Building Efficient Deep Neural Networks with Unitary Group Convolutions

Ritchie Zhao Yuwei Hu Jordan Dotzel Christopher De Sa Zhiru Zhang

Cornell University
Ithaca, NY 14853, USA

{rz252, yh457, jad443}@cornell.edu, cdesa@cs.cornell.edu, zhiruz@cornell.edu

Abstract

We propose unitary group convolutions (UGConvs), a building block for CNNs which compose a group convolution with unitary transforms in feature space to learn a richer set of representations than group convolution alone. UGConvs generalize two disparate ideas in CNN architecture, channel shuffling (i.e. ShuffleNet [29]) and block-circulant networks (i.e. CirCNN [6]), and provide unifying insights that lead to a deeper understanding of each technique. We experimentally demonstrate that dense unitary transforms can outperform channel shuffling in DNN accuracy. On the other hand, different dense transforms exhibit comparable accuracy performance. Based on these observations we propose HadaNet, a UGConv network using Hadamard transforms. HadaNets achieve similar accuracy to circulant networks with lower computation complexity, and better accuracy than ShuffleNets with the same number of parameters and floating-point multiplies.

1. Introduction

Deep convolutional neural networks (CNNs) have proven extremely successful at large-scale computer vision problems. Research over the past few years has made steady progress on improving CNN accuracy [26]. Concurrently, efforts have been made to reduce the number of parameters and floating-point multiplies (fpmuls) in CNNs. One major trend in this research space is the increasing sparsity of layer connections. Early networks such as AlexNet [13] and VGG [19] exclusively utilize dense mappings, i.e. convolutional (conv) or fully-connected (FC) layers that form a weight connection between every input and every output feature. More advanced architectures such as Xception [2] and MobileNets [8] make use of *depthwise separable* convolutions, which consist of a sparse spatial mapping (depthwise convolution) and a dense cross-channel mapping (pointwise convolution). Even more recently, ShuffleNet [29] replaces the pointwise convolutions with sparse *group convolutions*, and additionally proposes a channel shuffle to allow information to flow between groups. These changes to layer structure look to remove weight connections while retaining accuracy performance.

A different line of efficient CNNs research looks to train networks with circulant or block-circulant [1] weights [1, 20, 6, 22]. An $n \times n$ circulant matrix contains only n unique elements. Moreover, every circulant matrix $\mathbf{C}$ can be diagonalized by the normalized discrete Fourier matrix $\mathbf{F}$ as follows:

$$\mathbf{C} = \mathbf{F}^{*}\mathbf{D}\mathbf{F} \tag{1}$$

giving rise to an asymptotically faster algorithm for matrix multiplication via the fast Fourier transform (FFT). By exploiting these properties of circulant weights, these works can also reduce CNN complexity and model size.

In this paper, we propose the concept of *unitary group convolution* (UGConv), defined as a building block for neural networks that combines a weight layer (most commonly a group convolution) with unitary transforms in feature space. We show that group convs with channel shuffle (ShuffleNet) and block-circulant networks (CirCNN) are specific instances of UGConvs. By unifying two different lines of work in CNN literature, we gain a deeper understanding into the basic underlying idea — that group convolutions exhibit improved learning ability when performed in a transformed feature basis. Through a series of experiments, we then investigate how different transforms and UGConv structures affect the learning performance. Specifically, our contributions are as follows:

1. We propose the concept of unitary group convolutions. We show that ShuffleNets and circulant networks, techniques from two disparate lines of research, are in fact both instances of UGConv networks. This lets us unify the conceptual insights of both works.

[1] In this paper, block-circulant, block-diagonal, etc. refers to matrices consisting of square sub-matrices which are circulant, diagonal, etc. This is different from the canonical definition of a block-diagonal matrix.

2. We evaluate how different unitary transforms affect learning performance. Our experiments show that when the weight layer is highly sparse (i.e. the number of groups is large), dense transforms outperform simple permutations.

3. We propose HadaNets, UGConv networks using the easy-to-compute Hadamard transform. HadaNets obtain similar accuracy as circulant networks at a lower computation complexity, and outperform ShuffleNets with identical parameter and fpmul counts.

2. Related Work

2.1. Depthwise Separable and Group Convolutions

In a traditional convolutional layer, each 3D filter must learn both spatial and cross-channel correlations. A depthwise separable convolution decouples this into two steps: a depthwise convolution which only performs spatial filtering, and a pointwise convolution which only learns cross-channel mappings. The idea originated in Sifre 2014 [18] and was subsequently popularized by networks like Xception [2] and MobileNets [8]. These works showed that depthwise separable convolutions can outperform traditional convolutions using fewer parameters and fpmuls.

A group convolution divides the input and output features into mutually independent groups and performs a convolution in each one. Depthwise convs are specific cases of group convs with group size 1. Group convolutions were part of the original AlexNet, but only to facilitate training on multiple GPUs [13]; they gained popularity as a building block of efficient CNNs as part of ResNeXt [25] and ShuffleNet [29]. The latter proposed channel shuffling to promote cross-channel information flow, surpassing MobileNets in accuracy and parameter efficiency.

Interleaved group convolutions [28, 24, 21] examines interleaving group convs and channels shuffles, and showed how a specific combination of width and sparsity (i.e. number of groups) can maximize accuracy. Deep Roots [10] uses group convolutions with increasing group size deeper into the network to improve numerous existing models. Distinct from these works, we study the composition of group convs with dense unitary transforms.

2.2. Circulant and Block-Circulant Networks

An n-by-n circulant matrix requires only $O(n)$ storage space and $O(n \log n)$ operations for the matrix-vector product (see Equation (1)). Circulant weights can reduce the model size and computational complexity of CNNs in a deterministic manner. Cheng et al. in 2015 applies this to achieve 18x parameter reduction on AlexNet with only 0.7% Top-1 accuracy loss [1]. Other authors proposed variations of circulant structure. Moczulski et al.'s ACDC used cosine transforms to avoid complex values that arise with DFTs and added a second channel-wise filter [16]. Sindhwani et al. studied the superset of generalized Toeplitz-like matrices [20]. These works exclusively worked on structured FC layers.

More recently, Wang et al. [6, 23] proposed to use block-circulant matrices and applied them to both FC and convolutional layers. Block-circulant structure elegantly addresses the long-standing issue of non-square weight matrices. The same authors also leveraged the butterfly structure of the DFT to construct efficient accelerators for circulant nets in dedicated hardware [6, 22]. A more recent follow-up in this line of work proposed to use permuted block-diagonal matrices in specialized hardware [4].

2.3. Random Projections and Hadamard Networks

Our study on random orthogonal and Hadamard transforms is partly inspired by the Fastfood transform [14] and its application to CNNs [27]. This work is a well-known example of using random embeddings and Hadamard transforms in machine learning.

A recent work from Devici et al. [5] used Hadamard-transformed images as CNN inputs. Their work differs significantly from ours; they applied a single 2D Hadamard on the input image to extract frequency features while we use Hadamard throughout the network for channel mixing.

3. Unitary Group Convolutions

The basic idea of a UGConv is a group convolution sandwiched between two unitary transforms in feature space. Let $\mathbf{X}$ be an M-channel input tensor to a conv/FC layer. Each channel is a 2D feature map (for a dense layer the dimensions are 1×1). Let $\mathbf{x}^{(i)}$ denote the i'th channel in $\mathbf{X}$. Similarly, let $\mathbf{Y}$ be the N-channel output tensor, and let $\mathbf{W}$ be the weight tensor consisting of $M \times N$ filters. We can now define an ordinary conv layer below:

$$\mathbf{y}^{(j)} = \sum_{i=1}^{M} \mathbf{x}^{(i)} * \mathbf{W}^{(ij)}, \qquad 1 \leq j \leq N$$

Figure 1(a) illustrates such a conv or dense layer. Note that although the figure looks like matrix multiplication, each square represents a 2D weight filter or feature map.

A group convolution is simply a collection of G disjoint convolutions (G is the number of groups). Each conv takes M/G input channels and produces N/G output channels.

$$\tilde{\mathbf{y}}^{(g,j)} = \sum_{i=1}^{M/G} \tilde{\mathbf{x}}^{(g,i)} * \tilde{\mathbf{W}}^{(g,ij)}, \quad 1 \leq j \leq N/G \qquad (2)$$

Here g denotes the group ($1 \leq g \leq G$), and we re-index $\mathbf{x}$ and $\mathbf{y}$ with two indices (*group*, *channel in group*). Figure 1(b) illustrates how non-zero weights in a group conv

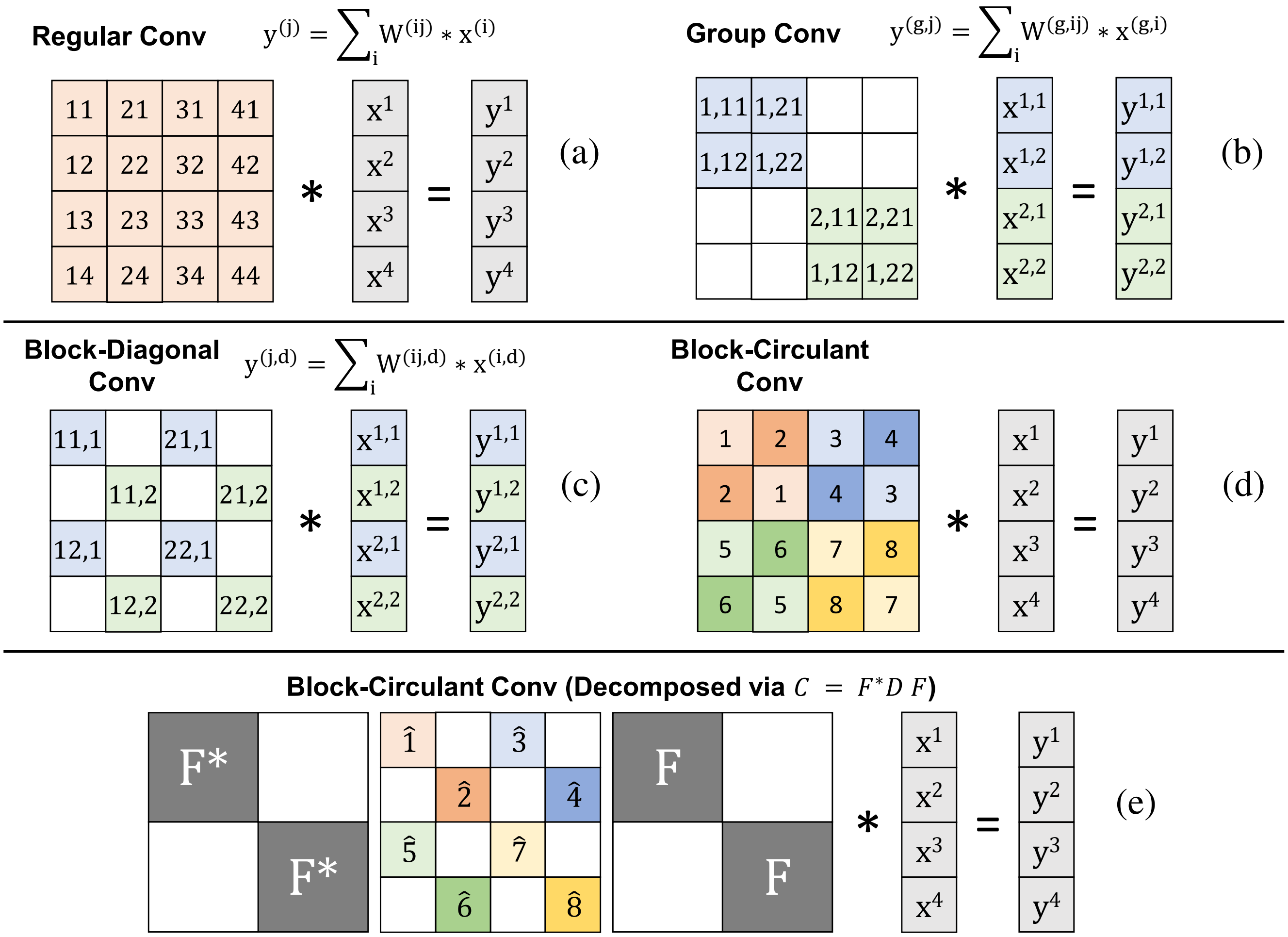

Figure 1. **Relation between group convs and circulant weights –** each square represents a 2D feature/filter, which can be 1×1 for an FC layer. (a) regular conv layer; (b) group conv with 2 groups; (c) same group conv reordered to show the block-diagonal weight structure; (d) block-circulant conv layer; (e) same block-circulant conv decomposed into block-DFTs and a block-diagonal conv layer.

form $\frac{M}{G} \times \frac{N}{G}$ blocks along the main diagonal. A group conv reduces parameter size and fpmuls by a factor of G relative to an ordinary conv. However, this is achieved by removing all weight connections between groups and negatively impacts learning behavior.

A UGConv recovers this lost learning ability by sandwiching the group conv between two cross-channel unitary transforms $\mathbf{P}$ and $\mathbf{Q}$ (Figure 2(a)). More formally, we can define a UGConv is:

$$\tilde{\mathbf{X}}_k = \mathbf{P}\mathbf{X}_k \qquad \forall k$$

$$\tilde{\mathbf{y}}^{(g,j)} = \sum_{i=1}^{M/G} \tilde{\mathbf{x}}^{(g,i)} * \tilde{\mathbf{W}}^{(g,ij)}, \quad 1 \le j \le N/G \qquad (3)$$

$$\mathbf{Y}_l = \mathbf{Q}\tilde{\mathbf{Y}}_l \qquad \forall l$$

For a tensor $\mathbf{X}$ containing M channels, $\mathbf{X}_k$ is defined as the M-length vector formed by taking the k'th element/pixel from each channel. $\mathbf{P} \in \mathbb{C}^{M\times M}$ and $\mathbf{Q} \in \mathbb{C}^{N\times N}$ are unitary matrix transforms applied element-wise over the input and output channels. We use tilde ($\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{W}}$) to indicate tensors in the transformed feature space. Note that: (1) $\mathbf{P}$ and $\mathbf{Q}$ can be identity transforms, and thus UGConv includes group convolutions; (2) unitary transforms preserve inner products, thus they should not diminish gradient magnitudes in the network; (3) UGConv can also be applied to FC layers (using 1×1 feature maps and a 1×1 group conv).

One key point to make is the equivalence between a *group conv* and a convolution with *block-diagonal weights* (i.e. weights which consist of sub-blocks of square diagonal matrices). Figure 1(c) shows a block-diagonal conv, which visually already looks identical to the group conv in Figure 1(b). More formally, divide $\mathbf{X}$ and $\mathbf{Y}$ into size $D \times 1$ sub-blocks, and W into $D \times D$ sub-blocks which are diagonal. Let i index the input sub-blocks ($0 \le i \le M/D - 1$), j index the output sub-blocks ($0 \le j \le N/D - 1$), and d index the channels within each sub-block ($1 \le d \le D$). We

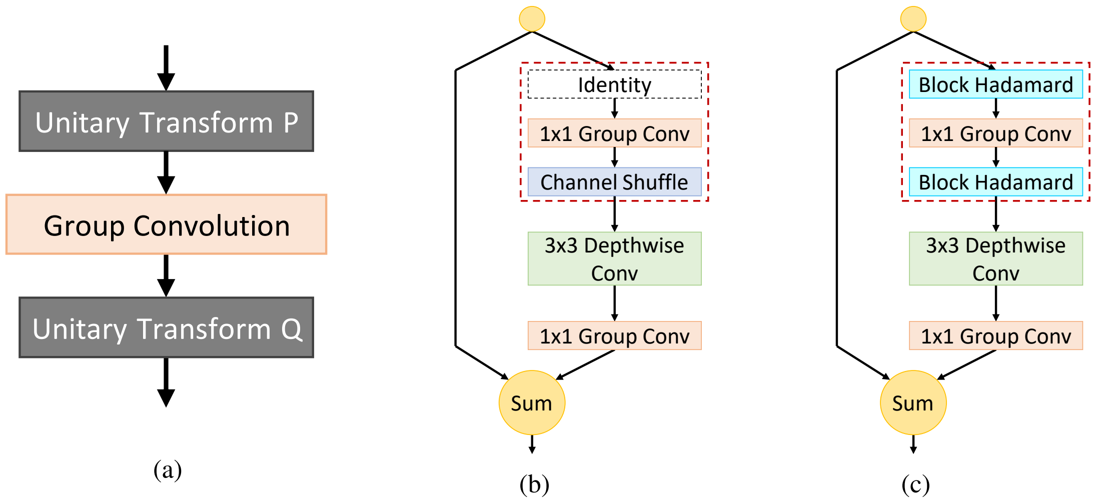

Figure 2. **CNN block architectures –** (a) a general block for unitary group convolutions; (b) a ShuffleNet block reproduced from the original paper [29]; (c) our proposed HadaNet variation. Note that both ShuffleNet and HadaNet blocks contain the UGConv pattern.

can express the block-diagonal conv as follows:

$$\mathbf{y}^{(j*D+d)} = \sum_{i=0}^{M/D-1} \mathbf{x}^{(i*D+d)} * \mathbf{W}^{(i*D+d\;\; j*D+d)}$$

Only D convs need to be performed for each $D \times D$ sub-block because they are diagonal. Similar to Equation 2, we can simplify notation by re-labeling using a tuple (*sub-block*, *channel in sub-block*). This removes the multiplies by D and allows i and j to start from 1. Then:

$$\mathbf{y}^{(j,d)} = \sum_{i=1}^{M/D} \mathbf{x}^{(i,d)} * \mathbf{W}^{(ij,d)}, \quad 1 \leq j \leq N/D \tag{4}$$

It is easy to see that Equation 4 matches Equation 2.

3.1. UGConv and ShuffleNet

ShuffleNet is a variant of the MobileNets architecture in which the pointwise convolutions (which take up 93.4% of the multiply-accumulate operations [29]) are converted into group convolutions. However, when multiple group convs are stacked together, the lack of connections between groups over many layers prevents the learning of cross-group correlations. To address this, ShuffleNet shuffles the output channels groups in a fixed, round-robin manner — for each group, the first channel is shuffled into group 1, the second channel into group 2, etc. This shuffle can be expressed as a permutation in feature space, and ShuffleNets are thus an example of of UGConvNets where $\mathbf{P}$ is identity and $\mathbf{Q}$ is a fixed permutation matrix.

ShuffleNet shows experimentally that it is beneficial to shuffle information across groups when stacking group convs. However, shuffling channels is not the only way to accomplish such information mixing.

3.2. UGConv and Circulant Networks

Circulant and block-circulant neural networks [6, 23] utilize layers that impose a block-circulant structure on their weight tensors. For an FC layer, the 2D weight matrix is made to be circulant. For a conv layer, the circulant structure is applied over the input and output channels axes. That is to say, given a 4D convolutional weight tensor with shape `(height, width, in_channels, out_channels)`, each 2D slice of this tensor `[i,j,:,:]` becomes circulant.

Figure 1(d) shows a block-circulant layer where each 2×2 sub-block of the weight tensor is circulant. By Equation (1), each $D \times D$ circulant matrix can be decomposed into a D-length DFT, a diagonal matrix, and a corresponding IDFT. In Figure 1(e), each $D \times D$ sub-block is diagonalized in this fashion. We use tilde to indicate weight values in the DFT-transformed space. The resulting weight structure is block-diagonal, and the weight layer sits between two block-DFT transforms. We know from the previous section that block-diagonal weights correspond to group convolutions. Therefore, *a block-circulant layer is just a group convolution in a transformed feature space*. This of course falls within the definition of a UGConv, with $\mathbf{P}$ and $\mathbf{Q}$ being block-DFT/IDFT transforms. Note that these DFTs are applied along the channels, and so circulant networks are *not* examining the spatial frequency components of the image.

We make a few additional notes about block-circulant layers. First, the size of the circulant blocks D is equal to the *number of groups* in the equivalent group conv (not the group size). Thus each D-length DFT touches a single channel in every group, fully mixing information between groups. Second, though our example uses a ”square” weight tensor (i.e. $M = N$), non-square block-circulant tensors can be diagonalized as well. As long as both M and N are divisible by D, the 'rectangular' weight tensor can be divided into $D \times D$ blocks. In this case, $\mathbf{P} \in \mathbb{C}^{\mathbf{M}\times\mathbf{M}}$ is not the inverse of $\mathbf{Q} \in \mathbb{C}^{\mathbf{N}\times\mathbf{N}}$, but each sub-block along the diagonal of $\mathbf{P}$ is the inverse of the corresponding sub-block in $\mathbf{Q}$. We say that $\mathbf{P}$ is the *block-inverse* of $\mathbf{Q}$.

Because $\mathbf{P}$ and $\mathbf{Q}$ are block-inverses, if we directly stack multiple such blocks many of the transforms will cancel out. However, practical DNNs include batch norm and/or nonlinearities between linear layers. The block-DFTs (and orthogonal transforms in general) do not commute with channel-wise or pointwise operations, which prevents trivial cancellation. However, note that channel shuffles *do* commute and cancel out in this manner.

3.3. Discussion of UGConvs

We have provided two specific examples from literature (ShuffleNet [29] and CirCNN [6]) which combine a structured sparse weight layer (group convolution) with unitary transforms. The transforms help to improve cross-channel representation learning without adding additional parameters. However, the two techniques have important differences. ShuffleNet's permutations are very lightweight as they require no arithmetic operations. However, permutations do not affect the sparsity of weight layer. On the other hand, CirCNN composes block-DFTs with a group conv to create an effective weight structure (i.e. circulant weights) which is dense. Moreover, it does so while still having less asymptotic computational complexity than unstructured dense weights.

We hypothesize that the representation learning capability of a UGConv layer is a function of both the sparsity of the weights as well as that of the transform. An unstructured dense weight layer offers the best learning capability; grouping introduces sparsity and degrades cross-channel learning performance, some of which can be recovered via transforms. Because dense transforms create dense weight structures (i.e. circulant weights), we believe they enable learning a richer set of representations compared to sparse transforms (i.e. channel shuffling). When the weight sparsity is low (i.e. number of groups is small), the difference between the two may be negligible in terms of network accuracy. However, we expect dense transforms to outperform shuffling when using many groups.

Another difference is that ShuffleNet applies channel shuffle on only one side of the weight layer, while CirCNN effectively applies transformations on both sides. We use the terms 1-sided and 2-sided UGConvs to refer to these two cases, and test both in our experiments.

3.4. The Hadamard Transform

One drawback of dense transforms such as DFT is that they require more computational overhead as compared to shuffling. Even using the 'fast' algorithm, each $n \times n$ DFT requires $O(n \log n)$ floating-point multiplies and adds. Furthermore, the fact that the DFT uses complex numbers may further complicate software/hardware implementations. Finally, the DFT is taken over the channels where there is no spatial structure — the transform exists purely to mix information across channels and not to perform domain-specific analysis. Given this, we would like to find a more efficient alternative.

Table 1. **Hadamard vs. Discrete Fourier transforms –** The entries of the DFT matrix are the complex roots of unity. The entries of the Hadamard matrix are $+1$ or -1. The last column shows the structure of $\mathbf{P}^*\mathbf{DP}$ where $\mathbf{D}$ is a diagonal matrix and $\mathbf{P}$ is the transform; differences are bolded.

	Fourier	**Hadamard**
Transform $\mathbf{P}$	$\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & \omega & \omega^2 & \omega^3\\ 1 & \omega^2 & \omega^4 & \omega^6\\ 1 & \omega^3 & \omega^6 & \omega^9\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{bmatrix}$
Structure of $\mathbf{P} * \mathbf{DP}$	$\begin{bmatrix}a & b & c & d\\ d & a & b & c\\ c & d & a & b\\ b & c & d & a\end{bmatrix}$	$\begin{bmatrix}a & b & c & d\\ \mathbf{b} & a & \mathbf{d} & c\\ c & d & a & b\\ \mathbf{d} & c & \mathbf{b} & a\end{bmatrix}$
FP Muls	$n \log n$	0
FP Adds	$n \log n$	$n \log n$

The Hadamard transform [17] is defined as a matrix containing only $+1/-1$ elements and whose rows and columns are mutually orthogonal. Table 1 shows a 4×4 Hadamard matrix. Because all coefficients have magnitude 1, the transform can be computed without multiplies, i.e. using adds/subtracts only. This is extremely important as floating-point multiplies are typically the computational bottleneck for DNN computation on both GPUs and specialized hardware. In addition, the Hadamard transform can be generated recursively like the Fourier transform, meaning that a fast Hadamard transform (FHT) exists similar to the FFT to compute a n-length Hadamard transform in $O(n \log n)$ adds/subtracts [17]. The recursive nature of FHT also enables Hadamard kernels to be implemented without explicitly storing the matrix itself; instead the matrix can be generated on the fly (similar to existing implementations of FFT kernels). This means neither FHT nor FFT requires storing additional parameters.

Hadamard is more efficient that DFT, but does it achieve the same learning performance? There is some high-level intuition that this would be the case: Table 1 compares the weight structure imposed by $\mathbf{P}^*\mathbf{DP}$ when $\mathbf{P}$ is DFT and Hadamard. DFT results in a circulant matrix; Hadamard results in a nearly identical weight matrix with only a few different elements. We hypothesize that there will be no accu-

Table 2. **Test error on a toy MNIST network –** a 'G' in the layer width columns indicates a group layer. In the transform columns, $\mathbf{P}$ and $\mathbf{Q}$ denote 1-sided pre-conv and post-conv transforms, respectively; $\mathbf{PQ}$ denotes a 2-sided transform. All values are averaged over 5 runs and 90% confidence bounds for each value are at most $\pm5\%$.

Layer Width			**Transform**						
L2	**L3**	**L4**	**None**	**Rand Ortho**			**Rand Perm**		
Conv3x3	FC	FC		P	Q	PQ	P	Q	PQ
20	20,G	10	6%	**4%**	**4%**	**4%**	5%	6%	5%
20	20,G	10,G	27%	10%	8%	**4%**	27%	26%	25%
20,G	20,G	10	25%	**10%**	**10%**	**10%**	27%	20%	21%
20,G	20,G	10,G	60%	23%	**17%**	20%	57%	55%	57%

racy impact in replacing circulant weights with Hadamard-diagonalizable weights in neural nets.

We further speculate that dense unitary transforms in general, including DFT and Hadamard, achieve comparable learning performance. This is again because the ordering of channels in DNNs is essentially random (i.e. the channel order encodes no useful information), meaning there are no patterns that can be exploited by one particular cross-channel transform and not others. The transforms in UGConv exist solely to connect different channel groups, and any dense transform will work as well as another. To test this hypothesis, we experiment with randomly generated orthogonal transforms in addition to DFT and Hadamard.

4. Experimental Validation

We first present ablation studies on a toy MNIST network followed by deeper CIFAR-10 models. These experiments build up insights on UGConv. We then demonstrate the utility of Hadamard using grouped ResNets and a ShuffleNet model from literature trained on ImageNet.

4.1. Dense Transforms vs. Shuffle

Our first experiment uses a toy MNIST network. This allows us to isolate the UGConv block and to compare dense orthogonal transforms versus permutations in a simple setting. We stress that the goal here is *not* to build a realistic classifier. The layer architecture is denoted below, where each layer is described as (*number of channels*)(*layer type*):

$$10\text{Conv3x3} - 20\text{Conv3x3} - \textbf{20FC} - 10\text{FC}$$

We perform 2×2 max pooling before each 3×3 conv layer, and a global average pool before the first FC layer. Each layer is followed by batch normalization and ReLU.

We convert the first FC layer of the network (20FC1, shown in bold) into a UGConv block (i.e. it becomes a grouped FC with transforms). The group number is equal to the number of channels to maximize sparsity. From this base architecture we derive three variations: (1) convert the preceding Conv3x3 layer into group conv; (2) convert the following FC layer into group FC; (3) convert both surrounding layers into group layers. These test the performance of transforms in the context of stacked group layers. Two types of transform are evaluated: randomly generated dense orthogonal and random permutation transforms. We test with both 1-sided (using one of $\mathbf{P}$ or $\mathbf{Q}$ and setting the other to identity) and 2-sided UGConvs ($\mathbf{P} = \mathbf{Q}^{-1}$). All results are averaged over five runs, and we regenerate the random transformation matrices between runs.

Table 2 shows our results. Due to the small size of the network, the 90% confidence bound for these values can be as large as $\pm5\%$. Nevertheless, differences between transforms are clearly demonstrated. When L3 is the only grouped layer in the network (row 1), transforms have little to no effect. However, when two or more group layers are stacked together, the dense orthogonal transforms achieve improved accuracy. Permutations did not improve accuracy in any experiment. This is a clear (albeit artificial) demonstration that when the number of groups is very large, dense transforms outperform permutations in learning ability.

Another interesting observation is that there is little difference between 1-sided and 2-sided transforms, regardless of whether the UGConv block is stacked before or after another group layer. For example, in Table 2 row 3, a dense orthogonal transform improves accuracy even when it is placed *after* both group layers. It may be surprising that a transform affects layers preceding it. But keep in mind that the transform also affects gradients on the backwards pass, allowing the same weights to 'see' more downstream activations during backpropagation. Alternatively, we can view the UGConv layer as a learnable structured weight layer (see Section 3.2) — within this perspective, the weight structure is a function of transforms both before or after.

4.2. Evaluation of Different Transforms

We have shown that dense orthogonal transforms can improve over shuffles in small DNNs with large group sizes. To validate our results on more realistic architectures, we perform experiments on CIFAR-10 [12] using ResNet [7]. We use UGConvs to replace the two 3×3 convolutions in

Table 3. **Test error for UGConvs on CIFAR-10 –** The first three columns show the number of groups used in the three stages (S1-S3). The **Base** column shows the test error with no transforms, and the other columns show *improvement* in test error over this baseline. Some entries are blank due to insufficient time to complete the experiments.

	# of Groups			**Base**	**1-sided Transforms**			**2-sided Transforms**				**Params**
	S1	S2	S3		Shuffle	Hada	Ortho	Shuffle*	Fourier	Hada	Ortho	
ResNet-20	4	8	16	19.5%	3.3%	4.0%	4.0%	3.1%	4.1%	**4.2%**	3.8%	25K
	8	16	32	23.8%	2.9%	4.3%	3.9%	4.1%	**5.4%**	**5.4%**	5.3%	14K
ResNet-56	4	8	16	16.0%	4.0%	4.4%	4.2%	4.0%	**4.7%**	4.5%	4.6%	76K
	8	16	32	20.6%	5.4%	6.1%	6.4%	5.8%	7.1%	**7.2%**	6.8%	41K
ShuffleNet-29	4	8	16	18.3%	2.7%	2.4%	3.1%	3.8%	**4.9%**	4.5%	4.2%	23K
	8	16	32	22.1%	0.6%	3.4%	3.6%	3.8%	5.1%	5.0%	**5.3%**	17K
ShuffleNet-56	4	8	16	16.2%	3.6%	3.5%	3.4%	3.9%	4.6%	4.5%	**4.7%**	41K
	8	16	32	19.7%	4.3%	4.4%	4.9%	5.2%	**6.0%**	**6.0%**	**6.0%**	29K
Mean	4	8	16	17.5%	3.4%	3.6%	3.7%	3.7%	**4.6%**	4.4%	4.3%	
	8	16	32	21.5%	3.3%	4.6%	4.7%	4.7%	**5.9%**	**5.9%**	**5.9%**	

each ResNet block, and to replace the 1×1 projection layers. ResNets are divided into three stages (S1, S2, S3), with later stages having more channels. We use more groups in later stages, keeping the ratio of channels to groups constant. Two models are tested: **ResNet-20** (3 block per stage) and **ResNet-56** (9 blocks per stage). We also experiment with the same high-level architecture but using the building block from ShuffleNet [29]. This block which contains two 1×1 convs and a 3×3 depthwise conv (see Figure 2(b)). Following ShuffleNet we apply transformations around the first 1×1 group conv only and make no changes to the second group conv. Again, two models are tested: **ShuffleNet-29** (3 block per stage) and **ShuffleNet-56** (6 blocks per stage).

We use layer widths and training hyperparameters from [7] and make use of standard data augmentations: padding 8 pixels on each side and randomly cropping back to original size, combined with a random horizontal flip [7, 9, 15]. Each network is trained for 200 epochs, and we report the mean test error over the last 5 epochs.

We test the following transforms: identity (None), ShuffleNet permutation (Shuffle), block-Hadamard (Hada), block-DFT (Fourier), and block-random-orthogonal (Ortho). The block transforms follow the same structure described in Section 3.2. For each transform, both 1-sided (letting $\mathbf{Q}$ be the transform and $\mathbf{P}$ identity) and 2-sided ($\mathbf{P}$ and $\mathbf{Q}$ are block-inverses) versions are tested where reasonable. The 1-sided DFT is left out because it introduced complex numbers into the network. For the 2-sided channel shuffle (Shuffle*), we set $\mathbf{P} = \mathbf{Q}$ to essentially perform additional shuffling; this is done since using block-inverse shuffles will lead to trivial cancellation. All results are displayed in Table 3 — the error rate with no transforms is given first followed by the accuracy improvement achieved with each UGConv setup. Our base error rates are high for CIFAR-10 because group convolutions significantly compress the network

A key result here is that dense orthogonal transforms perform similarly in accuracy. Fourier, Hada, and Ortho obtain results which are within a spread of 0.4% in both 1-sided and 2-sided settings. On the other hand, the shuffle transforms (1 and 2-sided) clearly perform worse for the larger group sizes. This confirms our hypothesis that Hadamard is comparable to DFT in learning performance while being much easier to compute. It also provides evidence that *all* dense UGConvs achieve comparable learning performance.

Another observation is that 2-sided transforms significantly outperform their 1-sided variants, which is different from the MNIST data. We currently do not have an explanation for this effect. One speculation was that 2-sided transforms perform better when the number of input and output channels did not match. However, further testing with the small MNIST network showed that this was not the case.

Finally, note that the accuracy trends remained the same whether the transforms were applied to 3×3 group convs in ResNet or 1×1 group convs in ShuffleNet. This is evidence that spatial and cross-channel dependencies are effectively decoupled in convolutional layers, and that the size of the filter does not significantly affect channel-space transforms.

4.3. Hadamard Networks on ImageNet

The data from previous sections point to two regimes: at low weight sparsity (i.e. small group numbers) a simple shuffle is sufficient to maximize accuracy. At large group numbers, however, dense transforms outperform shuffles. This section evaluates the 2-sided block-Hadamard transform against shuffle on ImageNet. Hadamard was chosen as it is far more efficient than other dense unitary transforms

Table 4. **Top-1 classification error on ImageNet –** we include data on both the original ShuffleNet (with our own code) and our pre-activation variation. Our baseline ShuffleNet implementation is close to the literature results (52.7%). For each model we show the number of parameters and fpmuls, as well as the overhead in additions from the Hadamard transform.

	Shuffle	Hada	Delta	Params	FPmuls	Hada Adds
ResNet-18 g8	46.4%	**44.6%**	(-1.8%)	1.9M	330M	7.8M
ResNet-18 g16	55.8%	**52.3%**	(-3.5%)	1.2M	226M	10.4M
ShuffleNet-x0.25 g8	53.6%	**52.6%**	(-1.0%)	0.46M	17M	0.95M

(see Section 3.4, and ShuffleNet was used for comparison as it is highly related work and a strong baseline. We refer to networks using Hadamard UGConvs as HadaNets. Figure 2 compares the residual blocks of ShuffleNet and HadaNet.

Due to hardware constraints, we chose small models with fairly large group size — this is the setting where dense transforms should perform the best compared to shuffle. We evaluate ResNet-18 following the ImageNet architecture from [7] and using group sizes 8 and 16 throughout the network. We also test with the ShuffleNet-x0.25 g8, which is the smallest ShuffleNet variant from [29]. This network has 50 layers and also uses 8 groups. Each network was trained with the hyperparameters and learning rate schedule described in their respective papers. We compare 1-sided shuffle to 2-sided block-Hadamard (note that ShuffleNet from literature already contains the 1-sided shuffle). All results are displayed in Table 4. Our reproduction of ShuffleNet-x0.25-g8 achieved a Top-1 error of 53.6%, which is close to the 52.7% reported in Table 2 of [29].

The results demonstrate that the Hadamard transform can indeed outperform shuffling in terms of accuracy on large scale datasets. ResNet-18 with group convs is a non-standard model, but it serves to show that the trends observed in CIFAR-10 ResNets carry over to ImageNet. On the other hand, ShuffleNet is a well-optimized baseline which obtains good accuracy performance on a very tight parameter and fpmul budget. In addition, despite very little hyperparameter tuning, HadaNet was able to improve slightly over ShuffleNet.

4.4. Practicality of HadaNet

HadaNet slightly outperforms ShuffleNet on accuracy, but requires extra floating-point adds. An N-channel group conv with B groups requires N^2/B fpmuls for the weight layer and $2N \log B$ adds for the two block-Hadamard transforms. Compared to multiplies, additions are already much cheaper in hardware. The last column of Table 4 shows the number of additions needed for each network if the fast Hadamard transform is used. The relative overhead of HadaNet is fairly small: the extra adds amount to only 2-5% of existing multiply-accumulates in those networks.

However, the overhead of the Hadamard transform depends on a well-optimized implementation. The reason we did not show runtime on GPU is that an $O(n \log n)$ fast Hadamard kernel operating along the channels is not currently available — as a result our own HadaNet implementation is fairly slow.

On the other hand, we believe the Hadamard transform might be useful for specialized DNN accelerators implemented with FPGAs [3] or ASICs [11]. Top computer hardware conferences already contain works demonstrating the use of circulant matrices for DNN compression in dedicated hardware [6, 22, 4]. These works show that DFTs can be very efficiently implemented in a dedicated module due to its recursive nature. We choose Hadamard because it also has the same recursive properties, meaning it should be even simpler in hardware due to lower computational complexity. All-in-all, this paper reveals that in high weight sparsity regimes, dense transforms outperform simple shuffling. HadaNet is more efficient than the existing state-of-the-art dense transform (i.e. DFT transforms) while achieving similar accuracy performance in DNNs.

5. Conclusions and Future Work

We introduce the concept of unitary group convolutions, a composition of group convolutions with unitary transforms in feature space. We use the UGConv framework to unify two disparate ideas in CNN literature, ShuffleNets and block-circulant networks, and provide valuable insights into both techniques. UGConvs with dense unitary transforms demonstrate superior ability to learn cross-channel mappings versus ordinary and shuffled group convolutions. Based on these these observations we propose HadaNet, a variant of ShuffleNet that improves accuracy on the ImageNet dataset without incurring additional parameters or floating-point multiplies.

One future work is to replace the Hadamard transform with a trained $0, +1, -1$ transform; training may allow the transform to adapt to the weights, and introducing zeros enables sparse compute reduction.

Acknowledgments

This work was supported in part by the Semiconductor Research Corporation (SRC) and DARPA. We would like to thank Prof. Yanzhi Wang (Northeastern University), Prof. Bo Yuan (Rutgers University), and their students for providing technical details and code regarding CirCNN [6].

References

[1] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An Exploration of Parameter Redundancy in Deep Networks with Circulant Projections. *Int'l Conf. on Computer Vision (ICCV)*, pages 2857–2865, 2015.

[2] François Chollet. Xception: Deep learning with Depthwise Separable Convolutions. *Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.

[3] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Maleen Abeydeera, Logan Adams, Hari Angepat, Christian Boehn, Derek Chiou, Oren Firestein, Alessandro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Ahmad El Husseini, Tamas Juhasz, Kara Kagi, Ratna K. Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Brandon Perez, Amanda Grace Rapsang, Steven K. Reinhardt, Bita Darvish Rouhani, Adam Sapek, Raja Seera, Sangeetha Shekar, Balaji Sridharan, Gabriel Weisz, Lisa Woods, Phillip Yi Xiao, Dan Zhang, Ritchie Zhao, , and Doug Burger. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave . *IEEE Micro*, 38(2):8–20, 2018.

[4] Chunhua Deng, Siyu Liao, Yi Xie, Keshab K. Parhi, Xuehai Qian, and Bo Yuan. PermDNN: Efficient Compressed Deep Neural Network Architecture with Permuted Diagonal Matrices . *Int'l Symp. on Microarchitecture (MICRO)*, Oct 2019.

[5] T. Ceren Deveci, Serdar Cakir, and A. Enis Cetin. Energy Efficient Hadamard Neural Networks. *arXiv preprint*, arXiv:1805.05421, May 2018.

[6] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, Xiaolong Ma, Yipeng Zhang, Jian Tang, Qinru Qiu, Xue Lin, and Bo Yuan. CirCNN: Accelerating and Compressing Deep Neural Networks using Block-Circulant Weight Matrices. *Int'l Symp. on Microarchitecture (MICRO)*, pages 395–408, 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv e-print*, arXiv:1512.0338, Dec 2015.

[8] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient Convolutional Neural Networks for Nobile Vision Applications. *arXiv e-print*, arXiv:1704.04861, 2017.

[9] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep Networks with Stochastic Depth. *European Conference on Computer Vision (ECCV)*, pages 646–661, 2016.

[10] Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi. Deep Roots: Improving CNN Efficiency with Hierarchical Filter Groups. *Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun 2017.

[11] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. *Int'l Symp. on Computer Architecture (ISCA)*, pages 1–12, 2017.

[12] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. *Tech report*, 2009.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[14] Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood —Approximating Kernel Expansions in Loglinear Time. *Int'l Conf. on Machine Learning (ICML)*, 85, 2013.

[15] Min Lin, Qiang Chen, and Shuicheng Yan. Network in Network. *arXiv e-print*, arXiv:1312.4400, 2013.

[16] Marcin Moczulski, Misha Denil, Jeremy Appleyard, and Nando de Freitas. ACDC: A Structured Efficient Linear Layer. *Int'l Conf. on Learning Representations (ICLR)*, 2016.

[17] William K Pratt, Julius Kane, and Harry C Andrews. Hadamard Transform Image Coding. *Proceedings of the IEEE*, 57(1):58–68, 1969.

[18] Laurent Sifre. Rigid-Motion Scattering for Image Classification. *Ph.D. thesis*, 2014.

[19] Karen Simonyan and Anderw Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-print*, arXiv:1409.15568, Apr 2015.

[20] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured Transforms for Small-Footprint Deep Learning. *Advances in Neural Information Processing Systems (NIPS)*, pages 3088–3096, 2015.

[21] Ke Sun, Mingjie Li, Dong Liu, and Jingdong Wang. IGCV3: Interleaved Low-Rank Group Convolutions for Efficient Deep Neural Networks. *arXiv e-print*, arXiv:1806.00178, Jun 2018.

[22] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, Yanzhi Wang, and Yun Liang. C-LSTM: Enabling Efficient LSTM using Structured Compression Techniques on FPGAs. *to appear in International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018.

[23] Yanzhi Wang, Caiwen Ding, Zhe Li, Geng Yuan, Siyu Liao, Xiaolong Ma, Bo Yuan, Xuehai Qian, Jian Tang, Qinru Qiu, and Xue Lin. Towards Ultra-High Performance and Energy Efficiency of Deep Learning Systems: An Algorithm-Hardware Co-Optimization Framework. *AAAI Conf' on Artificial Intelligence (AAAI)*, Feb 2018.

[24] Guotian Xie, Jingdong Wang, Ting Zhang, Jianhuang Lai, Richang Hong, and Guo-Jun Qi. IGCV2: Interleaved Structured Sparse Convolutional Neural Networks. *arXiv e-print*, arXiv:1804.06202, Apr 2018.

[25] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. *Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun 2017.

[26] Xiaowei Xu, Yukun Ding, Sharon Xiaobo Hu, Michael Niemier, Jason Cong, Yu Hu, and Yiyu Shi. Scaling for Edge Inference of Deep Neural Networks. *Nature Electronics*, 1(4):216, 2018.

[27] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep Fried Convnets. *Int'l Conf. on Computer Vision (ICCV)*, pages 1476–1483, 2015.

[28] Ting Zhang, Guo-Jun Qi, Bin Xiao, and Jingdong Wang. Interleaved Group Convolutions. *Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun 2017.

[29] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *arXiv e-print*, arXiv:1707.01083, Aug 2017.