SIAM J. SCI. COMPUT.
Vol. 45, No. 3, pp. S307–S328

MGIC: MULTIGRID-IN-CHANNELS NEURAL NETWORK ARCHITECTURES*

MOSHE ELIASOF†, JONATHAN EPHRATH‡, LARS RUTHOTTO§, AND ERAN TREISTER‡

Abstract. We present a multigrid-in-channels (MGIC) approach that tackles the quadratic growth of the number of parameters with respect to the number of channels in standard convolutional neural networks (CNNs). Thereby our approach addresses the redundancy in CNNs that is also exposed by the recent success of lightweight CNNs. Lightweight CNNs can achieve comparable accuracy to standard CNNs with fewer parameters; however, the number of weights still scales quadratically with the CNN's width. Our MGIC architectures replace each CNN block with an MGIC counterpart that utilizes a hierarchy of nested grouped convolutions of small group size to address this. Hence, our proposed architectures scale linearly with respect to the network's width while retaining full coupling of the channels as in standard CNNs. Our extensive experiments on image classification, segmentation, and point cloud classification show that applying this strategy to different architectures like ResNet and MobileNetV3 reduces the number of parameters while obtaining similar or better accuracy.

Key words. alternative CNN architectures, multilevel neural networks, compact and lightweight neural networks

MSC codes. 68T07, 65N55, 68T45

DOI. 10.1137/21M1430194

1. Introduction.

Convolutional neural networks (CNNs) [43] have achieved impressive accuracy for image classification, semantic segmentation, solution of partial differential equations, and other tasks [21, 39, 42]. The main idea behind CNNs is to define the linear operators in the neural network as convolutions with local kernels. This increases the network's computational efficiency (compared to the original class of networks) due to the essentially sparse convolution operators and the considerable reduction in the number of weights. The general trend in the development of CNNs has been to make deeper, wider, and more complicated networks to achieve higher accuracy [59].

In practical applications of CNNs, a network's feature maps are divided into channels, and the number of channels, c, can be defined as the width of the layer. A standard CNN layer connects any input channel with any output channel. Hence, the number of convolution kernels per layer is equal to the product of the number of input channels and output channels. Assuming the number of output channels is

* Received by the editors June 30, 2021; accepted for publication (in revised form) September 21, 2022; published electronically June 26, 2023.
https://doi.org/10.1137/21M1430194
Funding: This work was supported by the Israel Innovation Authority through the Avatar consortium and by grant 2018209 from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel. The work of the first author was supported by a Kreitman high-tech scholarship. The work of the third author was supported by NSF DMS 1751636.

† Corresponding author. Computer Science Department, Ben-Gurion University of the Negev, Beer Sheva, 8410501, Israel (eliasof@post.bgu.ac.il).

‡ Computer Science Department, Ben-Gurion University of the Negev, Beer Sheva, 8410501, Israel (ephrathj@post.bgu.ac.il, erant@cs.bgu.ac.il).

§ Departments of Mathematics and Computer Science, Emory University, Atlanta, GA 30322 USA (lruthotto@emory.edu).

proportional to the number of input channels, this $\mathcal{O}(c^2)$ growth of operations and parameters causes immense computational challenges. When the number of channels is large, convolutions are the most computationally expensive part of the training and inference of CNNs. Wide architectures exacerbate this trend with hundreds or thousands of channels, which are particularly effective in classification tasks involving a large number of classes. Increasing the network's width is advantageous in terms of accuracy and hardware efficiency compared to deeper, narrower networks [73]. However, the quadratic scaling causes the number of weights to reach hundreds of millions and beyond [36], and the computational resources (power and memory) needed for training and making predictions with such CNNs surpasses the resources of common systems [3]. This motivates the deep learning community to design more efficient network architectures with competitive performance.

Among the first approaches to reduce the number of parameters in CNNs are the methods of pruning [27, 28, 44] and sparsity-promoting [10, 26] that aimed to limit the connectivity between channels and have been typically applied to already trained networks. Once a network is trained, a substantial number of its weights can be removed with hardly any degradation of its accuracy. However, the resulting connectivity of these processes is typically unstructured, which may lead to inefficient deployment of the networks on hardware. Still, pruning serves as a proof-of-concept that the full connectivity between channels is superfluous, i.e., there is a redundancy in CNNs [51]. By contrast, we reduce the network architecture by using structured convolution operators, to enable efficient, balanced computations during training and inference.

Another recent effort to reduce the number of parameters of CNNs features lightweight architectures based on spatial grouped convolutions. The idea is to apply the more computationally expensive spatial convolutions—those involving larger-than-a scalar filters, say, 3×3 or 5×5—in small disjoint groups, creating a block-diagonal "local" connectivity between channels (see Figure 1). The extremity of grouped convolutions (diagonal connectivity) are the so-called depthwise convolutions which filter each input channel separately. Such depthwise convolutions have been commonly used together with pointwise 1×1 convolutions, which couple all the channels but with a single scalar for each pair of channel interactions instead of larger stencils like 3×3 (a 1×1 convolution is essentially a simple dense matrix-matrix multiplication). This way, the popular MobileNets [34, 35, 58] involve significantly fewer parameters than standard networks, while achieving comparable performance. The majority of

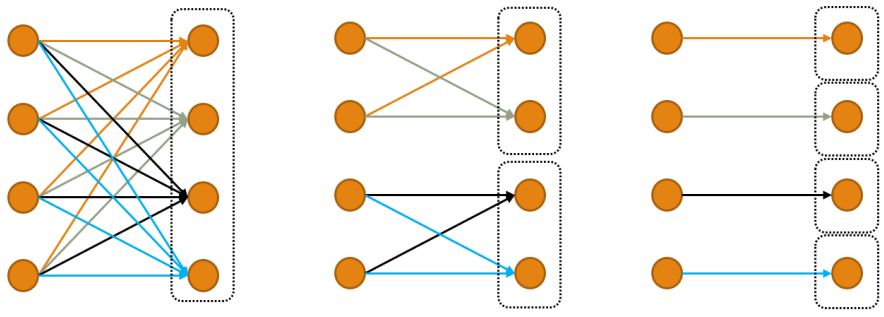

FIG. 1. *Standard fully coupled (left), grouped (block-diagonal) convolution operator with two groups (middle), and a depthwise (diagonal) convolution (right).*

the weights in MobileNets are in the pointwise operators, which scale with $\mathcal{O}(c^2)$. The strength of MobileNets, and their improvement, EfficientNet [61], is the inverse bottleneck structure, which takes a narrow network (with relatively few channels) and expands it by a significant factor to perform the depthwise and nonlinear activation. This way, although the number of parameters scales quadratically in the width in the 1×1 operators, it aims to maximize the spatial convolutions and activations as much as possible to increase the expressiveness of the network. The ShuffleNet [49] reduces the parameters of the pointwise operator by applying 1×1 convolutions to half of the channels and then shuffling them. Another closely related architecture is LeanConvNets [19], which employs 3- and 5-point convolution stencils to reduce computational complexity, together with grouped convolutions. The recent GhostNet [25] divides the feature space by some hyperparameter (typically, 2 or 4) such that only a subset of the channels is convolved by a 1×1 convolution, and the remaining channels are obtained from the densely learned portion of the feature space by depthwise convolutions.

In this work, we aim to obtain full connectivity between the channels of CNNs using locally grouped convolutions *only*. To obtain this, we adopt the multigrid approach [6] that is known to be effective in solving various large problems using local processing only on a hierarchy of grids or meshes. Such multigrid methods are primarily used to solve differential equations and other graphical problems (e.g., Markov chains [16]). They are based on a principle that a local iterative solution process on a fine grid can effectively eliminate errors that involve short distances on the grid (also known as "smoothing") but cannot reduce long-distance errors. Such errors can be approximated on a coarser grid, leading to two advantages. First, coarse grid procedures are less expensive than fine grid procedures as they involve fewer grid points. Second, traversing different scales leads to faster convergence. An alternative interpretation of this paradigm is that the multigrid hierarchy efficiently transfers information across all the grid points using local processing only, at a different scale at each level. Classical multigrid methods rely on the multiscale representation of functions in space but can also be used to tackle temporal problems [20]. Multigrid has been abundantly applied in the context of partial differential equations [6], but also in other areas such as optimization [4, 5, 62], Markov chains [16], and graphs [46, 52]. Multigrid approaches have also been used to warm-start the training of CNNs on high-resolution images with training on low-resolution images [23], adopting a multiscale approach in space. Similarly, [29, 38, 56] define multiscale architectures that extract and combine features from different resolution images scales. The DeepLabV3 architecture for semantic segmentation [12] also exploits multiscale representations. Other works [11, 14, 64] present different strategies to exploit the spatial multiscale structure and representation of the feature maps throughout CNNs to improve the performance of standard networks. Multigrid has also been used in the layer dimension for residual networks, e.g., to warm-start the training of a deep network by interpolating the weights of a trained shallow network with a larger step size [9], and improving or parallelizing the training through the layers [22, 40]. While the mentioned works apply the multigrid idea either in space or in layers (depth), in this work we use the multigrid idea in the channel space (width).

In this work we propose a *multigrid-in-channels* (MGIC) approach for building network architectures that connect all channels at each layer using local grouped convolutions only. We achieve this full coupling via a hierarchical structure of the features in the channel space, and grouped convolutions with relatively small groups of fixed size at each level of the hierarchy. This yields a computational complexity

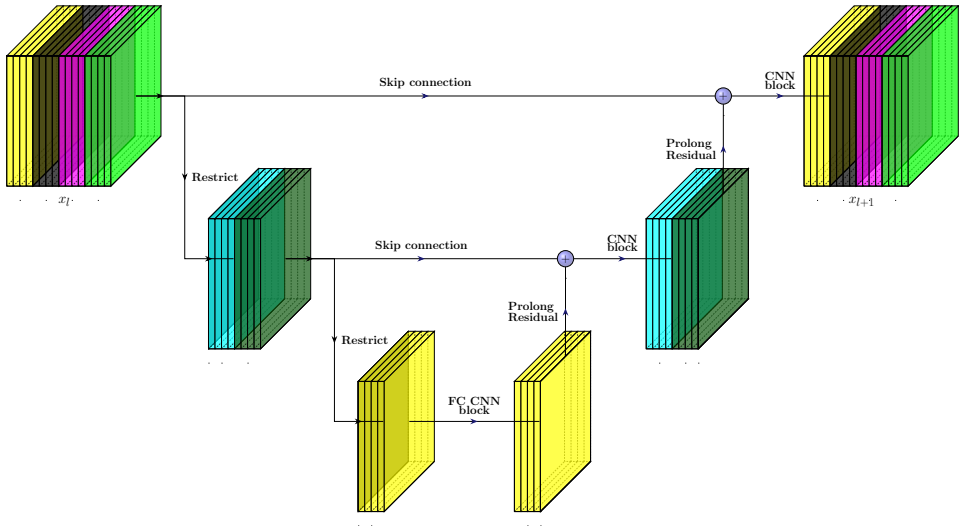

FIG. 2. *A three-level multigrid block for* 16 *input channels and a group size* $s_g = 4$. *Restrict and Prolong Residual denote grid transfer operators, which decrease and increase the number of channels, respectively. All the channels in the block are of the same spatial resolution. Each color denotes a group of channels that are mixed in a CNN block. The coarsest level uses a fully coupled CNN block.*

that scales linearly with the network's width. That is, for a fixed group size, doubling the number of channels in the network doubles the number of parameters. This allows us to use wider, deeper, and ultimately more expressive networks for a given computational budget.

Our MGIC block, which we show in Figure 2, can replace standard convolutional blocks in CNNs. At each level of the block, a particular width is considered in the channel space. Coarser levels in the block are defined by clusters of channels formed by averaging channels from different groups and therefore have fewer channels. On each level, we apply grouped convolutions on those clustered channels, which effectively connect different fine-level groups. We also note that other forms of local convolutions, like a chain or two-dimensional (2D) lattice structure, are also suitable in our framework, as long as they are of linear complexity. We chose grouped convolutions as they are the only type of local convolution in the channel space that is supported in deep learning frameworks.

Even though our MGIC block relies on local grouped convolutions, its multigrid structure enables communication between all channels. Our experiments on feature space representation using a MGIC block, as well as implicit function approximation, suggest that our method is of high capacity and expressiveness achieved in a linear scaling fashion. Furthermore, in our experiments on image classification, segmentation, and point cloud classification tasks, the MGIC block achieves competitive or superior performance with a relatively low number of parameters and FLOPs.

2. Toward efficient convolutions. Typical CNN architectures are composed of a series of blocks

$$\mathbf{x}_{l+1} = \text{CNN-block}(\mathbf{x}_l), \tag{2.1}$$

where $\mathbf{x}_l$ and $\mathbf{x}_{l+1}$ are the input and output features of the lth block, respectively. Each CNN block usually contains a sequence of basic layers, with associated weights

which are omitted in the following. For example, consider the bottleneck ResNet block [30] that reads

$$\mathbf{x}_{l+1} = \mathbf{x}_l + K_{l_3}\sigma(\mathcal{N}(K_{l_2}\sigma(\mathcal{N}(K_{l_1}\sigma(\mathcal{N}(\mathbf{x}_l)))))). \tag{2.2}$$

The operators K_{l_1} and K_{l_3} are fully coupled 1×1 convolutions, and K_{l_2} is a 3×3 convolution. The function $\sigma()$ is a nonlinear pointwise activation function, typically the ReLU function $\sigma(x) = \max(x, 0)$. $\mathcal{N}$ is a normalization operator often chosen to be a batch normalization. Architectures that follow a similar trend are MobileNets [34, 35, 58] and ResNeXt [70], in which K_{l_2} is a grouped (or depthwise) convolution to reduce the number of parameters and increase the ratio between activations and parameters.

A convolution layer takes c_{in} channels of feature maps and outputs c_{out} such channels. By definition, a fully coupled convolution is involved with $\mathcal{O}(c_{\text{in}} \cdot c_{\text{out}})$ parameters and FLOPs, i.e., both scale quadratically with the width. Since popular implementations of CNNs often use hundreds or thousands of channels, their full coupling leads to large computational costs and to millions of parameters for each layer, which may not be necessary.

To ease the computational costs, all or some of the convolution layers in the CNN block (2.1) may be grouped, dividing the input channels to several equally sized groups and applying a separate convolution kernel on each of them. In this work, we denote the number of channels in a group (its size) by s_g. For example, a standard fully coupled convolution with one group is defined by $s_g = c_{\text{in}}$, where c_{in} is the number of input channels, while a depthwise convolution is achieved by $s_g = 1$. As the group size gets smaller, the grouped convolutions involve fewer computations, at the cost of typically less expressive network architectures. That is because information cannot be shared between different groups in the feature maps during the grouped convolutions.

In this work, we propose to replace the CNN block in (2.1) by a novel multigrid block to obtain the forward propagation

$$\mathbf{x}_{l+1} = \text{MGIC block}(\mathbf{x}_l, \text{CNN block}, s_g, s_c), \tag{2.3}$$

which, as illustrated in Figure 2, uses a hierarchy of grids in the channel space and applies the original CNN block on the coarsest level. The parameter s_g defines the group size of the convolution operators in these CNN blocks, and s_c is the size of the coarsest grid. As we show in section 3.4, the number of parameters and FLOPs in the MGIC block scales *linearly* with respect to the number of channels, assuming that the group size is fixed. Note that the MGIC block is agnostic to its CNN block and therefore can be used for various CNN architectures, including future ones.

3. Multigrid-in-channels CNN architectures. In this section, we describe the MGIC block in detail. We start by defining the multigrid hierarchy. Then, we define the MGIC block and the grid transfer operators, which are essential to perform down- and up-sampling of the channel space, followed by a comprehensive description of the design of such a MGIC block. Finally, we compare the computational cost of a standard CNN layer with ours.

3.1. The multigrid hierarchy. The key idea of our multigrid architecture is to design a hierarchy of grids in the channel space (also referred to as "levels"), where the number of channels in the finest level corresponds to the original width of the network. The number of channels is halved between the levels until reaching the coarsest level, where the number of channels is smaller than or equal to the parameter s_c. Our

multigrid architecture is accompanied by a CNN block, like the ResNet block in (2.2), which is applied on each level. On the finest and intermediate levels, we only connect disjoint groups of channels using grouped convolutions. These convolutions have $\mathcal{O}(s_g \cdot c_{\text{in}})$ parameters, and we keep the group size s_g fixed throughout the network. Hence, as the network widens, the number of groups grows, and the number of parameters grows linearly. We allow interactions between all the channels on the coarsest grid, where we use the original CNN block without grouping. Since the coarsest grid contains only a few channels, this is not a costly operation. We note that our architecture is capable of performing more convolution layers and nonlinear activations per MGIC block, which is designed to replace a given CNN block. That is, our MGIC approach can yield models with higher capacity and expressiveness while retaining similar or lower computational cost, due to the use of grouped variants of the original, fully coupled CNN block.

3.2. The multigrid block. For simplicity, we assume that the CNN block and the MGIC block change neither the number of channels nor the spatial resolution of the images. That is, both $\mathbf{x}_l$ and $\mathbf{x}_{l+1}$ in (2.3) have c_{in} channels of the same spatial resolution. Given a CNN block, a group size s_g, and a coarsest grid size s_c, we define the multigrid block in Algorithm 1, and as an example we present and discuss a two-level version of it below. Here, the two-level hierarchy is denoted by levels $0, 1$, and $\mathbf{x}^{(0)} = \mathbf{x}_l$ are the input feature maps at the finest level (level 0). The two-level block is as follows:

$$\mathbf{x}^{(1)} = R_0 \mathbf{x}^{(0)}, \tag{3.1}$$

$$\mathbf{x}^{(1)} \leftarrow \text{CNN-block}(\mathbf{x}^{(1)}), \tag{3.2}$$

$$\mathbf{x}^{(0)} \leftarrow \mathbf{x}^{(0)} + \mathcal{N}(P_0(\mathbf{x}^{(1)} - R_0 \mathbf{x}^{(0)})), \tag{3.3}$$

$$\mathbf{x}_{l+1} = \text{CNN-block}(\mathbf{x}^{(0)}, s_g). \tag{3.4}$$

We first down-sample the channel dimension of the input feature maps $\mathbf{x}^{(0)}$ in (3.1) by a factor of 2, using a restriction operator R_0. This operation creates the coarse feature maps $\mathbf{x}^{(1)}$, which have the same spatial resolution as $\mathbf{x}^{(0)}$, but half the channels. The operator R_0 is implemented by a grouped 1×1 convolution; see a detailed discussion in section 3.3. Then, in (3.2) a nongrouped CNN block is applied on the coarse feature maps $\mathbf{x}^{(1)}$. This block couples all channels but involves only $c_{\text{in}}^2/4$ parameters instead of c_{in}^2. Following that, in (3.3) we use a prolongation operator P_0 to up-sample the residual $\mathbf{x}^{(1)} - R_0 \mathbf{x}^{(0)}$[1] from the coarse level to the fine level (up-sampling in channel space) and obtain a tensor with c_{in} channels. Adding the up-sampled residual is common in nonlinear multigrid schemes—we elaborate on this point below. Finally, in (3.4) we perform a grouped CNN block, which is of significantly lower computational cost than its nongrouped counterpart for a small s_g. An illustration of this architecture using three levels is presented in Figure 2. The multilevel block (Algorithm 1) is applied by iteratively reducing the channel dimensionality until reaching the coarsest grid size s_c. Hence, the number of levels at each layer is given by $n_{\text{levels}} = \lfloor \log_2(\frac{c_{\text{in}}}{s_c}) \rfloor$, and the architecture uses more levels as the channel space widens. By choosing $s_g = s_c$ we have grouped convolutions on the fine and intermediate levels and a fully connected layer only on the coarsest level,

[1] In the multigrid literature, this term is called a coarse grid correction, because it corrects the fine level solution using an interpolated coarse error approximation. Here, there is no iterative solution, as we just propagate feature maps through the network's layer. Hence, we use the term *residual* that is common in deep learning literature as the addition to the feature maps in a layer.

natural configuration that is also illustrated in Fig. 2. Specifically, in our experiments we choose $s_g = s_c = 2$, which can serve as default choices.

By letting the information propagate through the multigrid levels sequentially, we increase the number of convolutions and nonlinear activations compared to the original CNN block. Thereby, we aim to increase the expressiveness of our MGIC block, given a computational budget.

Algorithm 1: Multigrid-in-channels block.

Algorithm 1: Multigrid-in-channels block.

$\mathbf{x}_{l+1} = \textbf{MGIC-block}(\mathbf{x}_l, \text{CNN-block}, s_g, s_c)$.
 # Inputs:
$\mathbf{x}_l$ *- input feature maps with* c_{in} *channels.*
s_g*: group size.* s_c*: coarsest grid size.*
 # CNN-block: A reference CNN block, e.g., Eq. (2.2)
$\mathbf{x}^{(0)} = \mathbf{x}_l$
$n_{\text{levels}} = \left\lfloor \log_2(\frac{c_{\text{in}}}{s_c}) \right\rfloor$
 # Going down the levels, starting from $\mathbf{x}_l$
for $j = 0 : n_{\text{levels}}$ **do**
  $\mathbf{x}^{(j+1)} = R_j \mathbf{x}^{(j)}$
end
 # On the coarsest level we perform a non-grouped CNN-block:
$\mathbf{x}^{(n_{\text{levels}})} \leftarrow \text{CNN-block}(\mathbf{x}^{(n_{\text{levels}})})$
 # Going up the levels:
for $j = n_{\text{levels}} - 1 : 0$ **do**
  $\mathbf{x}^{(j)} \leftarrow \mathbf{x}^{(j)} + \mathcal{N}\left(P_j(\mathbf{x}^{(j+1)} - R_j \mathbf{x}^{(j)})\right)$
  $\mathbf{x}^{(j)} \leftarrow \text{CNN-block}(\mathbf{x}^{(j)}, \text{group\_size} = s_g)$
end
return $\mathbf{x}_{l+1} = \mathbf{x}^{(0)}$.

3.3. The choice of transfer operators P_j and R_j. The transfer operators play an important role in multigrid methods. In classical methods, the restriction R_j maps the fine level state of the iterative solution onto the coarse grid, and the prolongation P_j acts in the opposite direction, interpolating the coarse solution back to the fine grid. Clearly, in the coarsening process we lose information, since we reduce the dimension of the problem and the state of the iterate. The key idea is to design P_j and R_j such that the coarse problem captures the subspace that is causing the fine grid process to be inefficient. This results in two complementary processes: the fine level steps (dubbed as *relaxations* in multigrid literature) and the coarse grid correction.

To keep the computations low, at the jth level we choose R_j to be a grouped 1×1 convolution that halves the number of channels of its operand. We choose P_j to have the transposed structure of R_j. For R_0 and P_0 we choose the same number of groups as in the CNN block, e.g., for R_0 it will be $\frac{c_{\text{in}}}{s_g}$ groups. The operators R_j take groups of channels of size s_g and using a 1×1 convolution distills the information to $\frac{s_g}{2}$ channels. Similarly, the operators P_j interpolate the coarse channels back to the higher channel dimension using grouped 1×1 convolutions. This choice for the transfer operators corresponds to aggregation-based multigrid coarsening [63], where aggregates (groups) of variables are averaged to form a coarse grid variable. In section 4.1.1, we exemplify that the transfer operators preserve essential information of all channels.

The weights of the transfer operators are learned as part of the optimization and are initialized by positive weights with row-sums of 1. This initialization ensures that feature maps do not vanish as we multiply them by consecutive restrictions R_j to start the MGIC block (in the first loop of Algorithm 1) at the beginning of the training process. From this initialization we optimize the weights during training. This is also motivated by similar works on trained optimization algorithms [33] and their success in imaging applications [24].

The importance of up-sampled residuals. Adding the up-sampled residual in (3.3) is often called a τ-correction in multigrid literature, and it is the standard way to apply multigrid methods to solve nonlinear problems [6]. Here, it allows us to have a skip connection between corresponding levels of the multigrid cycle, introducing an identity mapping in (3.3) as guided by [30]. Furthermore, at each level j we up-sample only the difference between feature maps $\mathbf{x}^{(j+1)} - R_j\mathbf{x}^{(j)}$ and not the feature map $\mathbf{x}^{(j+1)}$ itself. This helps prevent feature maps summation when adding the up-sampled residual, which may lead to exploding gradients. By this definition, if the CNN block has an identity mapping, then so does the whole MGIC block in Algorithm 1.

Changing the channel resolution blocks. The structure of the MGIC block as in Figure 2 is more natural to equal input and output channel sizes, i.e., $c_{\text{in}} = c_{\text{out}}$. Hence, when we wish to change the number of channels, we define a lightweight shortcut that is designed to transform a tensor from c_{in} to c_{out} such that our MGIC blocks will be given an input where $c_{\text{in}} = c_{\text{out}}$. Specifically, to obtain low computational cost, we use a depthwise 3×3 convolution, although other alternatives such as a 1×1 convolution are also possible. In case we wish to change the spatial dimensions of the input tensor, we perform the same operation, only with a stride of 2.

3.4. The complexity of the MGIC block. Consider a case where we have $c = c_{\text{in}} = c_{\text{out}}$ channels in the network, and we apply a standard convolution layer using $d \times d$ convolution kernels (e.g., a 3×3 kernel). The output consists of c feature maps, where each one is a sum of the c input maps, each convolved with a kernel. Hence, such a convolution layer requires $\mathcal{O}(c^2 \cdot d^2)$ parameters, inducing a quadratic growth in the parameters and FLOPs.

Relaxation cost per level. On each level of an MGIC block, a relaxation step is performed. At the jth level, this relaxation step is realized by a grouped convolution of kernel size $d \times d$, with a group size of s_g that divides $\frac{c}{2^j}$ (since at each level we halve the channels space, starting from c channels), yielding $\frac{c}{s_g \cdot 2^j}$ groups. Therefore, the number of parameters required for such relaxation step is $\frac{s_g \cdot c \cdot d^2}{2^j}$. At the coarsest level, we have s_c channels which perform a fully coupled relaxation step, requiring $s_c \cdot d^2$ parameters.

The cost of restriction and prolongation. As discussed in section 3.3, the restriction and prolongation operators are implemented via grouped 1×1 convolutions, halving and doubling the feature space dimension, respectively. Those operators are learned at each level of our MGIC block. Therefore, the number of parameters for those operators at the jth level is $\frac{c}{2^j}$. The analysis here is similar to the case of the relaxation steps, only here $d = 1$, and we have no fully coupled operators on the coarsest level.

The total cost of an MGIC block. Combining the analysis from the paragraphs above, the total number of parameters for an MGIC block with n levels is as follows:

$$\sum_{j=0}^{n-1} \left(\frac{s_g \cdot c \cdot (d^2+1)}{2^j} \right) + s_c^2 \cdot d^2 < 2\left(s_g \cdot c \cdot (d^2+1)\right) + s_c^2 \cdot d^2. \tag{3.5}$$

If s_c is small (typically, we choose $s_c = s_g$), we can neglect the term $s_c^2 \cdot d^2$ to obtain $\mathcal{O}(s_g \cdot c \cdot (d^2+1))$ parameters. Therefore, since s_g and s_c are fixed and small, and the spatial dimension of the learned relaxation step convolution kernel size d is typically of small size (3, 5, or 7), our method scales linearly with respect to the network's width. This will be most beneficial if c is large, which is typical and usually required in order to obtain state-of-the-art performance on various tasks [7, 30] as discussed in section 1.

Memory footprint. During training, the memory footprint of MGIC is roughly twice as large as that of a single CNN block since all the maps in the hierarchy are saved for backpropagation. However, the main motivation for using lightweight networks is increasing the efficiency of the inference application. This is beneficial when the trained models are applied over and over again for inference, possibly on edge devices with fewer computational resources, like autonomous cars and drones. This typically offsets the increased cost of training, which is typically done once or a few times. Indeed, during inference the coarser feature maps are released while going up the hierarchy, so when applying the upmost CNN block, the memory footprint is identical to a single block. Following the complexity analysis above, all the feature maps $\{\mathbf{x}^{(l)}\}$ require about $\times 2$ the memory of $\mathbf{x}^{(0)}$, but the memory footprint of a CNN block can be higher than that. For example, the MobileNets [34] involve an inverse bottleneck with an expansion of $4-6$, rendering it more expensive in terms of peak-memory as the additional MGIC overhead.

4. Experiments. In this section, we report several experiments with our MGIC approach. We start with two proof-of-concept experiments, measuring how well our MGIC can compress the channel space, as well as its capacity to approximate implicit functions. Then, we test our method on image classification and segmentation and point cloud classification benchmarks. Our goal is to compare how different architectures perform using a relatively small number of parameters, aiming to achieve similar or better results with fewer parameters and FLOPs. We train all our models using an NVIDIA Titan RTX and implement our code using PyTorch software [55]. Our code is available at github.com/BGUCompSci/MultigridInChannelsCNNs.

4.1. Coarse channels representation. Our method aims to reduce parameters by introducing a hierarchical representation of the network's channels, and traverse this hierarchy in the forward pass. In this experiment we wish to quantify the effectiveness of our channel down- and up-sampling mechanism. Specifically, we wish to measure how well we can encode feature maps on the coarsest grid, where fully coupled convolutions are applied, using the transfer operators R and P only. To do so, we sample 1,024 images from ImageNet, and extract their feature maps from the first convolutional layer of a pretrained ResNet-50, containing 64 channels. Then, we encode and decode the feature maps using the restriction and prolongation operators, respectively. To study the transfer operators in isolation, we remove the CNN blocks and long skip connections in Figure 2 from the MGIC block. This experiment disentangles the concept of coarse channel representation used in MGIC from the actual CNN block that is used in the other experiments that we show in this paper. We experiment with several values of the group size parameter s_g and present the mean squared error (MSE) of the feature maps reconstruction in Table 1. The original feature maps and their reconstructions are shown in Figure 3. According to this experiment, our method is capable of faithfully representing the original channel space (obtaining low MSE values), even when it operates on a low-parameters and FLOPs budget (the values of s_g and s_c are small compared to 64 channels). Obviously, as we

TABLE 1
Feature maps reconstruction MSE versus s_g. s_c is fixed to 8. The numbers below are rounded.

s_g	32	16	8	4
MSE	0.011	0.013	0.017	0.024
Parameters	3,300	1,800	900	450

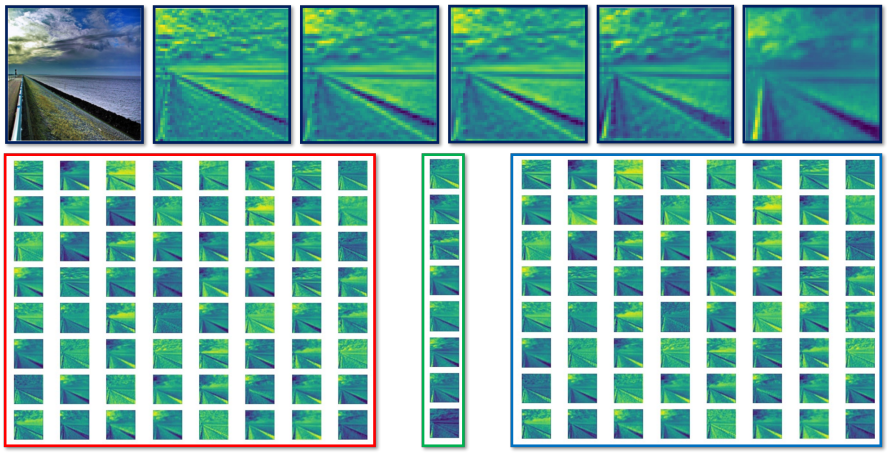

FIG. 3. *From left to right: Top—input image, one of its feature maps, and its reconstructions with $s_g = 32, 16, 8, 4$, respectively. s_c is kept at 8. Bottom—input feature maps, their coarsest grid representation, and their reconstruction from our MGIC block with down- and up-sampling only.*

use a larger group size, the approximation improves, but not dramatically—it is only a factor of about $2\times$ in the MSE between $s_g = 32$ and $s_g = 4$.

4.2. Performance in function approximation. In this experiment we demonstrate the efficacy in function approximation in a supervised learning setup. This property is important, especially in applications where we wish to model implicit functions via a neural networks, such as signed-distance fields for shape reconstruction and completion [54], and solution of PDEs [2, 47, 53, 57]—these works in particular approximate functions in an unsupervised manner. Here, we wish to approximate a family of functions

$$f(x, y, a, b, c) = a\cos(bx)\sin(cy), \tag{4.1}$$

where $x, y \in [0, 2\pi]\, a \in [0, 1]\, b \in [1, 2]\, c \in [10, 20]$. We use three methods—MobileNetV3 [34], GhostNet [25], and our MGIC-MobileNetV3. Since the input is a vector $[x, y, a, b, c] \in \mathbb{R}^5$ (treated as a spatial domain of size 1), the convolution kernels are effectively 1×1 convolutions. (Note that in any case these are the dominant operations in all the networks and are the driving force of neural network in general.) Throughout all the experiments, we used the network described in Table 2, where CNN block is replaced with the respective method. The architecture takes a hyperparameter c_{max}, which dictates the maximal width of the network. c_{max} is assumed to be a power of 2 and larger than 32. Furthermore, we use two CNN blocks for each width. For instance, when $c_{max} = 128$, there are a total of six CNN blocks of the following widths: $\{32, 32, 64, 64, 128, 128\}$ (these are the c_{out} values). The settings of these experiments are as follows: we randomly sample 10 million points of the functions defined by f in (4.1), where 95% of the points are used for training and the remaining 5% for testing,

TABLE 2

Network architecture used in the function approximation experiment, section 4.2.– denotes a nonapplicable parameter. BN denotes a batch-normalization operator. For CNN block, we consider and compare MobileNetV3, GhostNet, and our MGIC-MobileNetV3. c_{max} is the maximal number of channels (e.g., 128).

c_{in}	Operations	Expansion	c_{out}
5	1×1 Conv, BN, ReLU	–	16
16	CNN block	4	32
32	CNN block	4	32
32	CNN block	4	64
	⋮		
c_{max}	CNN block	4	c_{max}
c_{max}	CNN block	4	c_{max}
c_{max}	1×1 Conv, BN, ReLU	–	64
64	1×1 Conv	–	1

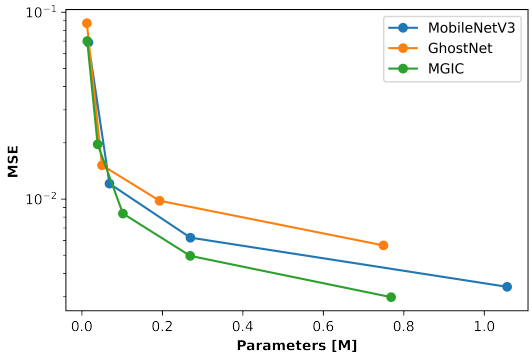

FIG. 4. *Approximation error of the function family from (4.1) with MobileNetV3, GhostNet, and our MGIC. Metric is in MSE as function of number of parameters.*

with a 10-fold cross-validation. We train each network for 1,000 epochs with a batch size of 20,000 points, using the SGD optimizer with a constant learning rate of 0.0001. The loss function is the MSE. The results, summarized in Figure 4, show that MGIC yields lower MSE with fewer parameters than MobileNetV3 and GhostNet, suggesting that at least for the purpose of approximating such functions, our MGIC architecture has a higher model capacity per number of parameters.

4.3. Image classification. We compare our approach with a variety of popular and recent networks like ResNet-50 [30], MobileNetV3, [34] and GhostNet [25] for image classification on the CIFAR10 and ImageNet datasets. We use an SGD optimizer with a mini-batch size of 256 for ImageNet and 128 for CIFAR-10, both for 100 epochs. Our loss function is cross-entropy. The initial learning rates for CIFAR-10 and ImageNet are 0.001 and 0.1, respectively. We divide them by 10 every 30 epochs. The weight decay is 0.0001, and the momentum is 0.9. As data augmentation, for both datasets, we use standard random horizontal flipping and crops, as in [30].

4.3.1. CIFAR-10. The CIFAR-10 dataset [41] consists of 60K natural images of size 32×32 with labels assigning each image into one of 10 categories. The data is split into 50K training and 10K test sets. Here, we use a ResNet-56 [30] architecture,

which includes ResNet layers of widths 16, 32, and 64 channels. Together with our MGIC block, with parameters $s_g = 8$, $s_c = 16$, MGIC-ResNet-56 has up to three levels. We compare our method with other recent and popular architectures such as AMC-ResNet-56 [31] and Ghost-ResNet-56 [25], and our baseline is the original ResNet-56. We report our results in Table 3, where we see large improvement over existing methods, while retaining low number of parameters and FLOPs.

4.3.2. ImageNet. The ImageNet [17] challenge ILSVRC 2012 consists of over 1.28M training images and 50K validation images from 1,000 categories. We resize the images to 224×224 [30]. We perform two experiments.

ResNet-50 compression. We compress the ResNet-50 architecture and compare our MGIC approach with other methods. As the goal of this experiment is to compress a standard ResNet-50 [30], we follow the exact architecture of the latter, which includes input feature maps of widths of 256, 512, 1,024, and 2,048 channels. We only replace each ResNet block layer by an MGIC-ResNet block, denoted as MGIC$(\cdot)$, as depicted in Table 4. The results are reported in Table 5, where we propose three

TABLE 3
Comparison of state-of-the-art methods for compressing ResNet-56 on CIFAR-10. - indicates unavailable results.

Architecture	Params [$\times 10^6$]	FLOPs [$\times 10^6$]	Test acc.
ResNet-56 [30]	0.85	125	93.0%
CP-ResNet-56 [32]	–	63	92.0%
ℓ_1 -ResNet-56 [44]	0.73	91	92.5%
AMC-ResNet-56 [31]	–	63	91.9%
Ghost-ResNet-56 [25]	0.43	63	92.7%
MGIC-ResNet-56 (ours)	0.41	60	**94.2%**

TABLE 4
MGIC-ResNet50 architecture. MGIC-ResNet (X) is the MGIC block applied with the ResNet bottleneck three-convolution block in (2.2)*, corresponding to the convolution kernel sizes in X as presented in* [30] *(i.e., left numbers correspond to either 1×1 or 3×3 filters of the convolutions, and the right numbers correspond to the output channels of each convolution). Conv2D is a 2D convolution layer followed by a BatchNorm operation and a ReLU nonlinear activation. # Rep is the number of block repetitions. c_{out} denotes the number of output channels. A maxpool operation occurs after every convolution and MGIC layer.*

Input	Layer	# Rep	c_{out}
$224^2 \times 3$	Conv2D 7×7	1	64
$112^2 \times 64$	Conv2D 3×3	1	64
$112^2 \times 64$	MGIC-ResNet $\left(\begin{bmatrix} 1\times 1\,64 \\ 3\times 3\,64 \\ 1\times 1\,256 \end{bmatrix}\right)$	3	256
$56^2 \times 256$	MGIC-ResNet $\left(\begin{bmatrix} 1\times 1\,128 \\ 3\times 3\,128 \\ 1\times 1\,512 \end{bmatrix}\right)$	4	512
$28^2 \times 512$	MGIC-ResNet $\left(\begin{bmatrix} 1\times 1\,256 \\ 3\times 3\,256 \\ 1\times 1\,1024 \end{bmatrix}\right)$	6	1024
$14^2 \times 1024$	MGIC-ResNet $\left(\begin{bmatrix} 1\times 1\,512 \\ 3\times 3\,512 \\ 1\times 1\,2048 \end{bmatrix}\right)$	3	2048
$7^2 \times 2048$	AvgPool2D 7×7	1	2048
$1^2 \times 2048$	FC	1	1000

TABLE 5
Comparison of state-of-the-art methods for compressing ResNet-50 on ImageNet dataset.

Model	Params [$\times 10^6$]	FLOPs [$\times 10^9$]	Top-1 Acc.%	Top-5 Acc.%
ResNet-50 [30]	25.6	4.1	75.3	92.2
Thinet-ResNet-50 [48]	16.9	2.6	72.1	90.3
NISP-ResNet-50-B [72]	14.4	2.3	–	90.8
Versatile-ResNet-50 [66]	11.0	3.0	74.5	91.8
SSS-ResNet-50 [37]	–	2.8	74.2	91.9
Ghost-ResNet-50 [25]	13.0	2.2	75.0	92.3
MGIC-ResNet-50 ($s_g = 32,\ s_c = 64$) (ours)	9.4	1.6	75.8	92.9
MGIC-ResNet-50 ($s_g = 64,\ s_c = 64$) (ours)	15.1	2.5	**77.9**	**93.7**
Shift-ResNet-50 [68]	6.0	–	70.6	90.1
Taylor-FO-BN-ResNet-50 [50]	7.9	1.3	71.7	–
Slimmable-ResNet-50 0.5× [71]	6.9	1.1	72.1	–
MetaPruning-ResNet-50 [45]	–	1.0	73.4	–
Ghost-ResNet-50 (s = 4) [25]	6.5	1.2	74.1	91.9
MGIC-ResNet-50 ($s_g = 16,\ s_c = 64$) (ours)	6.2	1.0	**74.3**	**92.0**

variants of our MGIC-ResNet-50 network, differing in the s_g parameter. For s_c we chose 64, which lead to three to five levels in our MGIC blocks throughout the network. Our results outperform the rest of the considered methods, and our network with $s_g = 64,\ s_c = 64$ also outperforms ResNeXt-50 [70] (25.0M parameters, 4.2B FLOPs, 77.8% top-1 accuracy), which is not shown in the table because the ResNeXt architecture utilizes more channels than ResNet-50 and therefore is not directly comparable.

ImageNet classification on a budget of FLOPs. In this experiment we compare our approach with recent light networks. In particular, we follow the MobileNetV3-Large [34] architecture for its efficiency and high accuracy, and replace the standard MobileNetV3 block with our MGIC block. Our building blocks are MGIC bottlenecks (dubbed MGIC-bneck). That is, we build a MGIC version of the bottleneck from MobileNetV3. Our MGIC-MobileNetV3 is given in Table 6. Note that this is the ×1.0 version and can be modified via the width multiplier α. Our parameter s_g controls the group size—therefore it determines the number of groups in each MGIC bottleneck. We denote the number of output channels by c_{out} and the number of hidden channels within a block (the dimension of the square operator K_{l_2} in the ResNet block (2.2)), also referred to as the expansion size, by $\#exp$. In case c_{out} and $\#exp$ are not divisible by s_g, we set s_g to the closest (smaller) integer to its intended value such that it divides them. For example, in our experiments we set $s_g = 64$, and for the network defined in Table 6, the fifth MGIC-bneck layer has $\#exp = 120$ and $c_{out} = 40$, meaning they do not divide by 64. Therefore we modify s_g to be the largest integer that is smaller than 64 and divides both $\#exp$ and c_{out}, giving $s_g = 40$ in this example. Our experiment is divided into three scales—small, medium, and large—where we scale our networks with width factors of $0.6, 1.0$, and 1.2, respectively. We find that our method obtains higher accuracy, with a similar number of FLOPs, as depicted from the results in Table 7 and Figure 5. Specifically, we compare our methods with and without the use of the h-swish activation function [34], where we see similar results. Compared to other popular and recent methods like MobileNetV3, GhostNet, and ShuffleNetV2, we obtain better accuracy given the same FLOPs.

Inference and training times. We measure the single thread inference times on one 224×224 image using lightweight models on a Samsung Galaxy S8 mobile

TABLE 6

MGIC-MobileNetV3 architecture. MGIC-bneck denotes a MGIC-bottleneck. The bottleneck is the same as in MobileNetV3, only in an MGIC form. Conv2D is a 2D convolution layer followed by a BatchNorm operation and a ReLU nonlinear activation. # exp denotes the expansion size. c_{out} denotes the number of output channels. SE stands for Squeeze-Excite. Pool denotes a maxpool operation, reducing the spatial size of the input. - denotes a nonapplicable option. ✓ and × denote true and false, respectively.

Input	Operation	# exp	c_{out}	SE	Pool
$224^2 \times 3$	Conv2D 3×3	16	–	×	✓
$112^2 \times 16$	MGIC-bneck	16	16	×	×
$112^2 \times 16$	MGIC-bneck	48	24	×	✓
$56^2 \times 24$	MGIC-bneck	72	24	×	×
$56^2 \times 24$	MGIC-bneck	72	40	✓	✓
$28^2 \times 40$	MGIC-bneck	120	40	✓	×
$28^2 \times 40$	MGIC-bneck	240	80	×	✓
$14^2 \times 80$	MGIC-bneck	200	80	×	×
$14^2 \times 80$	MGIC-bneck	184	80	×	×
$14^2 \times 80$	MGIC-bneck	184	80	×	×
$14^2 \times 80$	MGIC-bneck	480	112	✓	×
$14^2 \times 112$	MGIC-bneck	672	112	✓	×
$14^2 \times 112$	MGIC-bneck	672	160	✓	✓
$7^2 \times 160$	MGIC-bneck	960	160	×	×
$7^2 \times 160$	MGIC-bneck	960	160	✓	×
$7^2 \times 160$	MGIC-bneck	960	160	×	×
$7^2 \times 160$	MGIC-bneck	960	160	✓	×
$7^2 \times 160$	Conv2D 1×1	–	960	×	×
$7^2 \times 960$	AvgPool 7×7	–	960	×	×
$1^2 \times 960$	Conv2D 1×1	–	1280	×	×
$1^2 \times 1280$	FC	–	1000	×	×

device (using the TFLite tool [1]), and an Intel i9-9820X CPU—see Table 8 (averaged over 50 inferences). We observe that at least by these timings, the runtime of MGIC is on par with the considered architectures while obtaining higher accuracy. Additionally, we report the training time on a mini-batch of 128, 224×224 sized images from ImageNet, on an Nvidia Titan RTX GPU. Our MGIC requires slightly higher training time due to its higher complexity and multilevel channels structure. However, it yields higher accuracy compared to the considered methods.

4.4. Image semantic segmentation. We compare our method with MobileNetV3 on semantic segmentation on the Cityscapes [15] dataset. For the encoder part of the network, we build large and small variants, based on MobileNetV3-Large and MobileNetV3-Small, described in Tables 1 and 2 in [34], respectively. We also utilize the same LR-ASPP segmentation head and follow the observations from [34]. Namely, we reduce the number of channels in the last block of our networks by a factor of two and use 128 filters in the segmentation head. For training, we use the same data augmentation and optimization approach as in [12]. The results are shown in Table 9, where we report the mean intersection over union (mIoU) metric of our MGIC-Large with $s_g = 64$ and $s_g = 32$. We note that the results for the former are slightly better than those of MobileNetV3, while the performance of the latter are more favorble as they offer similar accuracy for less FLOPs and parameters. In addition, we read similar accuracy when using our MGIC-Small with $s_g = 64$.

4.5. Point cloud classification. The previous experiments were performed on structured CNNs, i.e., on 2D images. To further validate our method's generaliza-

TABLE 7
Comparison of state-of-the-art lightweight networks on ImageNet dataset classification.

Model	Params [$\times 10^6$]	FLOPs [$\times 10^6$]	Top-1 Acc.%	Top-5 Acc.%
ShuffleNetV1 0.5× (g = 8) [74]	1.0	40	58.8	81.0
MobileNetV2 0.35× [58]	1.7	59	60.3	82.9
ShuffleNetV2 0.5× [49]	1.4	41	61.1	82.6
MobileNeXt 0.35× [75]	1.8	80	64.7	–
MobileNetV3-Small 0.75× [34]	2.4	44	65.4	–
GhostNet 0.5× [25]	2.6	42	66.2	86.6
MGIC-MobileNetV3 0.6× (ours)	2.3	48	**67.0**	**87.3**
MGIC-MobileNetV3 0.6× (ours) no h-swish	2.3	45	**66.8**	**86.9**
MobileNetV1 0.5× [35]	1.3	150	63.3	84.9
MobileNetV2 0.6× [58]	2.2	141	66.7	–
ShuffleNetV1 1.0× (g = 3) [74]	1.9	138	67.8	87.7
ShuffleNetV2 1.0× [49]	2.3	146	69.4	88.9
MobileNeXt 0.75× [75]	2.5	210	72.0	–
MobileNetV3-Large 0.75× [34]	4.0	155	73.3	–
GhostNet 1.0× [25]	5.2	141	73.9	91.4
MGIC-MobileNetV3 1.0× (ours)	5.2	145	**74.8**	**92.0**
MGIC-MobileNetV3 1.0× (ours) no h-swish	5.2	138	**74.3**	**91.6**
MobileNetV2 1.0× [58]	3.5	300	71.8	91.0
ShuffleNetV2 1.5× [49]	3.5	299	72.6	90.6
FE-Net 1.0× [13]	3.7	301	72.9	–
FBNet-B [67]	4.5	295	74.1	–
ProxylessNAS [8]	4.1	320	74.6	92.2
MnasNet-A1 [60]	3.9	312	75.2	92.5
MobileNeXt 1.0× [75]	3.4	300	74.0	–
MobileNetV3-Large [34] 1.0×	5.4	219	75.2	–
GhostNet 1.3× [25]	7.3	226	75.7	92.7
MGIC-MobileNetV3 1.2× (ours)	7.1	233	**76.1**	**93.2**
MGIC-MobileNetV3 1.2× (ours) no h-swish	7.1	217	**76.2**	**93.4**

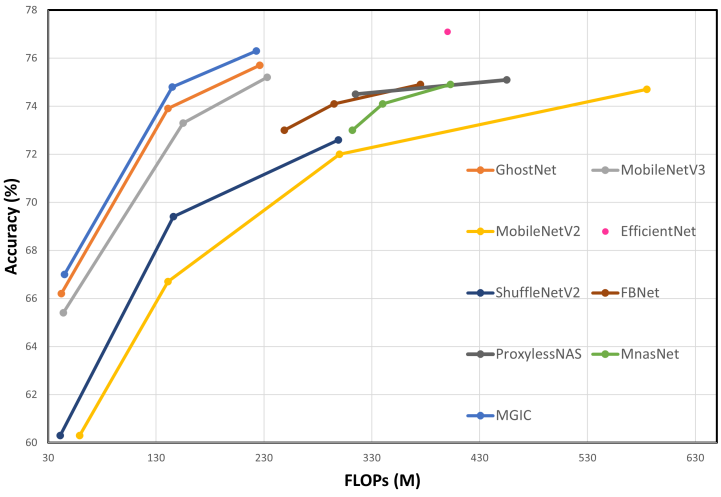

FIG. 5. *Top-1 accuracy versus FLOPs on ImageNet dataset.*

TABLE 8
Inference runtime of state-of-the-art small networks on a Samsung Galaxy S8 mobile device and a PC CPU and training time on an Nvidia Titan RTX GPU.

Metric	MobileNetV2 1.0x	MobileNetV3 0.75x	GhostNet 1.0x	MGIC MobileNetV3 1.0x
Accuracy [%]	71.8	73.3	73.9	**74.8**
Mobile inference [ms]	795	**418**	487	480
CPU inference [ms]	310	**140**	170	172
GPU training [s]	0.262	**0.215**	0.237	0.311

TABLE 9
Segmentation results on Cityscapes dataset. Metric is in mean intersection over union.

Backbone	Params [$\times 10^6$]	FLOPs [$\times 10^9$]	mIoU %
MobileNetV3-Small	0.47	2.90	68.38
MGIC-Small $s_g = 64$ (ours)	0.48	2.73	**68.52**
MobileNetV3-Large	1.51	9.74	72.64
MGIC-Large $s_g = 32$ (ours)	1.32	8.87	71.02
MGIC-Large $s_g = 64$ (ours)	1.67	9.62	**72.69**

TABLE 10
Graph neural network for point cloud classification. G-conv is a graph convolution layer. MLP is a multilayer perceptron. MaxPool is a global max-pooling layer. c_{out} denotes the number of output channels. $\times$ denotes the number of repetitions of the respective layer.

Input	Operation	c_{out}
1024×3	G-conv	64
1024×64	$2 \times$ G-conv	64
1024×64	MLP	64
1024×64	$3 \times$ G-conv	64
1024×64	MLP	64
1024×64	MaxPool	64
1×64	$3 \times$ MLP	64
1×64	FC	10

tion and usefulness, we incorporate it in graph convolutional networks (GCNs) to perform point cloud classification. Specifically, we use a smaller version of the architecture from [65], where we alter the width of the last three classifier layers from $1024, 512, 256$ to 64 in all of them. We define the architecture in Table 10, where G-conv denotes a graph convolution layer, according to the methods listed in Table 11, followed by a BatchNorm operation and a ReLU nonlinear activation. MLP is realized by a simple 1×1 convolution followed by a BatchNorm operation and a ReLU nonlinear activation. FC is a fully connected layer. In all networks, we define the adjacency matrix using the k-NN algorithm with $k = 10$. Then, we replace the GCN block with each of the backbones listed in Table 11, where we also report their performance on point-cloud classification on ModelNet-10 [69] benchmark where we sample 1,024 points from each shape.

4.6. Ablation study. *Parameter study.* To determine the impact of the hyperparameters s_g and s_c, we experiment on CIFAR-10 and ImageNet datasets for image classification. On CIFAR-10, we first fix s_g to 16 and observe how the number of parameters, number of FLOPs, and accuracy of MGIC-ResNet-56 change. Second, we fix s_c to 16, while modifying s_g, and examine our model's behavior, as reported

TABLE 11
ModelNet-10 classification.

Backbone	Params [$\times 10^6$]	FLOPs [$\times 10^6$]	Accuracy %
DGCNN [65]	0.16	125	91.6
diffGCN [18]	0.57	64	92.5
MGIC-diffGCN (ours)	**0.11**	**13.7**	**92.9**

TABLE 12
Influence of s_c and s_g in our MGIC framework on ResNet-56 architecture and CIFAR-10 dataset. s_g is fixed to 16.

s_c	s_g	Params [$\times 10^6$]	FLOPs [$\times 10^6$]	Accuracy %
64	16	0.53	91	94.7
32	16	0.5	76	94.6
16	16	0.47	65	94.3
16	32	0.79	100	94.8
16	16	0.53	85	94.7
16	8	0.41	60	94.2
16	4	0.29	45	92.8

TABLE 13
Hyperparameter study on ImageNet.

s_c	s_g	Params [$\times 10^6$]	Accuracy %
64	2	4.2	70.2
64	4	4.2	70.6
64	8	4.3	71.6
64	16	4.5	71.9
64	32	4.8	73.2
64	64	5.2	74.8
2	2	4.2	69.9
4	4	4.2	70.5
8	8	4.3	70.9
16	16	4.5	71.6
32	32	4.9	72.7

in Tables 12 and 13. On ImageNet, we examine two types of configurations using our MGIC-MobileNetV3 1.0x. In the first, we get $s_c = 64$ and experiment with different values of s_g. This experiment reveals the significance of the group size. Namely, it shows that as the group size grows, better accuracy is obtained (since it induces an increased channels connectivity), at the cost of more parameters. The second type of experiments considers various values of $s_c = s_g$, from 2 to 64. Like in the former, it can be concluded that increasing the coarsest grid size and the channel connectivity yields higher accuracy. Our conclusion from the results reported in Tables 12 and 13 is that a growth of s_g or s_c yields better accuracy at the cost of more parameters and FLOPs, since an increased communication between the channels is allowed. However, by reducing s_g and s_c, we obtain almost optimal accuracy at dramatically reduced costs, also as depicted in the experiments in sections 4.1 and 4.2.

Training of MGIC. Another interesting aspect is whether the performance improvement of MGIC over MobileNetV3 stems from higher efficacy of MGIC (per parameter or FLOP) or an easier and better training. In the case of the latter, it is expected that training MobileNetV3 for more epochs will lead to a smaller performance gap compared to our MGIC. In Figure 6, we present the obtained accuracy

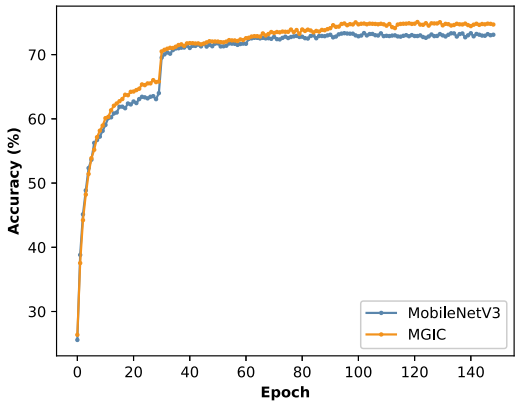

FIG. 6. *Per-epoch accuracy (%) on ImageNet classification. With MobileNetV3 the best accuracy is* 73.34% *and* 73.36% *throughout the first* 100 *and* 150 *epochs, respectively. With MGIC-MobileNetV3, the corresponding accuracy reads are* 75.02% *and* 75.08%*, respectively.*

per epoch on the ImageNet dataset with 150 epochs (as opposed to the 100-epochs results that are reported in Table 5). It can be seen from Figure 6 that adding more epochs, using the same training policy, does not improve performance for any of the methods. Therefore, it seems that given our training policy (which is identical to the one used in MobileNetV3), our MGIC shows better efficacy rather than easier or better training.

5. Conclusion. We present a novel multigrid-in-channels (MGIC) approach that improves the efficiency of convolutional neural networks (CNN) in both parameters and FLOPs, while using easy-to-implement structured grouped convolutions in the channel space. Applying MGIC, we achieve full coupling through a multilevel hierarchy of the channels, at only $\mathcal{O}(c)$ cost, unlike standard convolution layers that require $\mathcal{O}(c^2)$. This property is significant and desired to reduce both training and inference times, which also translates to a reduction in energy consumption. We also note that MGIC is most beneficial for wide networks, which are usually favored for state-of-the-art accuracy and performance. Our experiments for various tasks suggest that MGIC achieves accuracy comparable to or superior than other recent lightweight architectures at a given budget. Our MGIC block offers a universal approach for producing lightweight versions of networks suitable for different kinds of CNNs, GCNs, and traditional NNs, where fully connected layers are applied. Furthermore, it is future-ready, meaning it can also compress future architectures when available.

REFERENCES

[1] M. ABADI, P. BARHAM, J. CHEN, Z. CHEN, A. DAVIS, J. DEAN, M. DEVIN, S. GHEMAWAT, G. IRVING, M. ISARD, M. KUDLUR, J. LEVENBERG, R. MONGA, S. MOORE, D. G. MURRAY, B. STEINER, P. TUCKER, V. VASUDEVAN, P. WARDEN, M. WICKE, Y. YU, AND X. ZHENG, *Tensorflow: A system for large-scale machine learning*, in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, 2016, pp. 265–283.

[2] L. BAR AND N. SOCHEN, *Strong solutions for PDE-based tomography by unsupervised learning*, SIAM J. Imaging Sci., 14 (2021), pp. 128–155.

[3] M. BIANCHINI AND F. SCARSELLI, *On the complexity of neural network classifiers: A comparison between shallow and deep architectures*, IEEE Trans. Neural Netw. Learn. Syst., 25 (2014), pp. 1553–1565.

[4] A. Borzi and V. Schulz, *Multigrid methods for PDE optimization*, SIAM Rev., 51 (2009), pp. 361–395.
[5] A. Brandt and D. Ron, *Multigrid solvers and multilevel optimization strategies*, in Multilevel Optimization in VLSICAD, Springer, New York, 2003, pp. 1–69.
[6] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, SIAM, Philadelphia, 2000.
[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, *Language models are few-shot learners*, Adv. Neural Inf. Process. Syst., 33 (2020), pp. 1877–1901.
[8] H. Cai, L. Zhu, and S. Han, *ProxylessNAS: Direct neural architecture search on target task and hardware*, in Proceedings of the International Conference on Learning Representations, 2019.
[9] B. Chang, L. Meng, E. Haber, F. Tung, and D. Begert, *Multi-Level Residual Networks from Dynamical Systems View*, preprint, https://arxiv.org/abs/arXiv:1710.10348, 2017.
[10] S. Changpinyo, M. Sandler, and A. Zhmoginov, *The Power of Sparsity in Convolutional Neural Networks*, https://arxiv.org/abs/1702.06257, 2017.
[11] C.-F. R. Chen, Q. Fan, N. Mallinar, T. Sercu, and R. Feris, *Big-little net: An efficient multi-scale feature representation for visual and speech recognition*, in Proceedings of the International Conference on Learning Representations, 2019, https://openreview.net/forum?id=HJMHpjC9Ym.
[12] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, *Rethinking Atrous Convolution for Semantic Image Segmentation*, preprint, https://arxiv.org/abs/1706.05587, 2017.
[13] W. Chen, D. Xie, Y. Zhang, and S. Pu, *All you need is a few shifts: Designing efficient convolutional neural networks for image classification*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 7241–7250.
[14] Y. Chen, H. Fan, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, and J. Feng, *Drop an Octave: Reducing spatial redundancy in convolutional neural networks with octave convolution*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 3435–3444.
[15] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, *The cityscapes dataset for semantic urban scene understanding*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.
[16] H. De Sterck, T. A. Manteuffel, S. F. McCormick, Q. Nguyen, and J. Ruge, *Multilevel adaptive aggregation for markov chains, with application to web ranking*, SIAM J. Sci. Comput., 30 (2008), pp. 2235–2262.
[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F. F. Li, *ImageNet: A large-acale hierarchical image database*, in Proceedings of CVPR09, 2009.
[18] M. Eliasof and E. Treister, *DiffGCN: Graph convolutional networks via differential operators and algebraic multigrid pooling*, in Proceedings of Advances in Neural Information Processing Systems, 2020.
[19] J. Ephrath, M. Eliasof, L. Ruthotto, E. Haber, and E. Treister, *LeanConvNets: Low-cost yet effective convolutional neural networks*, IEEE J. Sel. Top. Signal Process., 14 (2020), pp. 894–904.
[20] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder, *Parallel time integration with multigrid*, SIAM J. Sci. Comput., 36 (2014), pp. C635–C661.
[21] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–587.
[22] S. Günther, L. Ruthotto, J. B. Schroder, E. C. Cyr, and N. R. Gauger, *Layer-parallel training of deep residual neural networks*, SIAM J. Math. Data Sci., 2 (2020), pp. 1–23.
[23] E. Haber, L. Ruthotto, E. Holtham, and S.-H. Jun, *Learning across scales—multiscale methods for convolution neural networks*, in Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018.
[24] K. Hammernik, T. Klatzer, E. Kobler, M. P. Recht, D. K. Sodickson, T. Pock, and F. Knoll, *Learning a variational network for reconstruction of accelerated MRI data*, Magn. Reson. Med., 79 (2018), pp. 3055–3071.

[25] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, *GhostNet: More features from cheap operations*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1580–1589.

[26] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, B. Catanzaro, and W. J. Dally, *DSD: Dense-sparse-dense training for deep neural networks*, in Proceedings of the International Conference on Learning Representations, 2017.

[27] S. Han, J. Pool, J. Tran, and W. J. Dally, *Learning both weights and connections for efficient neural network*, Int. J. Comput. Vis., 5 (2015), pp. 1135–1143.

[28] B. Hassibi and D. G. Stork, *Second order derivatives for network pruning: Optimal brain surgeon reconstruction*, Int. J. Comput. Vis., 5 (1992), pp. 164–171.

[29] J. He and J. Xu, *MgNet: A unified framework of multigrid and convolutional neural network*, Sci. China Math., 62 (2019), pp. 1331–1354.

[30] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[31] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, *AMC: AutoML for model compression and acceleration on mobile devices*, in Proceedings of the European Conference on Computer Vision, 2018, pp. 784–800.

[32] Y. He, X. Zhang, and J. Sun, *Channel pruning for accelerating very deep neural networks*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1389–1397.

[33] S. Hochreiter, A. Younger, and P. Conwell, *Learning to learn using gradient descent*, in Artificial Neural Networks — ICANN 2001, Springer, Berlin, 2001, pp. 87–94, https://doi.org/10.1007/3-540-44668-0_13

[34] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Lee, and H. Adam, *Searching for MobileNetv3*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 1314–1324.

[35] 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*, preprint, https://arxiv.org/abs/1704.04861.

[36] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, *GPipe: Efficient training of giant neural networks using pipeline parallelism*, in Advances in Neural Information Processing Systems, 2019, pp. 103–112.

[37] Z. Huang and N. Wang, *Data-driven sparse structure selection for deep neural networks*, in Proceedings of the European Conference on Computer Vision, 2018, pp. 304–320.

[38] T.-W. Ke, M. Maire, and S. X. Yu, *Multigrid neural architectures*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6665–6673.

[39] Y. Khoo and L. Ying, *SwitchNet: A neural network model for forward and inverse scattering problems*, SIAM J. Sci. Comput., 41 (2019), pp. A3182–A3201.

[40] A. Kopaničáková and R. Krause, *Globally Convergent Multilevel Training of Deep Residual Networks*, preprint, https://arxiv.org/abs/2107.07572, 2017.

[41] A. Krizhevsky and G. Hinton, *Learning Multiple Layers of Features from Tiny Images*, https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf, 2009.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, in Proceedings of Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[43] Y. LeCun, B. E. Boser, and J. S. Denker, *Handwritten digit recognition with a back-propagation network*, in Proceedings of Advances in Neural Information Processing Systems, 1990, pp. 396–404.

[44] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, *Pruning filters for efficient ConvNets*, in Proceedings of the International Conference on Learning Representations, 2017.

[45] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, *Metapruning: Meta learning for automatic neural network channel pruning*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 3296–3305.

[46] O. E. Livne and A. Brandt, *Lean algebraic multigrid (LAMG): Fast graph laplacian linear solver*, SIAM J. Sci. Comput., 34 (2012), pp. B499–B522.

[47] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *DeepXDE: A deep learning library for solving differential equations*, SIAM Rev., 63 (2021), pp. 208–228.

[48] J.-H. Luo, J. Wu, and W. Lin, *ThiNet: A filter level pruning method for deep neural network compression*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5058–5066.

[49] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, *ShuffleNet V2: Practical guidelines for efficient CNN architecture design*, in Proceedings of the European Conference on Computer Vision, 2018, pp. 116–131.

[50] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, *Importance estimation for neural network pruning*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 11264–11272.

[51] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, *Pruning convolutional neural networks for resource efficient transfer learning*, in Proceedings of the International Conference on Learning Representations, 2017.

[52] A. Napov and Y. Notay, *An efficient multigrid method for graph laplacian systems* II*: Robust aggregation*, SIAM J. Sci. Comput., 39 (2017), pp. S379–S403.

[53] G. Pang, L. Lu, and G. E. Karniadakis, *fPINNs: Fractional physics-informed neural networks*, SIAM J. Sci. Comput., 41 (2019), pp. A2603–A2626.

[54] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, *DeepSDF: Learning continuous signed distance functions for shape representation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 165–174.

[55] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, *Automatic differentiation in PyTorch*, in Proceedings of Advances in Neural Information Processing Systems, 2017.

[56] D. M. Pelt and J. A. Sethian, *A mixed-scale dense convolutional neural network for image analysis*, Proc. Natl. Acad. Sci. USA, 115 (2018), pp. 254–259.

[57] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Phys., 378 (2019), pp. 686–707.

[58] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *MobileNetV2: Inverted residuals and linear bottlenecks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

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

[60] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, *MnasNet: Platform-aware neural architecture search for mobile*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2820–2828.

[61] M. Tan and Q. V. Le, *EfficientNet: Rethinking model scaling for convolutional neural networks*, in Proceedings of the International Conference on Machine Learning, 2019.

[62] E. Treister, J. S. Turek, and I. Yavneh, *A multilevel framework for sparse optimization with application to inverse covariance estimation and logistic regression*, SIAM J. Sci. Comput., 38 (2016), pp. S566–S592.

[63] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

[64] H. Wang, A. Kembhavi, A. Farhadi, A. L. Yuille, and M. Rastegari, *Elastic: Improving CNNS with dynamic scaling policies*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2258–2267.

[65] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, *Dynamic graph CNN for learning on point clouds*, ACM Trans. Graph., 38 (2019), pp. 1–12.

[66] Y. Wang, C. Xu, C. Xu,C. Xu, and D. Tao, *Learning versatile filters for efficient convolutional neural networks*, in Proceedings of Advances in Neural Information Processing Systems, 2018, pp. 1608–1618.

[67] 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*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 10734–10742.

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

[69] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, *3D shapenets: A deep representation for volumetric shapes*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1912–1920.

[70] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, *Aggregated residual transformations for deep neural networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1492–1500.

[71] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, *Slimmable Neural Networks*, preprint, https://arxiv.org/abs/1812.08928, 2018.

[72] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, *NISP: Pruning networks using neuron importance score propagation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9194–9203.

[73] S. Zagoruyko and N. Komodakis, *Wide residual networks*, in Proceedings of the British Machine Vision Conference, E. R. H. Richard, C. Wilson, and W. A. P. Smith, eds., BMVA Press, Durham, UK, 2016, pp. 87.1–87.12, https://dx.doi.org/10.5244/C.30.87.

[74] X. Zhang, X. Zhou, M. Lin, and J. Sun, *ShuffleNet: An extremely efficient convolutional neural network for mobile devices*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6848–6856.

[75] D. Zhou, Q. Hou, Y. Chen, J. Feng, and S. Yan, *Rethinking bottleneck structure for efficient mobile network design*, in Proceedings of ECCV, 2020.