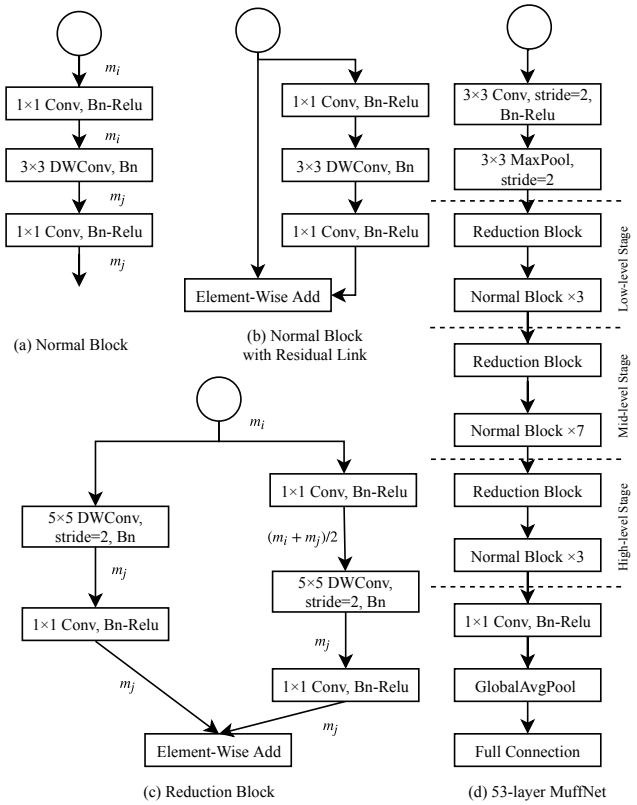


Figure 3. (a)-(c): Convolutional Blocks. (d): MuffNet structure diagram. m_i: number of input channels. m_j: number of output channels.

The total FLOPs of the structure parameterized by M is

$$\mathrm{FLOP}(M) = \sum_{i=1}^{L-1} \mathrm{FLOP}_i \ .$$

3.3. MuffNet: Put Everything Together

We now present the complete design of the MuffNet. The network diagram is given in Figure 3(d). Following the common practice, we assume the input image size is 224×224. The MuffNet in Figure 3(d) consists of three stages where we downsample the feature map to 28×28 (low-level stage), 14×14 (mid-level stage) and 7×7 (high-level stage) respectively. Within each stage, all layers are pairwise connected. There is no cross-stage shortcut links. We use reduction block to downsample the feature maps between two stages.

The MuffNet Figure 3(d) has 53 layers in total. We stack 3,7,3 normal blocks (with or without residual link) in the three stages respectively. In the first two layers we use 3×3 convolution and max-pooling with stride 2 to fast downsample the feature map to 56×56. After the last normal block, we expand the channels to 1024 via 1×1 convolution. The final output of the network is generated by a global average pooling and then a fully connected layer.

If the image size is under 56×56, we only need two downsampling layers. In this case the strides of the first 3×3 convolution and the first reduction block in Figure 3(d) are changed to 1. The max-pooling layer is removed too.

The feature map sharing parameter matrix M could be specified manually. We define the following three patterns as we find them effective in practice:

Average sharing The output channels are split evenly. That is $M_{i,j_1} = M_{i,j_2}$ for $j_1, j_2 \geq i$.

Linear sharing The number of shared channels is decayed linearly according the level distance. That is $M_{i,j} = O(1/(j-i+1))$.

Geometric sharing The number of shared channels is decayed exponentially according the level distance. That is $M_{i,j} = O(1/2^{j-i+1})$.

For comparison purpose, we define the dense sharing to imitate the feature map sharing pattern of DenseNet. In dense sharing, the input channels are the concatenation of outputs of all previous layers.

4. Theoretical Analysis

In this section, we will show that by multi-layer feature federation, the network is more capable in fitting high frequency components under the same computational budget. Therefore, the learning capacity and efficiency of the network is improved.

Suppose that we have a plain feedforward convolutional network of L layers with ReLU activation. Each layer has m_i channels. The input image size is $s_0 \times s_0$ with only one channel. Following the notation in [39], let N_f denote the number of linear regions split by the network activation patterns. The frequency vector is denoted as $\boldsymbol{\omega}$. Denote the Fourier function of the network as $\hat{f}(\omega)$. By Theorem 1 in [39], the magnitude of the frequency $\boldsymbol{\omega}$ is bounded by the following lemma.

Lemma 1 (Theorem 1 in [39]). *With probability almost one,*

$$|\hat{f}(\boldsymbol{\omega})| \leq cN_f \prod_{i=1}^{L} m_i \|\boldsymbol{\omega}\|^{-L-1} \tag{1}$$

where c is a universal constant.

To bound N_f, we refer to the following lemma.

Lemma 2 (Theorem 1 in [32]). *N_f is upper bounded by*

$$N_f \leq c(\prod_{i=1}^{L} m_i)^{(s_0^2)} .$$

Combining Lemma 1 and Lemma 2, we get the following key corollary.

Corollary 3. $|\hat{f}(\boldsymbol{\omega})| \leq c(\prod_{i=1}^{L} m_i)^{(s_0^2+1)} \|\boldsymbol{\omega}\|^{-L-1}$.

Corollary 3 reveals several important information. The term $\|\boldsymbol{\omega}\|^{-L-1}$ indicates an exponential decaying in high frequency components along the network depth. The order of frequency magnitude controlled by the layer width is polynomial. As the computational cost is on order of $O(\sum_i m_i m_{i+1})$, it is better to distribute the computation uniformly to all layers so that $|\hat{f}(\boldsymbol{\omega})|$ is maximized. This coincides with common practice: when we downsample the feature map, we usually double the number of channels.

Next, we would like to show that by federating feature maps of multiple depths, we can improve the dynamic of the network under a given computational budget. For sake of simplicity, we only consider layers in one stage. We set

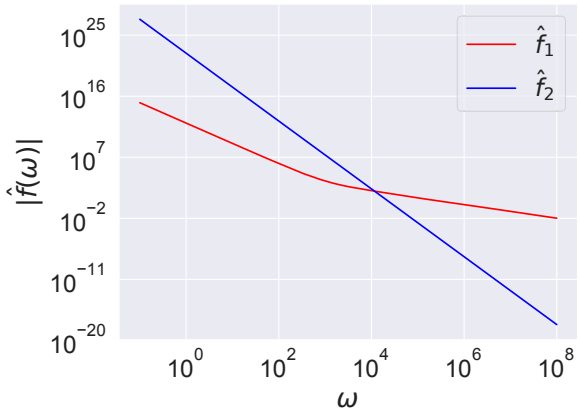


Figure 4. Spectrum of the networks.

$M_{i,j} = a$. From input image to the first feature map layer, we use a full convolution with kernel size 3. The FLOPs from the input image to the first layer is $9s_0^2 aL$. Counting from the second row and the second column, the sub-matrix $M_{2:L,2:L}$ has $(L-1)^2/2 + (L-1)$ non-zero elements. The total FLOPs is then

$$\begin{aligned}\text{FLOP}_1 =& 9as_0^2 L + 3s_0^2 \sum_{i=2}^{L}(\sum_{k=1}^{i-1} a)(\sum_{k=i}^{L} a) \\ =& 9as_0^2 L + 3a^2 s_0^2 \sum_{i=2}^{L}(i-1)(L+1-i) \\ =& 9as_0^2 L + a^2 s_0^2 L(L-1)(L+1)/2 \,.\end{aligned}$$

The corresponding frequency distribution upper bound is

$$|\hat{f}_1(\boldsymbol{\omega})| \leq \sum_{i=1}^{L} ca^{(i)(s_0^2+1)} \|\boldsymbol{\omega}\|^{-i-1} .$$

Now consider a plain feedforward network with layer width m. The FLOPS is

$$\text{FLOPS}_2 = 9s_0^2 m + s_0^2(2m^2 + 9m)(L-1) \,.$$

To compare two structures under the same budget $\text{FLOPS}_1 = \text{FLOPS}_2$, we set

$$\begin{aligned}m =& (-9L + \sqrt{\Delta})/(4\,(L-1)) \\ \Delta \triangleq& 81L^2 - 72aL + 72aL^2 + 4a^2 L - 4a^2 L^2 \\ & - 4a^2 L^3 + 4a^2 L^4 \,.\end{aligned}$$

The frequency distribution is

$$|\hat{f}_2(\boldsymbol{\omega})| \leq cm^{L(s_0^2+1)} \|\boldsymbol{\omega}\|^{-L-1} .$$

To visualize spectrums of the two structures, we set $c = 1, s_0 = 1, a = 32, L = 4$ and then plot the upper bounds of

the frequency distribution $|\hat{f}_1(\boldsymbol{\omega})|$ and $|\hat{f}_2(\boldsymbol{\omega})|$ in Figure 4. The x-axis is the frequency ω. The y-axis is the magnitude upper bound $|\hat{f}(\boldsymbol{\omega})|$. The spectrum of the MuffNet $\hat{f}_1(\boldsymbol{\omega})$ is the red line curve. The spectrum of the plain feedforward network $\hat{f}_2$ is the blue curve. The blue line has a larger magnitude at low frequency. After $\omega \geq 10^4$, the blue line decays below the red line. Figure 4 shows that the spectrum of the MuffNet is more uniformly distributed therefore is a better universal approximator for high frequency functions.

5. Experiments

We first describe our experiment settings on CIFAR [20], ImageNet [47] and MS COCO [26]. Then we conduct ablation study on ImageNet to find the best MuffNet structure. We focus on models around 40 MFLOPs in the ablation study for quick experiments. After fixing the best MuffNet structure, we generalize the structure to models of 140 MFLOPs and 300 MFLOPs.

5.1. Dataset and Training Settings

CIFAR Following the standard data augmentation scheme, we zero-pad each CIFAR image with 4 pixels and then randomly crop to 32×32. After cropping the image is randomly flipped horizontally. We use SGD with momentum 0.9, weight decay 10^{-4}, batch size 128. The initial learning rate is 0.1. The learning rate is decayed to zero following cosine function. We terminate training at 300 epochs.

ImageNet We use the same data augmentation scheme as InceptionV3 [54] for training. In testing stage we first rescale images to 256×256 and then center-crop to 224×224. In the ablation studies, we set the initial learning rate to 0.4. In the final benchmark, we set the initial learning rate to 0.5. For all experiments we use SGD with momentum 0.9, weight decay 4×10^{-4}, batch size 1024. For other training/evaluation settings we follow [11].

MS COCO We adopt SSD [28] as the detection framework and use the public domain gluon-cv code with default settings for training . We use COCO train+val for training and COCO minival for evaluation. All backbone models are pre-trained on ImageNet and then finetuned on detection dataset. We report mAP (mean Average Precision, COCO challenge metrics) against number of parameters and FLOPs.

5.2. Ablation Study

We search for the optimal feature map sharing patterns of three stages in the MuffNet, that is, average sharing, linear sharing and geometric sharing for low-level, mid-level and high-level stages.

Sharing Pattern	Param	FLOPs	ACC
none+none+none	1.38 M	42 M	59.69%
geometric+none+none	1.38 M	42 M	59.77%
average+none+none	1.38 M	42 M	**59.79%**
linear+none+none	1.38 M	42 M	59.75%
dense+none+none	1.38 M	42 M	59.71%

Table 1. Low-level Stage Sharing Patterns Comparison

Sharing Pattern	Param	FLOPs	ACC
none+none+none	1.38 M	42 M	59.69%
none+geometric+none	1.38 M	42 M	**60.34%**
none+average+none	1.38 M	42 M	59.89%
none+linear+none	1.38 M	42 M	60.13%
none+dense+none	1.38 M	42 M	59.76%

Table 2. Mid-level Stage Sharing Patterns Comparison

According to Figure 3(d), we design three groups of ablation experiments. We compare different feature map sharing patterns in 28×28, 14×14 and 7×7 stages. When we compare sharing patterns in one stage, we always use normal block with residual link in the other two stages. In the following experiments, we keep the FLOPs of each stage to be the same.

Low-level Stage Table 1 summarizes the top-1 accuracies of different feature map sharing patterns in low-level stage (28×28 stage). In the first column, we denote 'none' if we use the conventional residual blocks without feature map sharing in a stage. The 'dense' denotes the feature map sharing pattern used in DenseNet. 'linear+dense+none' means we use linear sharing in the low-level stage, dense sharing in the mid-level stage, and no sharing in the high-level stage. We find that in low-level stage, it is better to share feature maps rather than no sharing. Different feature map sharing patterns are comparable in accuracy. The average feature map sharing is slightly better.

Mid-level Stage Table 2 summarizes the top-1 accuracies of different feature map sharing patterns in mid-level stage (14×14 stage). Since we stack more blocks in this stage, the sharing pattern has significant impact. Similar to the low-level stage, conventional feedforward structure without feature map sharing is the worst choice. The best pattern is the geometric sharing, about 0.2% better than the rest. Linear sharing is also better than average sharing. This indicates that in the mid-level stage, we should enhance the information sharing between consecutive layers.

High-level Stage Table 3 summarizes the top-1 accuracies of different feature map sharing patterns in high-level stage (7×7 stage). In this stage, the conventional residual

Sharing Pattern	Param	FLOPs	ACC
none+none+none	1.38 M	42 M	**59.69%**
none+none+geometric	1.38 M	42 M	58.93%
none+none+average	1.38 M	42 M	58.95%
none+none+linear	1.38 M	42 M	58.87%
none+none+dense	1.38 M	42 M	58.56%

Table 3. High-level Stage Sharing Patterns Comparison

Sharing Pattern	FLOPs	ACC
none+none+none	42 M	59.69%
geometric+geometric+none	42 M	60.39%
geometric+average+none	42 M	60.31%
average+geometric+none	42 M	**60.60%**
average+average+none	42 M	60.30%
linear+linear+none	42 M	59.94%
linear+geometric+none	42 M	60.32%
linear+average+none	42 M	60.10%
geometric+linear+none	42 M	60.31%
average+linear+none	42 M	60.52%

Table 4. High-level Stage Sharing Patterns Comparison (Combination)

blocks without sharing is at least 0.7% better in top-1 accuracy. This shows that in high-level stage feature map sharing is discouraged.

Combination of low-level and mid-level stages In the above three ablation studies, we find that it is better not to share feature maps in high-level stages. In Table 4, we fix the high-level stage sharing pattern to be 'none' and we test the combinations of feature map sharing patterns of the low-level and mid-level stages. Since we choose the right pattern in the high-level stage, the overall accuracy in this experiment is better than above ones. In the above experiments, we find that the low-level stage prefers average sharing and mid-level stage prefers geometric sharing. In Table 4 we observe the same trend again. The 'average+geometric+none' achieves the best top-1 accuracy 60.6%. The 'average+linear+none' is the second best, about 0.08% falls behind.

Based on the above experiments, we propose our MuffNet configuration as following. The low-level stage consists of 3 normal blocks with average feature map sharing. The mid-stage consists of 7 normal blocks with geometric feature map sharing. The high-level stage consists of 3 residual blocks without feature map sharing, followed by fully connected layer. The M matrix of this structure is visualized in Figure 5. In the figure, each colored block represents a group of feature map channels parametrized by $M_{i,j}$. The top-left is $M_{0,0}$ and the right-bottom is $M_{15,15}$. The y-axis indices the source layer whose output channels

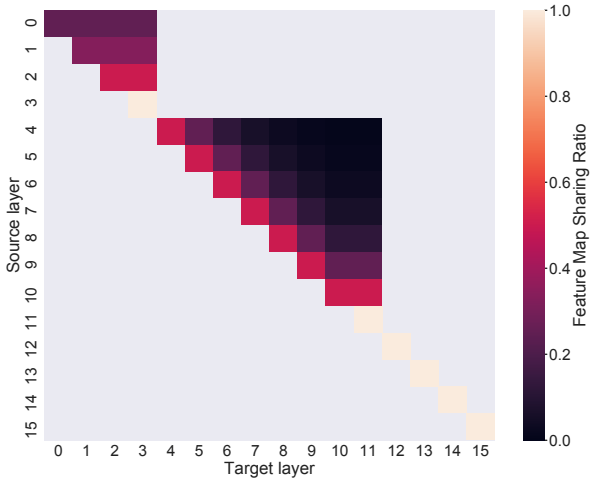


Figure 5. MuffNet M matrix visualization

Model	FLOPs	CIFAR10	CIFAR100
MuffNet_0.5	41 M	**93.44%**	**71.83%**
ShuffleNetV2_0.5	40 M	93.25%	71.42%
MobileNet_0.25	41 M	92.78%	71.29%
MobileNetV2_0.35	45 M	93.12%	71.34%
ResNet20	41 M	91.25%	-
MuffNet_1.0	174 M	**94.82%**	**76.23%**
ShuffleNetV2_1.0	174 M	94.60%	76.05%
MobileNet_0.5	156 M	93.94%	73.94%
MobileNetV2_0.75	175 M	94.71%	75.88%
ResNet56	126 M	93.93%	-
ResNet110	253 M	93.57%	-
DenseNet40	206 M	94.06%	75.58%
MuffNet_1.5	373 M	95.46%	77.65%
ShuffleNetV2_1.5	370 M	95.16%	77.32%
MobileNet_0.75	345 M	94.55%	75.90%
MobileNet_1.0	612 M	94.54%	76.47%
MobileNetV2_1.0	298 M	95.21%	77.44%
DenseNet_BC_100	288 M	**95.49%**	**77.73%**

Table 5. Top-1 Accuracy on CIFAR-10 and CIFAR-100

are split and the x-axis indices the target layer who takes the corresponding feature map group as input. For details about channel configuration, please check appendix.

5.3. Image Classification Benchmark

We compare MuffNet with popular baseline models on CIFAR-10, CIFAR-100 and ImageNet at 40 , 140 and 300 MFLOPS. The landmark budgets of FLOPs are selected as the same in [31] for fair comparison.

CIFAR We summarize top-1 accuracies on CIFAR-10 and CIFAR-100 in Table 5. Since the original papers of

Model	Param	FLOPs	Acc
MuffNet_0.5	1.4 M	42 M	**62.1%**
ShuffleNetV2_0.5	1.4 M	41 M	60.3%
ShuffleNet_0.5 (G=3)	-	38 M	56.8%
MobileNet_0.25	0.5 M	41 M	50.6%
MobileNetV2_0.35	-	50 M	60.3%
IGCV2_0.25	0.5 M	46 M	54.9%
MuffNet_1.0	2.3 M	146 M	**69.9%**
ShuffleNetV2_1.0	2.3 M	146 M	69.4%
ShuffleNet_1.0(G=3)	-	140 M	67.4%
MobileNet_0.5	1.3 M	149 M	63.7%
MobileNetV2_0.75	2.7 M	190 M	69.6%
IGCV2_0.5	1.3 M	156 M	65.5%
IGCV3-D(0.7)	2.8 M	210 M	68.5%
MuffNet_1.5	3.4 M	300 M	**73.1%**
ShuffleNetV2_1.5	3.5 M	299 M	72.6%
ShuffleNet_1.5(G=3)	3.4 M	292 M	71.5%
MobileNet_0.75	2.6 M	325 M	68.4%
MobileNet_1.0	4.2 M	569 M	70.6%
MobileNetV2_1.0	3.4 M	300 M	72.0%
CondenseNet(G=C=8)	2.9 M	274 M	71.0%
IGCV2-1.0	4.1 M	564 M	70.7%
IGCV3-D	3.5 M	318 M	72.2%

Table 6. Top-1 Accuracy on ImageNet

ShuffleNet and MobileNet did not report their performance on CIFAR, we report our replicated results. For fairness in the experiments we remove the first 3×3 convolution in the ShuffleNetV2, change the stride of the first reduction block to 1 and remove the max-pooling layer.

The proposed MuffNet outperforms baseline methods at 40 and 140 MFLOPs on both CIFAR-10 and CIFAR-100. At 300 MFLOPs, DenseNet achieves the best top-1 accuracy on both datasets. We notice that at 300 MFLOPs, the accuracies of different models are very close to each other. Since the image size of CIFAR is only 32×32, the overall gain of MuffNet is around 0.2%.

ImangeNet We summarize top-1 accuracies on ImageNet in Table 6. For easy comparison, we provide the scatter plot of top-1 accuracies of MuffNet, ShuffleNet and MobileNet in Figure 6. At 40 MFLOPs, the MuffNet-0.5 achieves 62.1% top-1 accuracy. This number is 1.8% better than the second best model ShuffleNetV2-0.5. Comparing to ShuffleNetV1-0.5, IGCV2-0.25 and other models, MuffNet outperforms them by a margin of 5%. This shows that the MuffNet is highly efficient for light-weight models. At 140 MFLOPs, MuffNet-1.0 achieves 69.9% top-1 accuracy. This number is 0.5% better than the second best model ShuffleNetV2-1.0 and 2.2% better than ShuffleNet-1.0 (G=3). When comparing to IGCV3-D (0.7), MuffNet

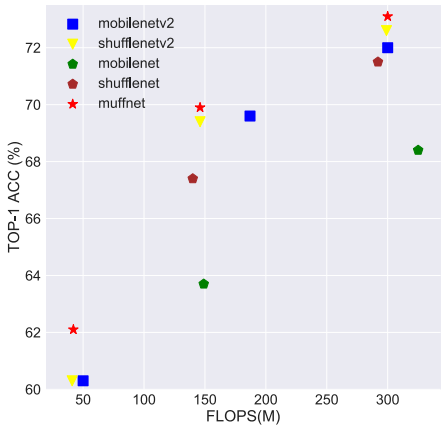


Figure 6. Top-1 accuracy v.s. FLOPs

Model	Param	FLOPs	mAP
MuffNet	3.4 M	300 M	**22.0%**
ShuffleNetV2	3.5 M	299 M	21.8%
MobileNetV2	3.4 M	300 M	21.9%
MobileNet	4.2 M	569 M	21.7%

Table 7. mAP on MS COCO. SSD512 as backbone.

is 1.1% better with 1/4 less FLOPs. At 300 MFLOPs, MuffNet-1.5 is still the best model at this scale, slightly better than ShuffleNetV2-1.5 (+0.5%). However, the number of parameters of MuffNet is smaller than ShuffleNetV2-1.5 which means that MuffNet is more memory efficient.

5.4. Object Detection Benchmark

We report the mAP of MuffNet, ShuffleNet and MobileNet on MS COCO dataset in Table 7. We use SSD512 as backbone for all models. Again, in the detection task MuffNet is superior than baseline methods at 300 MFLOPs. MuffNet achieves 22.0% mAP, 0.1% better than the second best model MobileNetV2 and 0.2% better than ShuffleNetV2. Comparing to MobileNet, MuffNet is 0.3% better while its computational cost is only half of MobileNet.

6. Conclusion

We propose a novel multi-layer feature federation structure, MuffNet, for mobile deep learning. MuffNet is a densely connected network with budgeted memory and computation. By ensembling convolutional sub-networks of different depths, the MuffNet allow both low and high frequencies pass through the network, resulting in a better universal approximator. MuffNet achieves the state-of-the-art performance in image classification and object detection benchmarks, especially for small models under 45 MFLOPs.

References

[1] H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *arXiv:1812.00332 [cs, stat]*, 2018. 3

[2] S. Changpinyo, M. Sandler, and A. Zhmoginov. The Power of Sparsity in Convolutional Neural Networks. *arXiv:1702.06257 [cs]*, 2017. 2

[3] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang. AdaNet: Adaptive Structural Learning of Artificial Neural Networks. In *ICML*, pages 874–883, 2017. 2

[4] X. Dong, S. Chen, and S. J. Pan. Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon. In *NIPS*, pages 4857–4867, 2017. 2

[5] G. Huang, S. Liu, L. v d Maaten, and K. Q. Weinberger. CondenseNet: An Efficient DenseNet Using Learned Group Convolutions. In *CVPR*, pages 2752–2761, 2018. 3

[6] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. 2017. 1

[7] A. Graves, A.-r. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, pages 6645–6649, 2013. 1

[8] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. In *NIPS*, pages 1135–1143, 2015. 2

[9] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. In *ICCV*, pages 2980–2988, 2017. 1

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016. 1

[11] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li. Bag of Tricks for Image Classification with Convolutional Neural Networks. *arXiv:1812.01187 [cs]*, 2018. 6

[12] Y. He, X. Zhang, and J. Sun. Channel Pruning for Accelerating Very Deep Neural Networks. In *ICCV*, pages 1398–1406, 2017. 2

[13] 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:1704.04861 [cs]*, 2017. 2, 3

[14] H. Hu, D. Dey, A. Del Giorno, M. Hebert, and J. A. Bagnell. LogDenseNet: How to Sparsify a DenseNet. *arXiv:1711.00002 [cs]*, 2017. 2, 3

[15] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. *arXiv: Neural and Evolutionary Computing*, 2016. 2

[16] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. In *CVPR*, pages 2261–2269, 2017. 1, 2

[17] 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 <0.5MB model size. *arXiv:1602.07360 [cs]*, 2016. 2, 3

[18] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population Based Training of Neural Networks. *arXiv:1711.09846 [cs]*, 2017. 2

[19] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu. Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes. *arXiv:1807.11205 [cs, stat]*, 2018. 1

[20] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 6

[21] V. Lebedev and V. S. Lempitsky. Fast ConvNets Using Group-Wise Brain Damage. In *CVPR*, pages 2554–2564, 2016. 2

[22] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin. Extremely Low Bit Neural Network: Squeeze the Last Bit Out With ADMM. In *AAAI*, 2018. 2

[23] F. Li, B. Zhang, and B. Liu. Ternary Weight Networks. *arXiv:1605.04711 [cs]*, 2016. 2

[24] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning Filters for Efficient ConvNets. *arXiv:1608.08710 [cs]*, 2016. 2

[25] S. R. Li, J. Park, and P. T. P. Tang. Enabling sparse winograd convolution by native pruning. *CoRR*, abs/1702.08597, 2017. 2

[26] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common Objects in Context. In *ECCV*, pages 740–755, 2014. 6

[27] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy. Sparse Convolutional Neural Networks. In *CVPR*, pages 806–814, 2015. 2

[28] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single Shot MultiBox Detector. In *ECCV*, pages 21–37, 2016. 6

[29] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning Efficient Convolutional Networks through Network Slimming. In *ICCV*, pages 2755–2763, 2017. 2

[30] J.-H. Luo, J. Wu, and W. Lin. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. In *ICCV*, pages 5068–5076, 2017. 2

[31] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *ECCV*, pages 116–131, 2018. 7

[32] Maithra Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. On the Expressive Power of Deep Neural Networks. In *ICML*, pages 2847–2854, 2017. 5

[33] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally. Exploring the Granularity of Sparsity in Convolutional Neural Networks. In *CVPR*, pages 1927–1934, 2017. 2

[34] M. Masana, J. van de Weijer, L. Herranz, A. D. Bagdanov, and J. M. Alvarez. Domain-Adaptive Deep Network Compression. In *ICCV*, pages 4299–4307, 2017. 2

[35] Matthieu Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *NIPS*, pages 3123–3131, 2015. 2

[36] Mengye Ren, A. Pokrovsky, B. Yang, and R. Urtasun. SBNet: Sparse Blocks Network for Fast Inference. *arXiv:1801.02108 [cs]*, 2018. 2

[37] Mikhail Figurnov, A. Ibraimova, D. P. Vetrov, and P. Kohli. PerforatedCNNs: Acceleration through Elimination of Redundant Convolutions. In *NIPS*, pages 947–955. 2016. 2

[38] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning Convolutional Neural Networks for Resource Efficient Inference. *arXiv:1611.06440 [cs, stat]*, 2016. 2

[39] Nasim Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. A. Hamprecht, Y. Bengio, and A. Courville. On the Spectral Bias of Neural Networks. *arXiv:1806.08734 [cs, stat]*, 2018. 2, 5

[40] J. Y.-H. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, pages 4694–4702, 2015. 1

[41] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In *NIPS*, volume 28, pages 442–450, 2015. 2

[42] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient Neural Architecture Search via Parameters Sharing. In *ICML*, pages 4092–4101, 2018. 2

[43] A. Prabhu, G. Varma, and A. M. Namboodiri. Deep Expander Networks: Efficient Deep Networks from Graph Theory. In *ECCV*, pages 20–36, 2018. 2

[44] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized Evolution for Image Classifier Architecture Search. *arXiv:1802.01548 [cs]*, 2018. 2

[45] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. V. Le, and A. Kurakin. Large-Scale Evolution of Image Classifiers. In *ICML*, pages 2902–2911, 2017. 2

[46] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6):1137–1149, 2017. 1

[47] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 6

[48] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets. In *ICASSP*, pages 6655–6659, 2013. 2

[49] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. 2018. 2

[50] S. Saxena and J. Verbeek. Convolutional Neural Fabrics. In *NIPS*, pages 4053–4061, 2016. 2

[51] R. Shin, C. Packer, and D. Song. Differentiable Neural Network Architecture Search. In *Workshop of CVLR*, 2019. 2

[52] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*, 2015. 1

[53] K. Sun, M. Li, D. Liu, and J. Wang. IGCV3: Interleaved Low-Rank Group Convolutions for Efficient Deep Neural Networks. *arXiv:1806.00178 [cs]*, 2018. 2

[54] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, pages 2818–2826, 2016. 6

[55] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *arXiv:1807.11626 [cs]*, 2018. 2

[56] Vadim Lebedev, V. Lebedev, Y. Ganin, M. Rakhuba, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. In *ICLR*, 2015. 2

[57] T. Véniat and L. Denoyer. Learning Time/Memory-Efficient Deep Architectures With Budgeted Super Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3492–3500, 2018. 2

[58] D. Wan, F. Shen, L. Liu, F. Zhu, J. Qin, L. Shao, and H. Tao Shen. TBN: Convolutional Neural Network with Ternary Inputs and Binary Weights. In *ECCV*, pages 315–332, 2018. 2

[59] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning Structured Sparsity in Deep Neural Networks. In *NIPS*, pages 2074–2082, 2016. 2

[60] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. *arXiv:1812.03443 [cs]*, 2018. 2, 3

[61] G. Xie, J. Wang, T. Zhang, J. Lai, R. Hong, and G.-J. Qi. IGCV2: Interleaved Structured Sparse Convolutional Neural Networks. *arXiv:1804.06202 [cs]*, 2018. 2

[62] Yann LeCun, J. S. Denker, and S. A. Solla. Optimal Brain Damage. In *NIPS*, volume 2, pages 598–605, 1989. 2

[63] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer. ImageNet Training in Minutes. In *International Conference on Parallel Processing*, pages 1:1–1:10, New York, NY, USA, 2018. 1

[64] D. Zhang, J. Yang, D. Ye, and G. Hua. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. In *ECCV*, pages 373–390, 2018. 2

[65] X. Zhang, X. Zhou, M. Lin, and J. Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *CVPR*, pages 6848–6856, 2018. 2

[66] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained Ternary Quantization. In *ICLR*, 2017. 2

[67] L. Zhu, R. Deng, M. Maire, Z. Deng, G. Mori, and P. Tan. Sparsely Aggregated Convolutional Networks. In *ECCV*, pages 192–208, 2018. 2, 3

[68] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware Channel Pruning for Deep Neural Networks. In *NIPS*, pages 883–894. 2018. 2