2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)

Resource Efficient 3D Convolutional Neural Networks

Okan Köpüklü[1], Neslihan Kose[2], Ahmet Gunduz[1], Gerhard Rigoll[1]

[1] Institute for Human-Machine Communication, TU Munich, Germany

[2] Dependability Research Lab, Intel Labs Europe, Intel Deutschland GmbH, Germany

Abstract

Recently, convolutional neural networks with 3D kernels (3D CNNs) have been very popular in computer vision community as a result of their superior ability of extracting spatio-temporal features within video frames compared to 2D CNNs. Although there has been great advances recently to build resource efficient 2D CNN architectures considering memory and power budget, there is hardly any similar resource efficient architectures for 3D CNNs. In this paper, we have converted various well-known resource efficient 2D CNNs to 3D CNNs and evaluated their performance on three major benchmarks in terms of classification accuracy for different complexity levels. We have experimented on (1) Kinetics-600 dataset to inspect their capacity to learn, (2) Jester dataset to inspect their ability to capture motion patterns, and (3) UCF-101 to inspect the applicability of transfer learning. We have evaluated the run-time performance of each model on a single Titan XP GPU and a Jetson TX2 embedded system. The results of this study show that these models can be utilized for different types of real-world applications since they provide real-time performance with considerable accuracies and memory usage. Our analysis on different complexity levels shows that the resource efficient 3D CNNs should not be designed too shallow or narrow in order to save complexity. The codes and pretrained models used in this work are publicly available [1].

1. Introduction

Ever since AlexNet [18] won the ImageNet Challenge (ILSVRC 2012 [24]), convolutional neural networks (CNNs) have dominated the majority of the computer vision tasks. Then the primary trend has been more on creating deeper and wider CNN architectures to achieve higher accuracies [10, 26, 29]. However, in real world computer vision applications such as face recognition, robot navigation and augmented reality, the tasks need to be carried out under runtime constraints on a computationally limited platform. Only recently, there has been a rising interest in building resource efficient convolutional neural networks but it is limited with 2-dimensional kernels (2D) [13, 11, 37, 20, 25].

The same history is repeating for CNNs with 3-dimensional (3D) kernels [9]. Since the large video datasets became available, the primary trend for video recognition tasks is again to achieve higher accuracies by building deeper and wider architectures [31, 22, 32, 9, 6]. Considering the fact that 3D CNNs achieve better performance for video recognition tasks compared to 2D CNNs [3], it is very likely that this 3D CNN architecture search will continue until the achieved accuracies saturate. However, real-world applications still require resource efficient 3D CNN architectures taking runtime, memory and power budget into account. This work aims to fill this research gap.

In this paper, we first have created the 3D versions of the well-known 2D resource efficient architectures: SqueezeNet, MobileNet, ShuffleNet, MobileNetV2 and ShuffleNetV2. We have evaluated t-he performance of these architectures on three publicly available benchmarks:

(1) Kinetics-600 dataset[3] to learn models' capacities.

(2) Jester dataset [1] to learn how well the models capture the motion.

(3) UCF-101 dataset [27] to evaluate the applicability of transfer learning for each model.

The computational complexity of the implemented architectures are measured in terms of floating point operations (FLOPs), which is widely used metric among resource efficient architectures. In this paper, the number of FLOPs refers to the number of multiply-adds. However, as highlighted by [20], the number of FLOPs is an indirect metric which does not give an actual performance indication like speed or latency. Therefore, for all the implemented architectures we have also evaluated their run-time performance on two different platforms, which are Nvidia Titan XP GPU and Jetson TX2 embedded system-on-module (SoM) with integrated 256-core Pascal GPU.

[1] https://github.com/okankop/Efficient-3DCNNs

2473-9944/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCVW.2019.00240

2. Related Work

Lately, there is a rising interest in building small and efficient neural networks [13, 11, 20, 23, 34, 7]. The common approaches used for this objective can be categorized under two categories: (i) Accelerating the pretrained networks, or (ii) directly constructing small networks by manipulating kernels. For the first one, [7, 8, 33, 21] proposes to prune either network connections or channels without reducing the performance of pretrained models. Additionally, many other methods apply quantization [23, 28, 34] or factorization [19, 14, 15] for the same objective. However, our focus is on the second one for directly designing small and resource efficient 3D CNN architectures.

Current well-known resource efficient CNN architectures are all constructed with 2D convolutional kernels and benchmarked at ImageNet. SqueezeNet [13] reduced the number of parameters and computation while maintaining the classification performance. MobileNet [11] makes use of depthwise separable convolutions to construct lightweight deep neural networks. The depthwise separable convolutions factorize the standard convolutions into a depthwise convolution followed by a 1x1 pointwise convolution. Compared to standard convolutions, depthwise separable convolutions use between 8 to 9 times less parameters and computations. ShuffleNet [37] proposes to use pointwise group convolutions and channel shuffle in order to reduce computational cost. MobileNetv2 [25] makes use of the inverted residual structure where the intermediate expansion layer uses depthwise convolutions. ShuffleNetV2 [20] builds on top of ShuffleNet [37] using channel split together with channel shuffle which realizes a feature reuse pattern.

These architectures intensively make use of group convolutions and depthwise separable convolutions. Group convolutions are first introduced in AlexNet [18] and efficiently utilized in ResNeXt [35]. Depthwise separable convolutions are introduced in Xception [5] and they are the main building blocks for majority of lightweight architectures.

All of the above-mentioned resource efficient architectures are 2D CNNs. They are designed to operate on static images and evaluated on a very large benchmark (i.e., ImageNet). To the best of our knowledge, this is the first work that proposes and evaluates resource efficient 3D CNNs on large scale video benchmarks.

3D CNNs such as well-known C3D [30] require significantly more parameters and computations compared to their 2D counterparts which make them harder to train and prone to overfitting. With the availability of large scale video datasets such as Sports-1M [16], Kinetics-400 [3], this problem is solved. Moreover, [3] proved that 3D CNNs achieve better accuracies compared to 2D CNNs for video classification task. Consequently, 3D CNN architecture search is an active area in research community to achieve higher accuracies.

Several 3D CNN architectures have been proposed recently. Carreira et al. propose Inflated 3D CNN (I3D) [3], where the filters and pooling kernels of a deep CNN are expanded to 3D, making it possible to leverage successful ImageNet architecture designs and their pretrained models. P3D [22] and (2+1)D [32] propose to decompose 3D convolutions into 2D and 1D convolutions operating on spatial and depth dimensions, respectively. In [9], 3D versions of famous ImageNet architectures such as ResNet [10], Wide ResNet [36], ResNeXt [35] and DenseNet [12] are evaluated and it has been shown that ResNeXt achieves better results compared to others. Recently, Feichtenhofer et al. propose a novel architecture named SlowFast [6], which uses a Slow pathway, operating at low frame rate, to capture static content of a video, and a Fast pathway, operating at high frame rate, to capture the dynamic content of a video.

Up to now, nearly all the 3D CNN architectures in the literature are heavyweight, requiring 10s and even 100s billions of floating point operations (FLOPs). Moreover, majority of these architectures also use optical flow modality, which increases the complexity even further. Our focus in this work is to evaluate 3D CNNs having less than 500 MFLOPs. Consequently, we have implemented the 3D version of SqueezeNet [13], MobileNet [11], MobileNetV2 [25], ShuffleNet [37] and ShuffleNetV2 [20] for 4 different complexity levels and then evaluated them on 3 different video benchmarks. We have evaluated our architectures only using RGB modality without computing costly optical flow modality.

3. Resource Efficient 3D CNN Architectures

In this section, we explain the details of the resource efficient 3D CNN architectures that have been proposed and evaluated within the scope of this work. We initially introduce the 3D versions of the well-know resource efficient 2D CNN architectures by explaining their building blocks and networks structures. Then we compare these models in terms of number of layers, nonlinearities, and skip connections. We conclude with training details of the models.

3.1. 3D Versions of Well-known Architectures

In this section, we give the implementation details of our resource efficient architectures with 3-dimensional kernels, which are converted from well-know resource efficient 2D CNN architectures. Main building blocks of each architecture are depicted in Fig. 1. The input is always considered as a clip of 16 frames with spatial resolution of 112 pixels. For all of the 3D CNN architectures, first convolutions always apply stride of (1,2,2). For the rest of the architectures, depth dimension is reduced together with spatial dimensions.

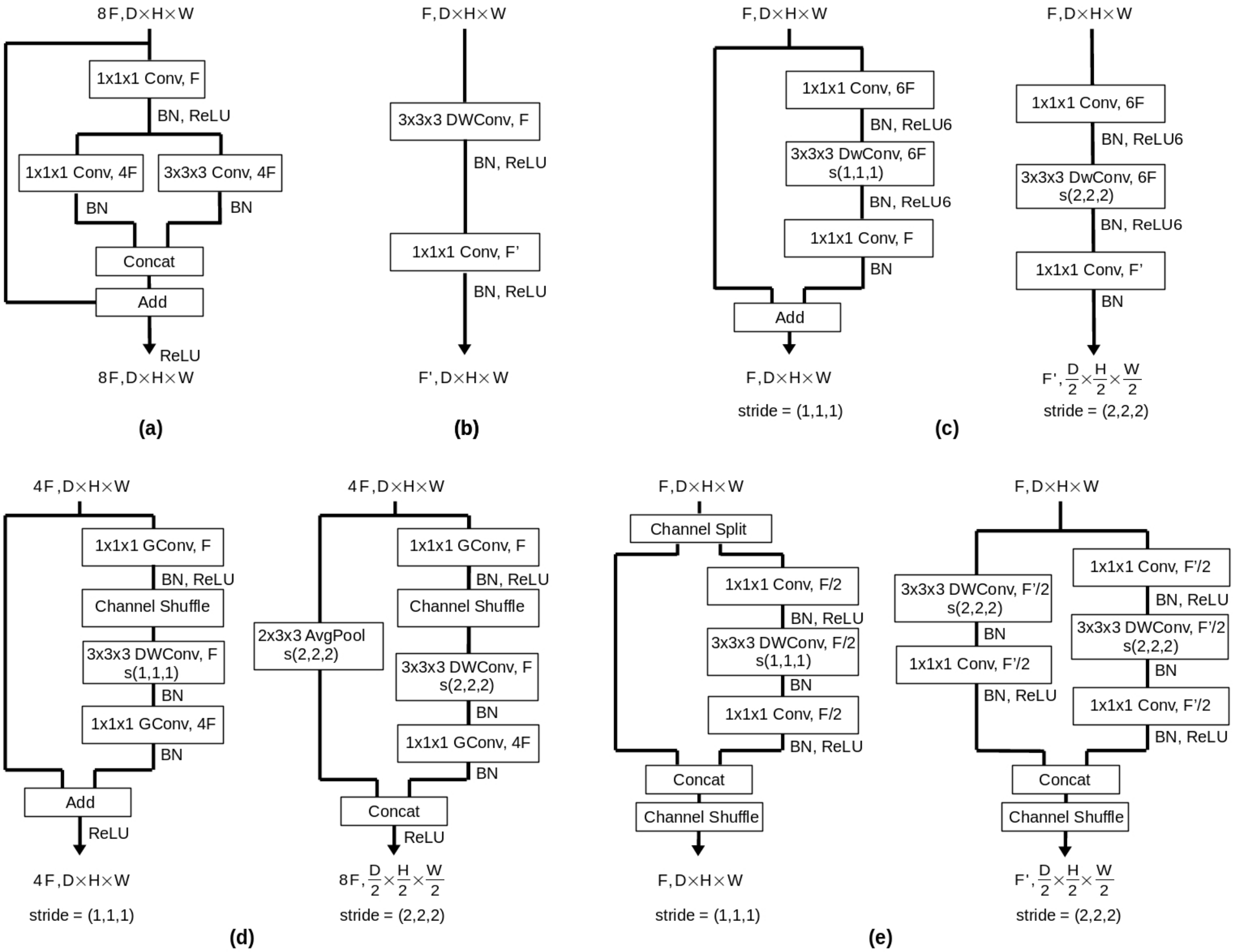

Figure 1: Main building block for each resource efficient 3D CNN architecture. *F* is the number of feature maps and D × H × W stands for Depth × Height × Width for the input and output volumes. *DWConv* and *GConv* stand for depthwise and group convolution, respectively. *BN* and *ReLU(6)* stand for Batch Normalization and Rectified Linear Unit (capped at 6), respectively. **(a)** SqueezeNet's Fire block; **(b)** MobileNet block; **(c)** left: MobileNetv2 block, right: MobileNetv2 block with spatiotemporal downsampling (2x); **(d)** left: ShuffleNet block, right: ShuffleNet block with spatiotemporal downsampling (2x); **(e)** left: ShuffleNetv2 block, right: ShuffleNetv2 block with spatiotemporal downsampling (2x).

3.1.1 3D-SqueezeNet

SqueezeNet [13] is considered as one of very first resource efficient CNN architectures with notable accuracy performance. It achieves the AlexNet [18]-level accuracy with 50 times fewer parameters and less than 0.5 MB model size.

The main building block of SqueezeNet is Fire block whose 3D version is depicted in Fig. 1 (a). As illustrated in Table 1, 3D-SqueezeNet begins with a convolution layer (Conv1), followed by 8 Fire blocks (Fire-2-9), ending with a final convolutional layer (Conv10).

In our experiments, we use SqueezeNet with simple bypass since it achieves the best result in its 2D version for ImageNet. SqueezeNet does not apply depthwise convolutions which is the main building block for majority of resource efficient architectures. Instead, it uses three strategies to reduce the number of parameters while maintaining accuracy: (i) Replacing 3x3 filters with 1x1 filters, (ii) decreasing the number of input channels to 3x3 filters, and (iii) downsampling late in the network so that convolution layers have large activation maps. Moreover, compared to other resource efficient architectures, SqueezeNet cannot be modified with $width\_multiplier$ parameter resulting in different complexities. Therefore, it is only experimented with its default configuration as shown in Table 8.

3.1.2 3D-MobileNetV1

MobileNets [11] apply depthwise separable convolutions which have a form that factorize a standard convolution

Layer / Stride	Filter size	Output size
Input clip		3x16x112x112
Conv1/s(1,2,2)	3x3x3	64x16x56x56
MaxPool/s(2,2,2)	3x3x3	64x8x28x28
Fire2		128x8x28x28
Fire3		128x8x28x28
MaxPool/s(2,2,2)	3x3x3	128x4x14x14
Fire4		256x4x14x14
Fire5		256x4x14x14
MaxPool/s(2,2,2)	3x3x3	256x2x7x7
Fire6		384x2x7x7
Fire7		384x2x7x7
MaxPool/s(2,2,2)	3x3x3	384x1x4x4
Fire8		512x1x4x4
Fire9		512x1x4x4
Conv10/s(1,1,1)	1x1x1	*NumCls*x1x4x4
AvgPool/s(1,1,1)	1x4x4	*NumCls*

Table 1: 3D-SqueezeNet architecture. Details of Fire block is given in Fig. 1 (a).

into a depthwise convolution and 1 × 1 convolution, which is called as pointwise convolution. In MobileNet architectures, the depthwise convolution applies a single filter to each input channel and then the pointwise convolution applies a 1 × 1 convolution to combine the outputs of the depthwise convolution. Different from the standard convolution, the depthwise separable convolution involves two layers which separates filtering and combining operations as illustrated in Fig. 1 (b). This process helps to decrease computation time and model size significantly. Unlike all recent popular CNN architectures, MobileNet does not contain skip connections. Therefore, depth of the network cannot be increased too much which hinders gradient flow.

Table 2 shows the details of the 3D-MobileNet architecture. 3D-MobileNet begins with a convolutional layer, followed by 13 MobileNet blocks, ending with a linear layer. MobileNet has 28 layers in case the depthwise and pointwise convolutions in each MobileNet block are counted as separate layers.

3.1.3 3D-MobileNetV2

MobileNetV2 [25] is another 2D resource efficient architecture. It builds upon the main idea of MobileNetV1 by using depthwise separable convolutions; however, it introduces two new components: 1) linear bottlenecks between the layers, and 2) shortcut connections between the bottlenecks. The idea behind 1) is both keeping the size of model low by decreasing number of channels and extracting as much as information by applying depthwise convolution after decompressing the data. This convolutional module allows to reduce memory usage during inference. On the other hand, 2) allows training faster and construct deeper models like ResNet architectures [10].

Layer / Stride	Repeat	Output size
Input clip		3x16x112x112
Conv(3x3x3)/s(1,2,2)	1	32x16x56x56
Block/s(2x2x2)	1	64x8x28x28
Block/s(2x2x2)	1	128x4x14x14
Block/s(1x1x1)	1	128x4x14x14
Block/s(2x2x2)	1	256x2x7x7
Block/s(1x1x1)	1	256x2x7x7
Block/s(2x2x2)	1	512x1x4x4
Block/s(1x1x1)	5	512x1x4x4
Block/s(1x1x1)	1	1024x1x4x4
Block/s(1x1x1)	1	1024x1x4x4
AvgPool(1x4x4)/s(1,1,1)	1	1024x1x1x1
Linear(1024x*NumCls*)	1	*NumCls*

Table 2: 3D-MobileNet architecture. Details of Block is given in Fig. 1 (b).

Fig. 1 (c) shows the MobileNetV2 block. Table 3 shows the layers of 3D-MobileNetV2 architecture. 3D-MobileNetV2 begins with a convolutional layer, followed by 17 MobileNetV2 blocks, and then a convolutional layer and finally ending with a linear layer.

3.1.4 3D-ShuffleNetV1

According to [37], ShuffleNet provides superior performance compared to MobileNet [11] by a significant margin, which is reported as absolute 7.8% lower ImageNet top-1 error at level of 40 MFLOPs. The model is also reported to achieve 13× actual speedup over AlexNet while maintain-

Layer / Stride	Repeat	Output size
Input clip		3x16x112x112
Conv(3x3x3)/s(1,2,2)	1	32x16x56x56
Block/s(1x1x1)	1	16x16x56x56
Block/s(2x2x2)	2	24x8x28x28
Block/s(2x2x2)	3	32x4x14x14
Block/s(2x2x2)	4	64x2x7x7
Block/s(1x1x1)	3	96x2x7x7
Block/s(2x2x2)	3	160x1x4x4
Block/s(1x1x1)	1	320x1x4x4
Conv(1x1x1)/s(1,1,1)	1	1280x1x4x4
AvgPool/s(1,1,1)	1	1024x1x1x1
Linear	1	*NumCls*

Table 3: 3D-MobileNetV2 architecture. Block is inverted residual block whose details are given in Fig. 1 (c) with stride 1 (left) and spatio temporal 2x downsampling (right).

Layer / Stride	**Repeat**	**Output size (groups=3)**
Input clip		3x16x112x112
Conv(3x3x3)/s(1,2,2)	1	24x16x56x56
MaxPool(3x3x3)/s(2,2,2)	1	24x8x28x28
Block/s(2x2x2)	1	240x4x14x14
Block/s(1x1x1)	3	240x4x14x14
Block/s(2x2x2)	1	480x2x7x7
Block/s(1x1x1)	7	480x2x7x7
Block/s(2x2x2)	1	960x1x4x4
Block/s(1x1x1)	3	960x1x4x4
AvgPool(1x4x4)/s(1,1,1)	1	960x1x1x1
Linear	1	*NumCls*

Table 4: 3D-ShuffleNet architecture. Its' main building block is given in Fig. 1 (d) with stride 1 (left) and spatio temporal 2x downsampling (right).

ing comparable accuracy.

The architecture uses two new operations, which are pointwise group convolution and channel shuffle which is depicted in Fig. 1 (d).

As illustrated in Table 4, 3D-ShuffleNet begins with a convolutional layer followed by 16 ShuffleNet blocks, which are grouped into three stages. In each stage, the number of output channels are kept same with the applied ShuffleNet blocks. For the next stage, the output channels are doubled and the spatial and depth dimensions are reduced to half. ShuffleNet architecture ends with a final linear layer. In ShuffleNet units, group number g controls the connection sparsity of pointwise convolutions. In this study, the group number is selected as 3.

3.1.5 3D-ShuffleNetV2

In ShuffleNetV2 [20] architecture, channel split operator is introduced different from V1. As illustrated in Fig. 1 (e), at the beginning of each block, the input of c feature channels are split into two branches with c-$c^{'}$ and $c^{'}$ channels, respectively. One branch remains as identity, and the other branch includes three convolutions with the same input and output channels. Different from ShuffleNet, the two 1×1 convolutions are not groupwise. After the convolutions, the two branches are concatenated and the number of channels keeps the same. At the end of the block, channel shuffle operation is applied to enable information communication between the two branches.

Table 5 shows the layers of 3D-ShuffleNetV2 architecture. 3D-ShuffleNetV2 architecture begins with a convolutional layer, followed by 16 ShuffleNetV2 blocks, and then a convolutional layer and finally ending with a linear layer. Similar to 3D-ShuffleNet, the stack of blocks are grouped into three stages, and at each stage the number of output channels are kept same while with the next stage, they are doubled. Different from the 3D-ShuffleNet, the number of channels in each stage are not fixed. Table 6 shows the number of channels (c_1, c_2, c_3, c_4) for different levels of complexities. Also, in 3D-ShuffleNet, the number of output channels in the final layer (c_4) is same after the third stage, whereas in 3D-ShuffleNetV2, different number of output channels are selected for different levels of complexities (Table 6).

Layer / Stride	**Repeat**	**Output size**
Input clip		3x16x112x112
Conv(3x3x3)/s(1,2,2)	1	24x16x56x56
MaxPool(3x3x3)/s(2,2,2)	1	24x8x28x28
Block/s(2x2x2)	1	c_1x4x14x14
Block/s(1x1x1)	3	c_1x4x14x14
Block/s(2x2x2)	1	c_2x2x7x7
Block/s(1x1x1)	7	c_2x2x7x7
Block/s(2x2x2)	1	c_3x1x4x4
Block/s(1x1x1)	3	c_3x1x4x4
Conv(1x1x1)/s(1,1,1)	1	c_4x1x4x4
AvgPool(1x4x4)/s(1,1,1)	1	c_4x1x1x1
Linear	1	*NumCls*

Table 5: 3D-ShuffleNetV2 architecture. Its' main building block is given in Fig. 1 (e) with stride 1 (left) and spatio temporal 2x downsampling (right). The number of channels (c_1, c_2, c_3, c_4) for different complexities are given in Table 6.

	Output channels				
	0.25x	**0.5x**	**1.0x**	**1.5x**	**2.0x**
c_1	32	48	116	176	244
c_2	64	96	232	352	488
c_3	128	192	464	704	976
c_4	1024	1024	1024	1024	2048

Table 6: The number of channels used in 3D-ShuffleNetv2 architecture for different levels of complexities.

3.1.6 Comperative Analysis

In this section, we compare the experimented architectures according to the number of layers, nonlinearities and skip connections. These design criteria plays an important role for the performance of the architectures. Comparison of the architectures are given in Table 7. For the number of layers, we counted the convolutional and linear layers. For the skip-connections, we have counted the addition or concatenation operations in the architectures. Finally, for the number of non-linearity, we have counted the ReLU operations in one inference time since it is the only non-linearity used for all the architectures.

It is noticeable that comparatively earlier architectures

Model	Number of		
	layers	non-lin.	skip-con.
3D-SqueezeNet	18	18	4
3D-ShuffleNetV1	50	33	16
3D-ShuffleNetV2	51	34	16
3D-MobileNetV1	28	27	0
3D-MobileNetV2	53	35	10

Table 7: Comparison of resource efficient 3D architectures according to the number of layers, non-linearity and skip-connections.

(i.e. SqueezeNet and MobileNetV1) have smaller number of layers, non-linearity and skip-connections. On the other hand, recent resource efficient architectures (i.e. ShuffleNetV1, ShuffleNetV2 and MobileNetV2) are deeper, in the order of 50 layers and 30 non-linearity. Corollary, they require more skip connections in order to facilitate better gradient update mechanism.

3.2. Training Details

Learning: For the training of the architectures, Stochastic Gradient Descent (SGD) with standard categorical cross-entropy loss is applied. For mini-batch size of SGD, largest fitting batch size is selected, which is usually in the order of 128 videos. The momentum, dampening and weight decay are set to 0.9, 0.9 and 1x10^{-3}, respectively. When the networks are trained from scratch, learning rate is initialized with 0.1 and reduced 3 times with a factor of 10^{-1} when the validation loss converges. For the training of UCF-101 benchmark, we have used the pretrained models of Kinetics-600. We have frozen the network parameters and fine-tuned only the last layer. For fine-tuning, we start with a learning rate of 0.01 and reduced it two times after 30^{th} and 45^{th} epochs with a factor of 10^{-1} and optimization is completed after 15 more epochs.

Regularization: Although Kinetics-600 and Jester are very large benchmarks and immune to over-fitting, UCF-101 still requires intensive regularization. Weight decay of 1x10^{-3} is applied for all the parameters of the network. A dropout layer is applied before the final conv/linear layer of the networks. While dropout ratio is kept at 0.2 for Kinetics-600 and Jester, it is increased to 0.9 for UCF-101.

Augmentation: For temporal augmentation, input clips are selected from a random temporal position in the video clip. If the video contains smaller number of frames than the input size, loop padding is applied. For the input to the networks, always 16-frame clips are used. For Jester benchmark, it is critical to capture the full content of the gesture video in the selected input clip. Therefore, we have applied downsampling of 2 by selected 16 frames from 32 frames for Jester benchmark [17].

For spatial augmentation, we have selected a random spatial position from the input video. Moreover, we have selected a scale randomly from $\{1, \frac{1}{2^{1/4}}, \frac{1}{2^{3/4}}, \frac{1}{2}\}$ in order to perform multi-scale cropping as in [9]. For Kinetics-600 and UCF-101, input clips are flipped with 50% probability. After the augmentations, input clip to the network has the size of 3 x 16 x 112 x 112 referring to number of input channels, frames, width and height pixels, respectively.

Recognition: For Kinetics-600 and UCF-101, we select non-overlapping 16-frame clips from each video sample. Then center cropping with scale 1 is applied to each clip. Using the pretrained models, class scores for each clip is calculated. For each video, we average the scores of all clips. The class with the highest score indicates the class label of the video.

Implementation: Network architectures are implemented in PyTorch and trained with a single Titan Xp GPU.

4. Experiments

In this section, we first explain the experimented datasets. Then, we discuss about the achieved results for the experimented network architectures together with their run-time performance on both NVIDIA Titan Xp and Jetson TX2 embedded system.

4.1. Datasets

• **Kinetics-600 dataset** is an extension of Kinetics-400 dataset, which contains 600 human action classes, with at least 600 video clips for each action. Each clip is approximately 10 seconds long and is taken from a different YouTube video. There are in total 392,622 training videos. For each class, there are also 50 and 100 validation and test videos, respectively. Since the labels for the test set is not publicly available, we have conducted our experiments on the validation set.

We selected Kinetics-600 benchmark in order to evaluate the capacity of the experimented networks. It is very rare that a real-life application tries to classify 600 different classes. However, these kind of very large-scale datasets are very useful to evaluate the capacity of the networks to learn. Although it is still necessary to capture the motion patterns in the video, the network should especially capture the spatial content in order to identify the correct class label of the video. For example, there are 9 different ”eating something” classes where ”something” is one of ”burger, cake, carrot, chips, doughnut, hotdog, ice cream, spaghetti, watermelon”. Although ”eating” action is same for all these, the true label can only be identified when the network captures discriminative features of what is being eaten.

• **Jester dataset** is currently the largest available hand gesture dataset. In each video sample of the dataset, a person performs pre-defined hand gestures in front of a laptop camera or webcam. There are in total 148,092 gesture videos under 27 classes. The dataset is divided into three subsets:

Model	**MFLOPs**	**Params**	**Speed (cps)**		**Accuracy (%)**		
			Titan XP	**Jetson TX2**	**Kinetics-600**	**Jester**	**UCF-101**
3D-ShuffleNetV1 0.5x	42	0.55M	398	69	35.51	89.23	64.39
3D-ShuffleNetV2 0.25x	42	0.83M	442	82	25.73	86.91	56.52
3D-MobileNetV1 0.5x	46	1.17M	290	57	31.74	87.61	62.17
3D-MobileNetV2 0.2x	42	0.96M	357	42	24.14	86.43	55.56
3D-ShuffleNetV1 1.0x	125	1.52M	269	49	45.31	92.27	76.00
3D-ShuffleNetV2 1.0x	119	1.91M	243	44	46.10	91.96	77.90
3D-MobileNetV1 1.0x	137	3.91M	164	31	40.07	90.81	70.95
3D-MobileNetV2 0.45x	126	1.40M	203	19	36.47	90.21	68.31
3D-ShuffleNetV1 1.5x	235	2.92M	204	31	52.75	93.12	81.73
3D-ShuffleNetV2 1.5x	215	3.16M	186	34	52.05	93.16	82.32
3D-MobileNetV1 1.5x	273	8.22M	116	19	48.24	91.28	76.00
3D-MobileNetV2 0.7x	245	2.05M	130	13	45.59	93.34	77.32
3D-ShuffleNetV1 2.0x	393	4.78M	161	24	56.84	93.54	84.96
3D-ShuffleNetV2 2.0x	360	6.64M	146	26	55.17	93.71	83.32
3D-MobileNetV1 2.0x	454	14.10M	88	15	48.53	92.56	76.18
3D-MobileNetV2 1.0x	446	3.12M	93	9	50.65	94.59	81.60
3D-SqueezeNet	728	2.15M	682	46	40.52	90.77	74.94
ResNet-18	5557	33.24M	334	17	57.65	93.34	80.09
ResNet-50	6782	44.24M	183	11	63.00	93.70	88.92
ResNet-101	10612	83.29M	142	8	64.18	94.10	87.02
ResNeXt-101	6932	48.34M	122	7	68.30	94.89	89.08
I3D [2]	88202	12.90M	—	—	71.90	—	—

Table 8: Comparison of resource efficient 3D architectures over video classification accuracy, number of parameters and speed on two different platforms and four levels of computation complexity. The calculations of MFLOPs, parameters and speeds are done for Kinetics-600 benchmark. For speed calculations (clips per second (cps)), the used platforms are Titan Xp and Jetson TX2; and the batch size is set to 8. All models takes 16 frames input with 112 x 112 spatial resolution except for I3D, which takes 64 frames input with 224 x 224 spatial resolution.

training set (118,562 videos), validation set (14,787 videos), and test set (14,743 videos). Since the labels for test set is not publicly available, we have conducted our experiments on the validation set.

Unlike Kinetics-600 benchmark, in Jester dataset, spatial content of the all video samples are same: A person sitting in front of a camera performs a hand gesture from almost the same distance. Moreover, the selection of classes are more focused on the movement of the hand. That is why, Jester benchmark is suitable to inspect the ability of the networks in capturing motion patterns.

• **UCF101 dataset** is an action recognition dataset of realistic action videos, collected from YouTube. It consists of 101 action classes, over 13k clips and 27 hours of video data. Compared to Kinetics-600 and Jester datasets, UCF-101 contains very little amount of training videos, hence prone to over-fitting. For the evaluation of UCF-101 dataset, we have used only split-1. We selected UCF-101 benchmark in order to inspect the applicability of transfer learning for the experimented network architectures.

4.2. Results

In this section, we elaborate on our findings in the experiments that we have conducted for 5 different network architectures, 4 levels of complexity (except for SqueezeNet) on 3 different benchmarks. Moreover, runtime performance of the models are evaluated on 2 different platforms, namely Titan XP and Jetson TX2 embedded system. According to the results in Table 8, the following conclusions can be inferred:

Accuracy:

(i) The deeper architectures (3D-ShuffleNet, 3D-ShuffleNetV2, 3D-MobileNetV2) achieve better results compared to shallower architectures (3D-SqueezeNet, 3D-MobileNetV1). Accordingly, resource efficient 3D CNNs should not be designed too shallow in order to save complexity.

(ii) Motion patterns are better captured with depthwise convolutions. Since depthwise convolutions have kernels of 3x3x3, they can capture relations in depth dimension together with spatial dimension. The main building block of 3D-MobileNetV2 is the inverted residual block, which expands the number of channels to the input of depthwise convolution layers with an expansion ratio. Therefore, it contains more depthwise convolution filters compared to other architectures. Consequently, it achieves by far best performance in Jester benchmark, although it has inferior results in Kinetics-600 and UCF-101 benchmarks.

(iii) All models showed comparatively similar performance on both Kinetics-600 and UCF-101 datasets. This shows transfer learning is a valid approach for resource efficient 3D CNNs since there is a direct correlation between model performances on these two datasets.

Complexity level:

(iv) There is a severe performance degradation if the networks are scaled with very small $width\_multiplier$ in order to satisfy the required computational complexity. For example, in the first block of the Table 8, we can see that 3D-MobileNetV2 0.2x and 3D-ShuffleNetV2 0.25x achieve 5-9% worse than 3D-ShuffleNetV1 0.5x and 3D-MobileNetV1 0.5x in Kinetics-600 benchmark. Capacity of the models degrades severely as the $width\_multiplier$ gets smaller, especially when it is less than 0.5. We can see the same pattern on all three benchmarks that we have experimented.

(v) The main design criteria of the 3D-SqueezeNet is to save number of parameters, not computations. Therefore it has the smallest number of parameters at the highest complexity level. However, it also has around 300 million more FLOPs compared to other architectures since it does not make use of depthwise convolutions.

Runtime performance:

(vi) Although the network architectures contain similar FLOPs, some architectures are much faster than others. As highlighted by [20], this is due to several other factors affecting speed such as memory access cost (MAC) and degree of parallelism, which are not taken into account by FLOPs.

(vii) 3D-SqueezeNet is the only architecture that does not make use of depthwise convolutions, hence contains highest FLOPs. However, surprisingly it has the highest runtime performance. This is due to the latest CUDNN [4] library which is specifically optimized for standard convolutions. Similar results can also be observed with ResNet and ResNeXt architectures.

(viii) Runtime performance heavily depends on the hardware that the network architecture is running. For example, for the highest two complexity levels, 3D-ShuffleNetV1 is the faster than 3D-ShuffleNetV2 on GPU, whereas 3D-ShuffleNetV2 achieves higher runtime than 3D-ShuffleNetV1 on Jetson TX2.

State-of-the-art comparison:

(ix) Architectures with more parameters and FLOPs like ResNets, ResNeXt-101 and I3D achieve generally better results for datasets measuring the capacity of the tested architectures like Kinetics dataset as evaluated and shown in Table 8. However, network design makes a huge difference. For example, 3D-ShuffleNetV1 2.0x achieves similar performance with ResNet-18, although ResNet-18 requires 7 times more parameters and 14 times FLOPs .

(x) The architecture design should be done according to the given task. As inverted residual block excels at capturing dynamic motions, 3D-MobileNetV2 1.0x achieves better results than much wider and deeper ResNet-101 (around 20 times more parameters and FLOPs) at Jester benchmark.

5. Conclusion

In recent years, the research in action recognition has mostly focused on obtaining the best accuracy by generating deep and wide CNN architectures. However, real-world applications require resource efficient architectures that take runtime, memory and power budget into account. Recently, several resource efficient 2D CNN architectures have been proposed. However, there is a lack of architectures for 3D counterparts. This work aims to fill this research gap.

The proposed architectures are generated by implementing the 3D versions of Squeezenet, MobileNet, MobileNetV2, ShuffleNet, ShuffleNetV2 architectures for 4 different complexity levels. The performance of these architectures have been evaluated using 3 different benchmarks, which are selected according to analyze models' capacities, how well the models capture the motion and the applicability of transfer learning for each model.

According to the analysis for 4 different complexity levels, the results show that these resource efficient 3D CNN architectures provide considerable classification performances. Using the $width\_multiplier$, the capacity of the architectures can be modified flexibly. The results on Jester benchmark show that depthwise convolutions are very good at capturing motion patterns. Moreover, nearly all models run in real-time both at Titan XP and Jetson TX2. As the results proved the applicability of transfer learning, these architectures can be used for other real-world applications by using pretrained models.

Acknowledgements

We give our special thanks to Stefan Hörmann for his assistance to this work. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU and Jetson TX2 Development Kit used for this research.

References

[1] `https://www.twentybn.com/datasets/jester/v1`.

[2] J. Carreira, E. Noland, A. Banki-Horvath, C. Hillier, and A. Zisserman. A short note about kinetics-600. *arXiv preprint arXiv:1808.01340*, 2018.

[3] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[4] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[5] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[6] C. Feichtenhofer, H. Fan, J. Malik, and K. He. Slowfast networks for video recognition. *arXiv preprint arXiv:1812.03982*, 2018.

[7] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[8] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[9] K. Hara, H. Kataoka, and Y. Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6546–6555, 2018.

[10] 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*, pages 770–778, 2016.

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

[12] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[13] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[14] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[15] J. Jin, A. Dundar, and E. Culurciello. Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474*, 2014.

[16] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[17] O. Köpüklü and G. Rigoll. Analysis on temporal dimension of inputs for 3d convolutional neural networks. In *Proceedings of the IEEE international conference on image processing, applications and systems*, 2018.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[20] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164*, 5, 2018.

[21] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[22] Z. Qiu, T. Yao, and T. Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *proceedings of the IEEE International Conference on Computer Vision*, pages 5533–5541, 2017.

[23] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnornet: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[25] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520. IEEE, 2018.

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[27] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[28] D. Soudry, I. Hubara, and R. Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pages 963–971, 2014.

[29] 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*, pages 1–9, 2015.

[30] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[31] D. Tran, J. Ray, Z. Shou, S.-F. Chang, and M. Paluri. Convnet architecture search for spatiotemporal feature learning. *arXiv preprint arXiv:1708.05038*, 2017.

[32] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

[33] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[34] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[35] 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*, pages 1492–1500, 2017.

[36] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[37] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6848–6856. IEEE, 2018.