Video Classification with Channel-Separated Convolutional Networks

Du Tran Heng Wang Lorenzo Torresani Matt Feiszli
Facebook AI
{trandu,hengwang,torresani,mdf}@fb.com

Abstract

Group convolution has been shown to offer great computational savings in various 2D convolutional architectures for image classification. It is natural to ask: 1) if group convolution can help to alleviate the high computational cost of video classification networks; 2) what factors matter the most in 3D group convolutional networks; and 3) what are good computation/accuracy trade-offs with 3D group convolutional networks.

This paper studies the effects of different design choices in 3D group convolutional networks for video classification. We empirically demonstrate that the amount of channel interactions plays an important role in the accuracy of 3D group convolutional networks. Our experiments suggest two main findings. First, it is a good practice to factorize 3D convolutions by separating channel interactions and spatiotemporal interactions as this leads to improved accuracy and lower computational cost. Second, 3D channel-separated convolutions provide a form of regularization, yielding lower training accuracy but higher test accuracy compared to 3D convolutions. These two empirical findings lead us to design an architecture – Channel-Separated Convolutional Network (CSN) – which is simple, efficient, yet accurate. On Sports1M and Kinetics, our CSNs are comparable with or better than the state-of-the-art while being 2-3 times more efficient.

1. Introduction

Video classification has witnessed much good progress in recent years. Most of the accuracy gains have come from the introduction of new powerful architectures [3, 30, 23, 37, 35]. However, many of these architectures are built on expensive 3D spatiotemporal convolutions. Furthermore, these convolutions are typically computed across all the channels in each layer. 3D CNNs have complexity $\mathcal{O}(CTHW)$ as opposed to the cost of $\mathcal{O}(CHW)$ of 2D CNNs, where T denotes the number of frames, H, W the spatial dimensions and C the number of channels. For both foundational and practical reasons, it is natural to ask which parameters in these large 4D kernels matter the most.

Kernel factorizations have been applied in several settings to reduce compute and improve accuracy. For example, several recent video architectures factor 3D convolution in space and time: examples include P3D [23], R(2+1)D [30], and S3D [37]. In these architectures, a 3D convolution is replaced with a 2D convolution (in space) followed by a 1D convolution (in time). This factorization can be leveraged to increase accuracy and/or to reduce computation. In the still-image domain, separable convolution [7] is used to factorize the convolution of 2D $k \times k$ filters into a pointwise 1×1 convolution followed by a depthwise $k \times k$ convolution. When the number of channels is large compared to k^2, which is usually the case, this reduces FLOPs by $\sim k^2$ for images. For the case of 3D video kernels, the FLOP reduction is even more dramatic: $\sim k^3$.

Inspired by the accuracy gains and good computational savings demonstrated by 2D separable convolutions in image classification [7, 17, 39], this paper proposes a set of architectures for video classification – 3D Channel-Separated Networks (CSN) – in which all convolutional operations are separated into either pointwise $1{\times}1{\times}1$ or depthwise $3{\times}3{\times}3$ convolutions. Our experiments reveal the importance of *channel interaction* in the design of CSNs. In particular, we show that excellent accuracy/computational cost balances can be obtained with CSNs by leveraging channel separation to reduce FLOPs and parameters as long as high values of channel interaction are retained. We propose two factorizations, which we call *interaction-reduced* and *interaction-preserved*. Compared to 3D CNNs, both our interaction-reduced and interaction-preserved CSNs provide higher accuracy and FLOP savings of about 2.5-3× *when* there is enough channel interaction. We experimentally show that the channel factorization in CSNs acts as a regularizer, leading to a higher training error but better generalization. Finally, we show that our proposed CSNs outperform or are comparable with the current state-of-the art methods on Sports1M and Kinetics while being 2–3 times faster.

2. Related Work

Group convolution. Group convolution was adopted in AlexNet [20] as a way to overcome GPU memory limitations. Depthwise convolution was introduced in MobileNet [17] as an attempt to optimize model size and computational cost for mobile applications. Chollet [7] built an extreme version of Inception [28] based on 2D depthwise convolution, named Xception, where the Inception block was redesigned to include multiple separable convolutions. Concurrently, Xie *et al.* proposed ResNeXt [36] by equipping ResNet [16] bottleneck blocks with groupwise convolution. Further architecture improvements have also been made for mobile applications. ShuffleNet [39] further reduced the computational cost of the bottleneck block with both depthwise and group convolution. MobileNetV2 [25] improved MobileNet [17] by switching from a VGG-style to a ResNet-style network, and introducing a "reverted bottleneck" block. All of these architectures are based on 2D CNNs and are applied to image classification while our work focuses on 3D group CNNs for video classification.

Video classification. In the last few years, video classification has seen a major paradigm shift, which involved moving from hand-designed features [21, 8, 24, 31] to deep network approaches that learn features and classify end-to-end [29, 18, 26, 11, 33, 34, 12]. This transformation was enabled by the introduction of large-scale video datasets [18, 19] and massively parallel computing hardware, *i.e*., GPU. Carreira and Zisserman [3] recently proposed to inflate 2D convolutional networks pre-trained on images to 3D for video classification. Wang *et al.* [35] proposed non-local neural networks to capture long-range dependencies in videos. ARTNet [32] decouples spatial and temporal modeling into two parallel branches. Similarly, 3D convolutions can also be decomposed into a Pseudo-3D convolutional block as in P3D [23] or factorized convolutions as in R(2+1)D [30] or S3D [37]. 3D group convolution was also applied to video classification in ResNeXt [15] and Multi-Fiber Networks [5] (MFNet).

Among previous approaches, our work is most closely related to the following architectures. First, our CSNs are similar to Xception [7] in the idea of using channel-separated convolutions. Xception factorizes 2D convolution in channel and space for object classification, while our CSNs factorize 3D convolution in channel and space-time for action recognition. In addition, Xception uses simple blocks, while our CSNs use bottleneck blocks. The variant ir-CSN of our model shares similarities with ResNeXt [36] and its 3D version [15] in the use of bottleneck block with group/depthwise convolution. The main difference is that ResNext [36, 15] uses group convolution in its 3×3×3 layers with a fixed group size (*e.g*., $g = 32$), while our ir-CSN uses depthwise convolutions in all 3×3×3 layers which makes our architecture fully channel-separated. As we will show in section 4.2, making our network fully channel-separated helps not only to reduce a significant amount of compute, but also to improve model accuracy by better regularization. We emphasize that our contribution includes not only the design of CSN architectures, but also a systematic empirical study of the role of channel interactions in the accuracy of CSNs.

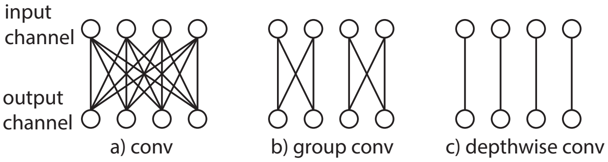

Figure 1. **Group convolution**. Convolutional filters can be partitioned into groups with each filter receiving input from channels only within its group. (a) A conventional convolution, which has only one group. (b) A group convolution with 2 groups. (c) A depthwise convolution where the number of groups matches the number of input/output filters, i.e., each group contains only one channel.

3. Channel-Separated Convolutional Networks

In this section, we discuss the concept of 3D channel-separated networks. Since channel-separated networks use group convolution as their main building block, we first provide some background about group convolution.

3.1. Background

Group convolution. Conventional convolution is implemented with dense connections, i.e., each convolutional filter receives input from all channels of its previous layer, as in Figure 1(a). However, in order to reduce the computational cost and model size, these connections can be sparsified by grouping convolutional filters into subsets. Filters in a subset receive signal from only channels within its group (see Figure 1(b)). Depthwise convolution is the extreme version of group convolution where the number of groups is equal to the number of input and output channels (see figure 1(c)). Xception [7] and MobileNet [17] were among the first networks to use depthwise convolutions. Figure 1 presents an illustration of conventional, group, and depthwise convolutional layers for the case of 4 input channels and 4 output channels.

Counting FLOPs, parameters, and interactions. Dividing a conventional convolutional filter into G groups reduces compute and parameter count by a factor of G. These reductions occur because each filter in a group receives input from only a fraction $1/G$ of the channels from the previous layer. In other words, channel grouping restricts feature interaction: only channels within a group can interact. If multiple group convolutional layers are stacked directly on top of each other, this feature segregation is further amplified since each channel becomes a function of small channel-subsets in all preceding layers. So, while group convolution saves compute and parameters, it also reduces feature interactions.

We propose to quantify the amount of channel interaction as the number of pairs of two input channels that are connected through any output filter. If the convolutional layer has C_{in} channels and G groups, each filter is connected to C_{in}/G input channels. Therefore each filter will have $\binom{\frac{C_{in}}{G}}{2}$ interacting feature pairs. According to this definition, the example convolutions in Figure 1(a)-(c) will have 24, 4, and 0 channel interaction pairs, respectively.

Consider a 3D convolutional layer with spatiotemporal convolutional filters of size $k \times k \times k$ and G groups, C_{in} input channels, and C_{out} output channels. Let THW be the total number of voxels in the spatiotemporal tensor provided as input to the layer. Then, the number of parameters, FLOPs (floating-point operations), and number of channel interactions can be measured as:

$$\#\text{parameters} = C_{out} \cdot \frac{C_{in}}{G} \cdot k^3 \quad (1)$$

$$\#\text{FLOPs} = C_{out} \cdot \frac{C_{in}}{G} \cdot k^3 \cdot THW \quad (2)$$

$$\#\text{interactions} = C_{out} \cdot \binom{\frac{C_{in}}{G}}{2} \quad (3)$$

Recall that $\binom{n}{2} = \frac{n(n-1)}{2} = \mathcal{O}\left(n^2\right)$. We note that while FLOPs and number of parameters are commonly used to characterize a layer, the "amount" of channel interaction is typically overlooked. Our study will reveal the importance of this factor.

3.2. Channel Separation

We define channel-separated convolutional networks (CSN) as 3D CNNs in which all convolutional layers (except for `conv1`) are either $1 \times 1 \times 1$ conventional convolutions or $k \times k \times k$ depthwise convolutions (where, typically, $k = 3$). Conventional convolutional networks model channel interactions and local interactions (i.e., spatial or spatiotemporal) jointly in their 3D convolutions. Instead, channel-separated networks decompose these two types of interactions into two distinct layers: $1 \times 1 \times 1$ conventional convolutions for channel interaction (but no local interaction) and $k \times k \times k$ depthwise convolutions for local spatiotemporal interactions (but not channel interaction). Channel separation may be applied to any $k \times k \times k$ traditional convolution by decomposing it into a $1 \times 1 \times 1$ convolution and a depthwise $k \times k \times k$ convolution.

We introduce the term "channel-separated" to highlight the importance of channel interaction; we also point out that the existing term "depth-separable" is only a good description when applied to tensors with two spatial dimensions and one channel dimension. We note that channel-separated networks have been proposed in Xception [7] and MobileNet [17] for image classification. In video classification, separated convolutions have been used in P3D [23], R(2+1)D [30], and S3D [37], but to decompose 3D convolutions into separate temporal and spatial convolutions. The network architectures presented in this work are designed to separate channel interactions from spatiotemporal interactions.

3.3. Example: Channel-Separated Bottleneck Block

Figure 2 presents two ways of factorizing a 3D bottleneck block using channel-separated convolutional networks. Figure 2(a) presents a standard 3D bottleneck block, while Figure 2(b) and 2(c) present interaction-preserved and interaction-reduced channel-separated bottleneck blocks, respectively.

Interaction-preserved channel-separated bottleneck block is obtained from the standard bottleneck block (Figure 2(a) by replacing the $3 \times 3 \times 3$ convolution in (a) with a $1 \times 1 \times 1$ traditional convolution and a $3 \times 3 \times 3$ depthwise convolution (shown in Figure 2(b)). This block reduces parameters and FLOPs of the traditional $3 \times 3 \times 3$ convolution significantly, but preserves all channel interactions via a newly-added $1 \times 1 \times 1$ convolution. We call this an *interaction-preserved* channel-separated bottleneck block and the resulting architecture an *interaction-preserved channel-separated network* (ip-CSN).

Interaction-reduced channel-separated bottleneck block is derived from the preserved bottleneck block by removing the extra $1 \times 1 \times 1$ convolution. This yields the depthwise bottleneck block shown in Figure 2(c). Note that the initial and final $1 \times 1 \times 1$ convolutions (usually interpreted respectively as projecting into a lower-dimensional subspace and then projecting back to the original dimensionality) are now the only mechanism left for channel interactions. This implies that the complete block shown in (c) has a reduced number of channel interactions compared with those shown in (a) or (b). We call this design an *interaction-reduced* channel-separated bottleneck block and the resulting architecture an *interaction-reduced channel-separated network* (ir-CSN).

3.4. Channel Interactions in Convolutional Blocks

The interaction-preserving and interaction-reducing blocks in section 3.3 are just two architectures in a large spectrum. In this subsection we present a number of convolutional block designs, obtained by progressively increasing the amount of grouping. The blocks differ in terms of compute cost, parameter count and, more importantly, channel interactions.

Group convolution applied to ResNet blocks. Figure 3(a)

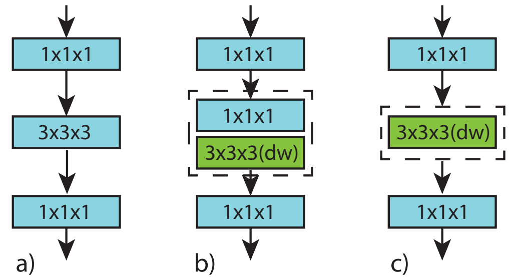

Figure 2. **Standard vs. channel-separated convolutional blocks.** (a) A standard ResNet bottleneck block. (b) An interaction-preserved bottleneck block: a bottleneck block where the 3×3×3 convolution in (a) is replaced by a 1×1×1 standard convolution and a 3×3×3 depthwise convolution (shown in dashed box). (c) An interaction-reduced bottleneck block, a bottleneck block where the 3×3×3 convolution in (a) is replaced with a depthwise convolution (shown in dashed box). We note that channel interaction is preserved in (b) by the 1×1×1 convolution, while (c) lost all of the channel interaction in its 3×3×3 convolution after factorization. Batch norm and ReLU are used after each convolution layer. For simplicity, we omit the skip connections.

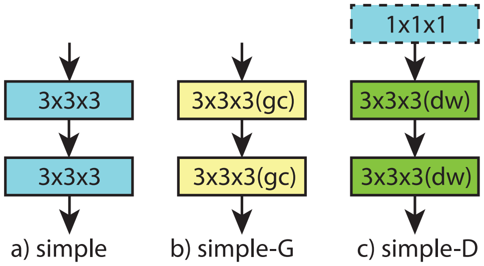

Figure 3. **ResNet simple block transformed by group convolution**. (a) Simple block: a standard ResNet simple block with two 3×3×3 convolutional layers. (b) Simple-G block: a ResNet simple block with two 3×3×3 group convolutional layers. (c) Simple-D block: a ResNet simple block with two 3×3×3 depthwise convolutional layers with an optional 1×1×1 convolutional layer (shown in dashed box) added when increasing number of filters is needed. Batch norm and ReLU are used after each convolution layer. For simplicity, we omit the skip connections.

presents a ResNet [16] **simple** block consisting of two 3×3×3 convolutional layers. Figure 3(b) shows the **simple-G** block, where the 3×3×3 layers now use grouped convolution. Likewise, Figure 3(c) presents **simple-D**, with two depthwise layers. Because depthwise convolution requires the same number of input and output channels, we optionally add a 1×1×1 convolutional layer (shown in the dashed rectangle) in blocks that change the number of channels.

Figure 4(a) presents a ResNet **bottleneck** block consisting of two 1×1×1 and one 3×3×3 convolutional layers. Figures 4(b-c) present **bottleneck-G** and **bottleneck-D** where the 3×3×3 convolutions are grouped and depthwise, respectively. If we further apply group convolution to the two 1×1×1 convolutional layers, the block becomes

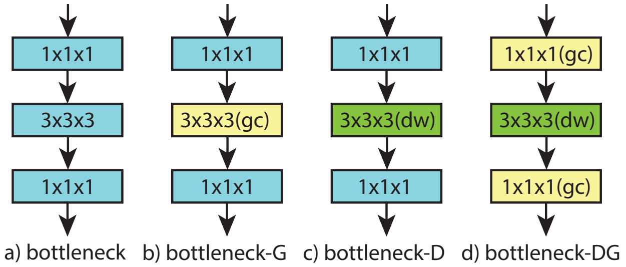

Figure 4. **ResNet bottleneck block transformed by group convolution**. (a) A standard ResNet bottleneck block. (b) Bottleneck-G: a ResNet bottleneck block with a 3×3×3 group convolutional layer. (c) Bottleneck-D: a bottleneck block with a 3×3×3 depthwise convolution (previously named as ir-CSN, the new name of Bottleneck-D is used here for simplicity and analogy with other blocks). (d) Bottleneck-DG: a ResNet bottleneck block with a 3×3×3 depthwise convolution and two 1×1×1 group convolutions. We note that from (a) to (d), we gradually apply group convolution to the 3×3×3 convolutional layer and then the two 1×1×1 convolutional layers. Batch norm and ReLU are used after each convolution layer. For simplicity, in the illustration we omit to show skip connections.

a **bottleneck-DG**, as illustrated in Figure 4(d). In all cases, the 3×3×3 convolutional layers always have the same number of input and output channels.

There are some deliberate analogies to existing architectures here. First, bottleneck-G (Figure 4(b)) is exactly a ResNeXt block [36], and bottleneck-D is its depthwise variant. Bottleneck-DG (Figure 4(d)) resembles the ShuffleNet block [39], without the channel shuffle and without the downsampling projection by average pooling and concatenation. The progression from simple to simple-D is similar to moving from ResNet to Xception (though Xception has many more 1×1×1 convolutions). We omit certain architecture-specific features in order to better understand the role of grouping and channel interactions.

4. Ablation Experiment

This empirical study will allow us to cast some light on the important factors in the performance of channel-separated network and will lead us to two main findings:

1. We will empirically demonstrate that within the family of architectures we consider, similar depth and similar amount of channel interaction implies similar accuracy. In particular, the interaction-preserving blocks reduce compute significantly but preserve channel interactions, with only a slight loss in accuracy for shallow networks and an increase in accuracy for deeper networks.

2. In traditional 3×3×3 convolutions all feature maps interact with each other. For deeper networks, we show

layer name	output size	ResNet3D-simple	ResNet3D-bottleneck
conv1	T×112×112	3×7×7, 64, stride 1×2×2	
pool1	T×56×56	max, 1×3×3, stride 1×2×2	
conv2_x	T×56×56	$\begin{bmatrix} 3\times3\times3, 64 \\ 3\times3\times3, 64 \end{bmatrix}\times b_1$	$\begin{bmatrix} 1\times1\times1, 256 \\ 3\times3\times3, 64 \\ 1\times1\times1, 256 \end{bmatrix}\times b_1$
conv3_x	$\frac{T}{2}\times28\times28$	$\begin{bmatrix} 3\times3\times3, 128 \\ 3\times3\times3, 128 \end{bmatrix}\times b_2$	$\begin{bmatrix} 1\times1\times1, 512 \\ 3\times3\times3, 128 \\ 1\times1\times1, 512 \end{bmatrix}\times b_2$
conv4_x	$\frac{T}{4}\times14\times14$	$\begin{bmatrix} 3\times3\times3, 256 \\ 3\times3\times3, 256 \end{bmatrix}\times b_3$	$\begin{bmatrix} 1\times1\times1, 1024 \\ 3\times3\times3, 256 \\ 1\times1\times1, 1024 \end{bmatrix}\times b_3$
conv5_x	$\frac{T}{8}\times7\times7$	$\begin{bmatrix} 3\times3\times3, 512 \\ 3\times3\times3, 512 \end{bmatrix}\times b_4$	$\begin{bmatrix} 1\times1\times1, 2048 \\ 3\times3\times3, 512 \\ 1\times1\times1, 2048 \end{bmatrix}\times b_4$
pool5	1×1×1	spatiotemporal avg pool, fc layer with softmax	

Table 1. **ResNet3D architectures considered in our experiments**. Convolutional residual blocks are shown in brackets, next to the number of times each block is repeated in the stack. The dimensions given for filters and outputs are time, height, and width, in this order. $b_{1,\dots,4}$ are number of blocks implemented at conv2_x, conv3_x, conv4_x, conv5_x, respectively. The series of convolutions culminates with a global spatiotemporal pooling layer that yields a 512- or 2048-dimensional feature vector. This vector is fed to a fully-connected layer that outputs the class probabilities through a softmax.

that this causes overfitting.

4.1. Experimental setup

Dataset. We use Kinetics-400 [19] for the ablation experiments in this section. Kinetics is a standard benchmark for action recognition in videos. It contains about $260K$ videos of 400 different human action categories. We use the training split ($240K$ videos) for training and the validation split ($20K$ videos) for evaluating different models.

Base architecture. We use *ResNet3D*, presented in Table 1, as our base architecture for most of our ablation experiments in this section. More specifically, our model takes clips with a size of T×224×224 where $T = 8$ is the number of frames, 224 is the height and width of the cropped frame. Two spatial downsampling layers (1×2×2) are applied at `conv1` and at `pool1`, and three spatiotemporal downsampling (2×2×2) are applied at `conv3_1`, `conv4_1` and `conv5_1` via convolutional striding. A global spatiotemporal average pooling with kernel size $\frac{T}{8}\times7\times7$ is applied to the final convolutional tensor, followed by a fully-connected (fc) layer performing the final classification.

Data augmentation. We use both spatial and temporal jittering for augmentation. Specifically, video frames are scaled such that the shorter edge of the frames becomes s while we maintain the frame original aspect ratio. During training, s is uniformly sampled between 256 and 320. Each clip is then generated by randomly cropping windows of size 224×224. Temporal jittering is also applied during training by randomly selecting a starting frame and decoding T frames. For the ablation experiments in this section we train and evaluate models with clips of 8 frames ($T = 8$) by skipping every other frame (all videos are pre-processed to 30fps, so the newly-formed clips are effectively at 15fps).

Training. We train our models with synchronous distributed SGD on GPU clusters using caffe2 [2] (with 16 machines, each having 4 GPUs). We use a mini-batch of 8 clips per GPU, thus making a total mini-batch of 512 clips. Following [30], we set epoch size to 1M clips due to temporal jitterring augmentation even though the number of training examples is only about 240K. Training is done in 45 epochs where we use model warming-up [14] in the first 10 epochs and the remaining 35 epochs will follow the half-cosine period learning rate schedule as in [10]. The initial learning rate is set to 0.01 per GPU (equivalent to 0.64 for 64 GPUs).

Testing. We report clip top-1 accuracy and video top-1 accuracy. For video top-1, we use center crops of 10 clips uniformly sampled from the video and average these 10 clip-predictions to obtain the final video prediction.

4.2. Reducing FLOPs, preserving interactions

In this ablation, we use CSNs to vary both FLOPs and channel interactions. Within this architectural family, channel interactions are a good predictor of performance, whereas FLOPs are not. In particular, FLOPs can be reduced significantly while preserving interaction count.

Table 2 presents results of our interaction-reduced CSNs (ir-CSNs) and interaction-preserved CSNs (ip-CSNs) and compare them with the ResNet3D baseline using different number of layers. In the shallow network setting (with 26 layers), both the ir-CSN and the ip-CSN have lower accuracy than ResNet3D. The ir-CSN provides a computational savings of 3.6x but causes a 2.9% drop in accuracy. The ip-CSN yields a saving of 2.9x in FLOPs with a much smaller drop in accuracy (0.7%). We note that all of the shallow models have very low count of channel interactions: ResNet3D and ip-CSN have about 0.42 giga-pairs (0.42×10^9 pairs), while ir-CSN has only 0.27 giga-pairs (about 64% of the original). This observation suggests that shallow instances of ResNet3D benefit from their extra parameters, but the preservation of channel interactions reduces the gap for ip-CSN.

Conversely, in deeper settings both ir-CSNs and ip-CSNs outperform ResNet3D (by about $0.9 - 1.4\%$). Furthermore, the accuracy gap between ir-CSN and ip-CSN becomes smaller. We attribute this gap shrinking to the fact that, in the 50-layer and 101-layer configurations, ir-CSN has nearly the same number of channel interactions as ip-CSN since most interactions stem from the 1×1×1 layers. One may hypothesize that ip-CSN outperforms ResNet3D and ir-CSN because it has more nonlinearities (ReLU). To answer this question, we trained ip-CSNs without ReLUs between the 1×1×1 and the 3×3×3 layers and we observed

model	depth	video@1	FLOPs	params	interactions
		(%)	$\times 10^9$	$\times 10^6$	$\times 10^9$
ResNet3D	26	**65.3**	14.3	20.4	0.42
ir-CSN	26	62.4	4.0	1.7	0.27
ip-CSN	26	64.6	5.0	2.4	0.42
ResNet3D	50	69.4	29.5	46.9	5.68
ir-CSN	50	70.3	10.6	13.1	5.42
ip-CSN	50	**70.8**	11.9	14.3	5.68
ResNet3D	101	70.6	44.7	85.9	8.67
ir-CSN	101	71.3	14.1	22.1	8.27
ip-CSN	101	**71.8**	15.9	24.5	8.67

Table 2. **Channel-Separated Networks vs. ResNet3D**. In the 26-layer configuration, the accuracy of ir-CSN is 2.9% lower than that of the ResNet3D baseline. But ip-CSN, which preserves channel interactions, is nearly on par with the baseline (the drop is only 0.7%). In the the 50- and 101-layer configurations, both ir-CSN and ip-CSN outperform ResNet3D while reducing parameters and FLOPs. ip-CSN consistently outperforms ir-CSN.

no notable difference in accuracy. We can conclude that traditional 3×3×3 convolutions contain many parameters which can be removed without an accuracy penalty in the deeper models. We further investigate this next.

We also experimented with a space-time decomposition of the 3D filters [23, 30, 37] in ir-CSN-50. This model obtains 69.7% on Kinetics validation (vs. 70.3% of vanilla ir-CSN-50) while requiring more memory and having roughly the same GFLOPs as ir-CSN. The small accuracy drop may be due to the fact that CSN 3D filters are already channel-factorized and the space-time decomposition may limit excessively their already constrained modeling ability.

4.3. What makes CSNs outperform ResNet3D?

In section 4.2 we found that both ir-CSNs and ip-CSNs outperform the ResNet3D baseline when there are enough channel interactions, while having fewer parameters and greatly reducing FLOPs. It is natural to ask: what makes CSNs more accurate? Figure 5 provides a useful insight to answer this question. The plot shows the evolution of the training errors of ip-CSN and ResNet3D, both with 101 layers. Compared to ResNet3D, ip-CSN has higher training errors but lower testing error (see validation accuracy shown in Table 2). This suggests that the channel separation in CSN regularizes the model and prevents overfitting.

4.4. The effects of different blocks in group convolutional networks

Here we start from our base architecture (shown in Table 1) then ablatively replace the convolutional blocks with those presented in section 3.4. Again we find that channel interaction plays a critical role in understanding the results.

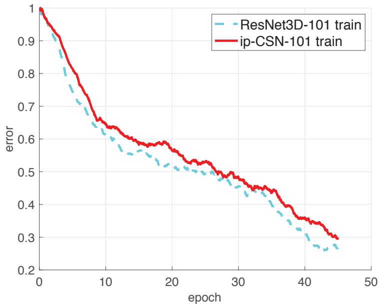

Figure 5. **Training error as a function of training iterations for ip-CSN-101 and ResNet3D-101 on Kinetics**. ip-CSN has higher training error, but lower testing error (compare validation accuracies in Table 2). This suggests that the channel separation provides a beneficial regularization, combatting overfitting.

Model	block	config	name
ResNet3D-18	simple	[2, 2, 2, 2]	simple-8
ResNet3D-50	bottleneck	[3, 4, 6, 3]	bottleneck-16

Table 3. **Naming convention**. We name architectures by block name followed by the total number of blocks (see last column). Only two block names are given in this table. More blocks are presented in section 3.4.

Naming convention. Since the ablation in this section will be considering several different convolutional blocks, to simplify the presentation, we name each architecture by block type (as presented in section 3.4) and total number of blocks, as shown in the last column of Table 3.

Figure 6 presents the results of our ablation on convolutional blocks. It shows the video top-1 accuracy on the Kinetics validation set vs the model computational cost (# FLOPs). We note that, in this experiment, we use our base architecture with two different numbers of blocks (8 and 16) and just vary the type of convolutional block and number of groups to study the tradeoffs. Figure 6(a) presents our ablation experiment with simple-X-8 and bottleneck-X-8 architectures (where X can be none, G, or D, or even DG in the case of bottleneck block). Similarly, Figure 6(b) presents our ablation experiment with simple-X-16 and bottleneck-X-16 architectures. We can observe the computation/accuracy effects of the group convolution transformation. Reading each curve from right to left (i.e. in decreasing accuracy), we see simple-X transforming from simple block to simple-G (with increasing number of groups), then to simple-D block. For bottleneck-X, reading right to left shows bottleneck block, then transforms to bottleneck-G (with increasing groups), bottleneck-D, then finally to bottleneck-DG (again with increasing groups).

While the general downward trend is expected as we decrease parameters and FLOPs, the shape of the simple and bottleneck curves is quite different. The simple-X models degrade smoothly, whereas bottleneck-X stays relatively flat (particularly bottleneck-16, which actually *increases* slightly as we decrease FLOPs) before dropping sharply.

In order to better understand the different behaviors of

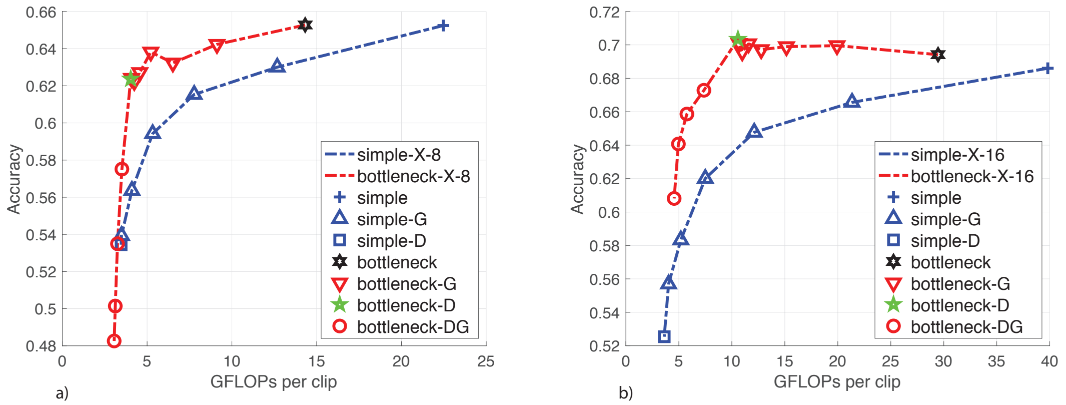

Figure 6. **ResNet3D accuracy/computation tradeoff by transforming group convolutional blocks**. Video top-1 accuracy on the Kinetics validation set against computation cost (# FLOPs) for a ResNet3D with different convolutional block designs. (a) Group convolution transformation applied to simple and bottleneck blocks with shallow architectures with 8 blocks. (b) Group convolution transformation applied to simple and bottleneck blocks with deep architectures with 16 blocks. The bottleneck-D block (marked with green starts) gives the best accuracy tradeoff among the tested block designs. Base architectures are marked with black hexagrams. Best viewed in color.

the simple-X-Y and bottleneck-X-Y models (blue vs. red curves) in Figure 6 and the reasons behind the turning points of bottleneck-D block (green start markers in Figure 6), we plot the performance of all these models according to another view: accuracy vs channel interactions (Figure 7).

As shown in Figure 7, the number of channel interactions in simple-X-Y models (blue squares and red diamonds) drops quadratically when group convolution is applied to their 3×3×3 layers. In contrast, the number of channel interactions in bottleneck-X-Y models (green circles and purple triangles) drops marginally when group convolution is applied to their 3×3×3 since they still have many 1×1×1 layers (this can be seen in the presence of two marker clusters which are circled in red: the first cluster includes purple triangles near the top-right corner and the other one includes green circles near the center of the figure). The channel interaction in bottleneck-X-Y starts to drop significantly when group convolution is applied to their 1×1×1 layers, and causes the model sharp drop in accuracy. This fact explains well why there is no turning point in simple-X-Y curves and also why there are turning points in bottleneck-X-Y curves. It also confirms the important role of channel interactions in group convolutional networks.

Bottleneck-D block (also known as ir-CSN) provides the best computation/accuracy tradeoff. For simple blocks, increasing the number of groups causes a continuous drop in accuracy. However, in the case of the bottleneck block (i.e. bottleneck-X-Y) the accuracy curve remains almost flat as we increase the number of groups until arriving at the bottleneck-D block, at which point the accuracy degrades dramatically when the block is turned into a bottleneck-DG (group convolution applied to 1×1×1 layers). We conclude that a bottleneck-D block (or ir-CSN) gives the best computation/accuracy tradeoff in this family of ResNet-style blocks, due to its high channel-interaction count.

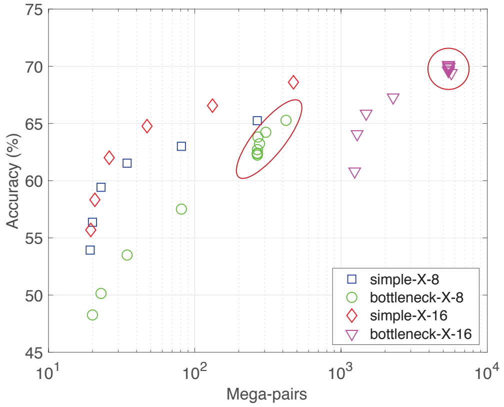

Figure 7. **Accuracy vs. channel interactions**. Plotting the Kinetics validation accuracy of different models with respect to their total number of channel interactions. Channel interactions are presented on a log scale for better viewing. Best viewed in color.

5. Comparison with the State-of-the-Art

Datasets. We evaluate our CSNs on Sports1M [18] and Kinetics-400 [19]. Sports1M is a large-scale action recognition dataset containing 1.1 million videos from 487 sport action classes. For Sports1M, we use the public train and test splits provided with the dataset. For Kinetics, we use the train split for training and the validation set for testing.

Training. Differently from our ablation experiments, here we train our CSNs with 32-frame clip inputs ($T = 32$) with a sampling rate of 2 (skipping every other frame) following the practice described in [30]. All the other training settings such as data augmentation and optimization parameters are the same as those described in our previous section.

Testing. For Sports1M, we uniformly sample 10 clips per video, scale the shorter edge to 256 (keeping aspect ratio), and use only the center crop of 224×224 per clip for inference. We average the softmax predictions of these 10 crops

Method	input	video@1	video@5	GFLOPs×crops
C3D [29]	RGB	61.1	85.2	N/A
P3D [23]	RGB	66.4	87.4	N/A
Conv Pool [38]	RGB+OF	71.7	90.4	N/A
R(2+1)D [30]	RGB	73.0	91.5	152×N/A
R(2+1)D [30]	RGB+OF	73.3	91.9	305×N/A
ir-CSN-101	RGB	74.8	92.6	56.5×10
ip-CSN-101	RGB	74.9	92.6	63.6×10
ir-CSN-152	RGB	**75.5**	**92.7**	74.0×10
ip-CSN-152	RGB	**75.5**	**92.8**	83.3×10

Table 4. **Comparison with state-of-the-art architectures on Sports1M**. Our CSNs with 101 or 152 layers outperform all the previous models by good margins while being 2-4x faster.

for video prediction. On Kinetics, since the 30 crops evaluation in [35] is widely adopted, we follow this setup for a fair comparison with previous approaches.

Results on Sports1M. Table 4 compares results of our CSNs with those of previous methods on Sports1M. Our ir-CSN-152 and ip-CSN-152 outperform C3D [29] by 14.4%, P3D [23] by 9.1%, Conv Pool [38] by 3.8%, and R(2+1)D [30] by 2.2% on video top-1 accuracy while being 2-4x faster than R(2+1)D. Our ir-CSN-101, even with a smaller number of FLOPs, still outperforms all previous work by good margins. On large-scale benchmarks like Sports1M, the difference between ir-CSN and ip-CSN is very small. The added benefit of ir-CSN is that it has smaller GFLOPs, especially in deeper settings where the number of channel interactions is similar to that of ip-CSN. This is consistent with the observation from our ablation.

Results on Kinetics. We train our CSN models on Kinetics and compare them with current state-of-the-art methods. In addition to training from scratch, we also finetune our CSNs with weights initialized from models pre-trained on Sports1M. For a fair comparison, we compare our CSNs with the methods that use only RGB as input. Table 5 presents the results. Our ip-CSN-152, even when trained from scratch, outperforms all of the previous models, except for SlowFast [10]. Our ip-CSN-152, pre-trained on Sports1M outperforms I3D [3], R(2+1)D [30], and S3D-G [37] by 8.1%, 4.9%, and 4.5%, respectively. It also outperforms recent work: A^2-Net [4] by 4.6%, Global-reasoning networks [6] by 3.1%. We note that our ip-CSN-152 achieves higher accuracy than both I3D with Non-local Networks (NL) [35] and SlowFast [10] (+1.5% and +0.3%) while being also faster (3.3x and 2x, respectively). Our ip-CSN-152 is still 0.6% lower than SlowFast augmented with Non-Local Networks. Finally, recent work [13] has shown that R(2+1)D can achieve strong performance when pre-trained on a large-scale weakly-supervised dataset. We pre-train/finetune ir-CSN-152 on the same dataset and compare it with R(2+1)D-152 (the last two rows of Table 5). In this large-scale setup, ir-CSN-152 outperforms R(2+1)D-152 by 1.3% in video top-1 accuracy while being 3.4x faster.

Method	pretrain	vi@1	vi@5	GFLOPs×crops
ResNeXt [15]	none	65.1	85.7	N/A
ARTNet(d) [32]	none	69.2	88.3	24×250
I3D [3]	ImageNet	71.1	89.3	108×N/A
TSM [22]	ImageNet	72.5	90.7	65×N/A
MFNet [5]	ImageNet	72.8	90.4	11×N/A
Inception-ResNet [1]	ImageNet	73.0	90.9	N/A
R(2+1)D-34 [30]	Sports1M	74.3	91.4	152×N/A
A^2-Net [4]	ImageNet	74.6	91.5	41×N/A
S3D-G [37]	ImageNet	74.7	93.4	71×N/A
D3D [27]	ImageNet	75.9	N/A	N/A
GloRe [6]	ImageNet	76.1	N/A	55×N/A
I3D+NL [35]	ImageNet	77.7	93.3	359×30
SlowFast [10]	none	78.9	93.5	213×30
SlowFast+NL [10]	none	**79.8**	**93.9**	234×30
ir-CSN-101	none	76.2	92.2	73.8×30
ip-CSN-101	none	76.7	92.3	83.0×30
ir-CSN-152	none	76.8	92.5	96.7×30
ip-CSN-152	none	77.8	92.8	108.8×30
ir-CSN-101	Sports1M	78.1	93.4	73.8×30
ip-CSN-101	Sports1M	78.5	93.5	83.0×30
ir-CSN-152	Sports1M	79.0	93.5	96.7×30
ip-CSN-152	Sports1M	**79.2**	**93.8**	108.8×30
R(2+1)D-152* [13]	IG-65M	81.3	95.1	329×30
ir-CSN-152*	IG-65M	**82.6**	**95.3**	96.7×30

Table 5. **Comparison with state-of-the-art architectures on Kinetics**. Accuracy is measured on the Kinetics validation set. For fair evaluation, the comparison is restricted to models trained on RGB input. Our ir-CSN-152 is better than or comparable with previous models while being multiple times faster. *Models leveraging large-scale pre-training, thus not comparable with others.

6. Conclusion

We have presented Channel-Separated Convolutional Networks (CSN) as a way of factorizing 3D convolutions. The proposed CSN-based factorization not only helps to significantly reduce the computational cost, but also improves the accuracy when there are enough channel interactions in the networks. Our proposed architecture, ir- and ip-CSN, significantly outperform existing methods and obtains state-of-the-art accuracy on two major benchmarks: Sports1M and Kinetics. The model is also multiple times faster than current competing networks. We have made code and pre-trained models publicly available [9].

Acknowledgements. We thank Kaiming He for insightful discussions and Haoqi Fan for help in improving our training framework.

References

[1] Yunlong Bian, Chuang Gan, Xiao Liu, Fu Li, Xiang Long, Yandong Li, Heng Qi, Jie Zhou, Shilei Wen, and Yuanqing Lin. Revisiting the effectiveness of off-the-shelf temporal modeling approaches for large-scale video classification. *CoRR*, abs/1708.03805, 2017. 8

[2] Caffe2-Team. Caffe2: A new lightweight, modular, and scalable deep learning framework. `https://caffe2.ai/`. 5

[3] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, 2017. 1, 2, 8

[4] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. Aˆ2-nets: Double attention networks. In *NeuIPS*, pages 350–359, 2018. 8

[5] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. Multi-fiber networks for video recognition. In *ECCV*, 2018. 2, 8

[6] Yunpeng Chen, Marcus Rohrbach, Zhicheng Yan, Shuicheng Yan, Jiashi Feng, and Yannis Kalantidis. Graph-based global reasoning networks. In *CVPR*, 2019. 8

[7] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 1, 2, 3

[8] Piotr Dollar, Vincent Rabaud, Garrison Cottrell, and Serge Belongie. Behavior recognition via sparse spatio-temporal features. In *Proc. ICCV VS-PETS*, 2005. 2

[9] Facebook. Video model zoo. `https://github.com/facebookresearch/VMZ`, 2018. 8

[10] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *ICCV*, 2019. 5, 8

[11] Christoph Feichtenhofer, Axel Pinz, and Richard P. Wildes. Spatiotemporal residual networks for video action recognition. In *NIPS*, 2016. 2

[12] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *CVPR*, 2016. 2

[13] Deepti Ghadiyaram, Matt Feiszli, Du Tran, Xueting Yan, Heng Wang, and Dhruv Mahajan. Large-scale weakly-supervised pre-training for video action recognition. In *CVPR*, 2019. 8

[14] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 5

[15] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *CVPR*, 2018. 2, 8

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2, 4

[17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 1, 2, 3

[18] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014. 2, 7

[19] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017. 2, 5, 7

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 2

[21] Ivan Laptev and Tony Lindeberg. Space-time interest points. In *ICCV*, 2003. 2

[22] Ji Lin, Chuang Gan, and Song Han. Temporal shift module for efficient video understanding. *CoRR*, abs/1811.08383, 2018. 8

[23] Zhaofan Qiu, Ting Yao, and Tao Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *ICCV*, 2017. 1, 2, 3, 6, 8

[24] Sreemananath Sadanand and Jason Corso. Action bank: A high-level representation of activity in video. In *CVPR*, 2012. 2

[25] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. 2

[26] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014. 2

[27] Jonathan C. Stroud, David A. Ross, Chen Sun, Jia Deng, and Rahul Sukthankar. D3D: distilled 3d networks for video action recognition. *CoRR*, abs/1812.08249, 2018. 8

[28] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 2

[29] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015. 2, 8

[30] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *CVPR*, 2018. 1, 2, 3, 5, 6, 7, 8

[31] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *ICCV*, 2013. 2

[32] Limin Wang, Wei Li, Wen Li, and Luc Van Gool. Appearance-and-relation networks for video classification. In *CVPR*, 2018. 2, 8

[33] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *ECCV*, 2016. 2

[34] Xiaolong Wang, Ali Farhadi, and Abhinav Gupta. Actions ˜ transformations. In *CVPR*, 2016. 2

[35] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018. 1, 2, 8

[36] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017. 2, 4

[37] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning for video understanding. In *ECCV*, 2018. 1, 2, 3, 6, 8

[38] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015. 8

[39] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 1, 2, 4