Temporal Factorization of 3D Convolutional Kernels

Gabriëlle Ras, Luca Ambrogioni, Umut Güçlü, and Marcel A. J. van Gerven

Department of Artificial Intelligence, Radboud University,
Donders Centre for Cognition, Nijmegen, the Netherlands
{g.ras,l.ambrogioni,u.guclu,marcel.vangerven}@donders.ru.nl

Abstract. 3D convolutional neural networks are difficult to train because they are parameter-expensive and data-hungry. To solve these problems we propose a simple technique for learning 3D convolutional kernels efficiently requiring less training data. We achieve this by factorizing the 3D kernel along the temporal dimension, reducing the number of parameters and making training from data more efficient. Additionally we introduce a novel dataset called Video-MNIST to demonstrate the performance of our method. Our method significantly outperforms the conventional 3D convolution in the low data regime (1 to 5 videos per class). Finally, our model achieves competitive results in the high data regime (> 10 videos per class) using up to 45% fewer parameters.

Keywords: 3D convolution · Convolutional neural network · Factorization

1 Introduction

Modern deep learning has celebrated tremendous success in the area of automatic feature extraction from data with a grid-like structure, such as images. This success can be largely attributed to the convolutional neural network architecture [9], specifically 2D convolutional neural networks (CNNs). These networks are successful due the principles of sparse connectivity, parameter sharing and invariance to translation in the input space [2]. Loosely said, 2D CNNs efficiently find class-discriminating local features independent of where they appear in the input space. Since video is essentially a sequence of images/frames, 2D CNNs can be and are used to extract features from the individual frames of the sequence [6]. However, the drawback of this method is that the temporal information between frames is discarded. Temporal information is important when we want to perform tasks on video such as gesture, action and emotion recognition or classification. One possible way to simulate the use of time is to stack a recurrent layer after the convolutional layers [15]. But correlated spatiotemporal features will not be learnt because spatial and temporal features are explicitly learned in separate regions of the network. To solve this problem [1] proposed to expand the 2D convolution into a 3D convolution, essentially treating time as a third dimension. Ref. [5] used these 3D convolutions to build a 3D CNN for action recognition without using any recurrent layers. It is important to notice that the principles that govern 2D CNNs also govern 3D CNNs. Translation invariance in time is useful because the precise beginning and ending of an action are typically ill-defined [14]. Even though 3D CNNs have been shown to work for different kinds of tasks on video data, they remain difficult to train. There are roughly three main issues with 3D CNNs. First, they are parameter-expensive, requiring an abundance of GPU memory. Second, they are data-hungry, requiring much more training data compared to their 2D counterparts. And third, the increase in free parameters leads to a larger search space. As a result these models can be unstable and take a longer time to train. Existing literature tries to solve these problems by essentially avoiding the use of 3D convolutions completely. The most common method is the factorization of the 3D convolution into a 2D convolution followed by a 1D convolution at the layer level [11,13] or at the network level [7,8,12].

1.1 Contribution

We propose a simple and novel method to structure the way 3D kernels are learned during training. This method is based on the idea that nearby frames change very little in appearance. Each 3D convolutional kernel is represented as one 2D kernel with a set of transformation parameters. The 3D kernel is then constructed by sequentially applying a spatial transformation [4] *directly inside the kernel*, allowing spatial manipulation of the 2D kernel values. We achieve the following benefits:

- A reduction in the size of the search space by imposing a sequential prior on the kernel values;
- a reduction in the number of parameters in the 3D convolutional kernel;
- efficient learning from fewer videos.

2 Related Work

Previously, an entire 3D convolutional neural network was factorized into separate spatial and temporal layers called factorized spatio-temporal convolutional networks [12]. This was achieved by decomposing a stack of 3D convolutional layers into a stack of spatial 2D convolutional layers followed by a temporal 1D convolutional layer. Ref. [13] followed in this line of research by factorizing the individual 3D convolutional filters into separate spatial and temporal components called R(2+1)D blocks. Both methods managed to separate the temporal component from the spatial one. One on the network level [12] and one on the layer level [13]. To our knowledge our approach provides the first instance of a temporal factorization at the single kernel level. In effect, we applied the concept of the spatial transformer network [4] to the 3D convolutional kernel to obtain a factorization along the temporal dimension.

3 Methods

The proposed method uses fewer parameters compared to regular 3D convolutions and it imposes a strong sequential dependency on the relationship between temporal kernel slices. In theory our method should allow efficient feature extraction from video data, using fewer parameters and fewer data. The method is explained in Section 3.1. We demonstrate the performance of our method on a variant of the classic MNIST dataset [10] which we call Video-MNIST. The details of this dataset are explained in Section 3.2. As models we implement 3D and 3DTTN variants of LeNet-5: LeNet-5-3D and LeNet-5-3DTTN respectively (see Section 3.3). Training and inference details are explained in Section 3.4.

3.1 Temporal factorization of the 3D convolutional kernel

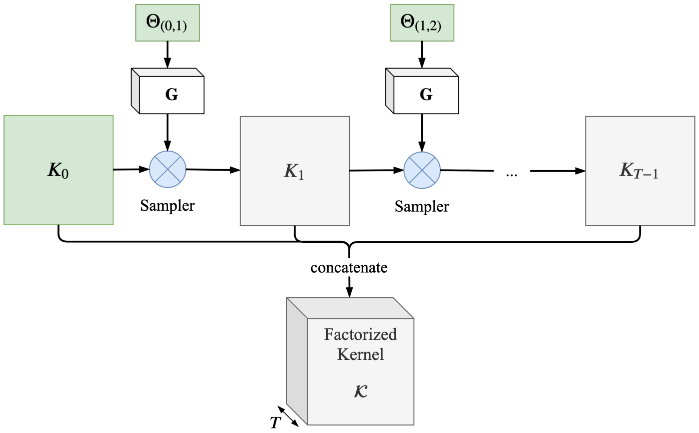

Fig. 1. The architecture of the temporally factorized 3D kernel. Each temporal kernel slice is sampled using the previous temporal slice and a transformation matrix. The resulting kernel slices are concatenated to form on the 3D kernel. Instead of learning the entire kernel we only learn K_0 and the set of transformations $\mathbf{\Theta}$.

Consider a 3D convolutional layer consisting of N 3D kernels. We focus on the inner workings of a single kernel $\mathcal{K} \in R^{T \times W \times H}$, where T, W and H refer to the temporal resolution, width and height of the kernel respectively. Without loss of generality, we will assume that the input has a channel with a dimension of one.

If we slice $\mathcal{K}$ along the temporal dimension, we end up with T 2D kernels $K \in R^{W \times H}$. Let us refer to the temporal slice at $t = 0$ as K_0. Instead of learning entire $\mathcal{K}$ directly, we only learn K_0 and $\mathbf{\Theta} \in R^{(T-1) \times 2 \times 3}$. We factorize $\mathcal{K}$ such that K_t with $t > 0$ depends indirectly on K_0 via $K_{t+1} = f(K_t; \Theta_{(t,t+1)})$ where $\Theta_{(t,t+1)} \in R^{3 \times 2}$ with $1 \leq t \leq T - 1$ are the learnable parameters of the transformation function f. For every pair of slices (K_t, K_{t+1}) we have

$$\Theta_{(t,t+1)} = \begin{bmatrix} \theta_{11} \; \theta_{12} \; \theta_{13} \\ \theta_{21} \; \theta_{22} \; \theta_{23} \end{bmatrix} \tag{1}$$

$\mathbf{\Theta}$ can be further restricted to only contain affine transformation parameters. That is, scaling s, rotation r, translation in the horizontal direction t_x and translation in the vertical direction t_y. This yields:

$$\Theta_{(t,t+1)} = \begin{bmatrix} s\cos r, \; -s\sin r, \; t_x s\cos r - t_y s\sin r \\ s\sin r, \;\; s\cos r, \;\; t_x s\sin r + t_y s\cos r \end{bmatrix} \tag{2}$$

In that case, $\mathcal{K}$ has only $W \cdot H + 4(T-1)$ free parameters. We can additionally add the restriction that there is only one shared transformation per kernel. That is, $\Theta_{(t,t+1)} = \Theta_{(t+1,t+2)} = ... = \Theta_{(T-1,T)}$ for $0 \leq t \leq T-1$. This results in just $W \cdot H + 4$ parameters. Essentially, given Θ, f modifies K_t to become K_{t+1}, sequentially building the 3D kernel from K_0. This way we impose a strong sequential relationship between the slices along the temporal dimension of our kernel.

The nonlinear transformation $f(K;\Theta)$ is applied in two stages. First, Θ is transformed into a sampling grid $\mathbf{G}$ that matches the shape of the input feature map, plus an explicit dimension for each spatial dimension, $\mathbf{G} \in R^{W \times H \times 2}$. Here K_t is the input feature map and K_{t+1} is the output feature map. We should think of this $\Theta \mapsto \mathbf{G}$ transformation as an explicit spatial mapping of Θ into the input feature space. Each coordinate (x, y) from the input space is split in separate G_x with $x \in [1, \ldots, W]$ and G_y with $y \in [1, \ldots, H]$ components, and calculated as

$$\begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \theta_{11} \; \theta_{12} \; \theta_{13} \\ \theta_{21} \; \theta_{22} \; \theta_{23} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3}$$

Now that we have sampling grid $\mathbf{G}$ we can obtain a spatially transformed output feature map K_{t+1} from our input feature map K_t. To interpolate the values of our new temporal kernel slice we use bilinear interpolation. For one particular pixel coordinate (x, y) in the output map we compute

$$K_{t+1,x,y} = \sum_{i=1}^{h}\sum_{j=1}^{w} K_{t,i,j} \max(0, 1 - |G_x - i|) \max(0, 1 - |G_y - j|)\,. \tag{4}$$

Given that our method transforms temporal kernel slices, we refer to 3D kernels composed with our method as 3DTT kernels. Convolutional networks that use 3DTT kernels instead of regular 3D kernels are referred to as 3DTT convolutional networks or 3DTTNs.

3.2 Video-MNIST

In order to test our method we constructed a dataset, referred to as Video-MNIST, in which each class has a different appearance and dynamic behavior.

Video-MNIST is a novel variant of the popular MNIST dataset. It contains 70000 sequences, each sequence containing 30 frames showing an affine transformation on a single original digit moving in a 28 × 28 pixel frame. The class-specific affine transformations are restricted to scale, rotation and x, y translations; see Table 1. We maintain the same train-validation-test split as in the original MNIST dataset. To make the problem more difficult and reliant on both spatial and motion cues, classes 0, 1, 5 and 7 and 9 contain random variations of their specific transformation respectively. For classes 0, 1, and 7 the initial direction (left or right) and the initial velocity at which the digit travels per frame is varied. In class 5 the direction of rotation and the size of the radius of the circular path are varied. In addition we also allow the digits to go partially out of frame or almost vanish (2 and 6). We also made sure that there are overlapping movements between classes, such as rotation or translation in the same direction (3, 4 and 8). Finally some classes can appear visually similar because of the transformation (6 and 9). In Figure 2 one example of each class is illustrated.

Table 1. Class-specific affine transformation applied on the original MNIST digits.

digit	**transformation description**	**parameter(s)**
0	moves horizontally	t_x
1	moves vertically	t_y
2	scales down and then up	s
3	rotates clockwise	r
4	rotates counter-clockwise	r
5	moves along a circular path	t_x, t_y
6	scales up while rotating clockwise	s, r
7	moves horizontally while rotating counter-clockwise	t_x, r
8	rotates clockwise and then counter-clockwise	r
9	random rotation and horizontal+vertical movements	r, t_x, t_y

3.3 Model architectures

We use the LeNet-5 architecture [10] as it is a good starting point for training a model based on a variant of the MNIST dataset. The original 2D convolutions are replaced with regular 3D convolutions and 3DTT convolutions for the LeNet-5-3D and the LeNet-5-3DTTN respectively. The number of filters in each convolutional layer can vary since during experimentation we noticed that we can achieve better performance by either increasing or reducing the number of filters in the convolutional layers for both LeNet-5-3D and LeNet-5-3DTTN. LeNet-5-3D serves as the baseline model.

3.4 Training and inference

Training All models are optimized using SGD with mometum of 0.9. Depending on the model, the starting learning rate value can vary from $1e-8$ to $5e-9$. The

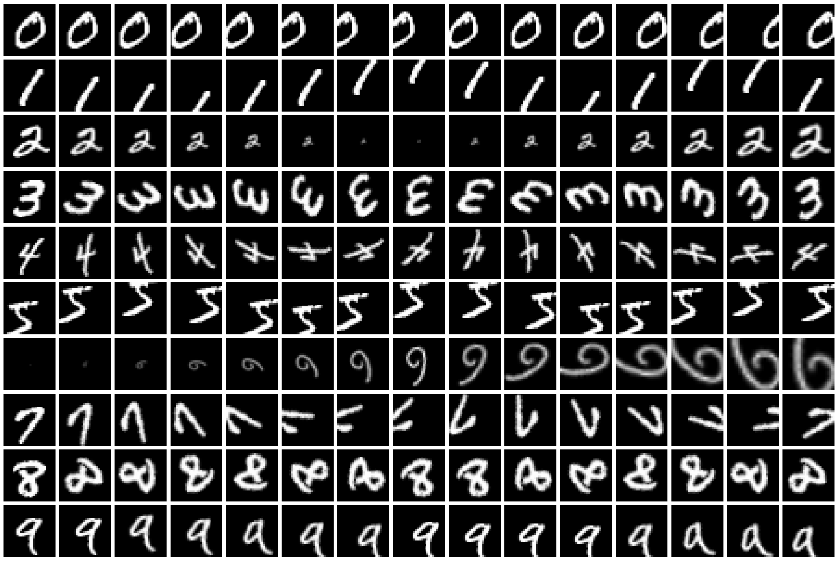

Fig. 2. One example from each Video-MNIST class. Instead of displaying the full 30 frames sequence we display 15 frames, skipping one frame each time, such that the figure fits within the margins of the page.

models are trained for a total of 100 epochs where every 10th epoch the learning rate decreases exponentially if the validation accuracy has not improved. We noticed that 100 epochs provides a good time-window for the models to converge. Generally a batch size of 20 is used unless we are training on only 10 videos, in which case a batch size of 10 is used. Initialized LeNet-5-3D model weights as well as all fully connected layers follow a Kaiming-Uniform scheme [3][1]. LeNet-5-3DTTN initializes K_0 with weights sampled from a Gaussian distribution[2]. In our main experiments we use a parameterization of Θ with the following initialization: $s = 1$, $r = 0$, $t_x = 0$ and $t_y = 0$.

Replication of video selection Each model is ran 30 times (runs) with the same initialization parameters but with different randomly initialized weights for the convolution and fully connected layers. The training data is randomly selected across different runs by using a seed. The seed assures that the same videos are chosen again when we execute the same run with a different model or when we use different initialization parameters. This way we can compare only the difference between the model architecture and parameters without confounding our results with video variance. Given that we experiment with very

[1] The default setting in PyTorch.

[2] Experimentally this gave the best results, however there was very little difference between different types of initializations.

few videos, we make sure that the classes are represented equally in the randomly selected training data.

Inference Model selection is based on the accuracy of the validation split. The 30 models in the run with highest average accuracy are ran against the test split. Each run is essentially the same model using the same hyperparameters but with different randomized weight initializations for the convolutional and fully connected layers. In the end the test results of the 30 models are averaged and the standard error of the results is calculated. The final results can be seen in Figure 3.

3.5 Setup

To test if our method can outperform conventional 3D convolutions on very few datapoints we train each separate model on a different number of training videos. This way we can test how efficient our method is. The total number of videos are varied from low to high: $10, 20, 30, 40, 50, 100, 500, 1000, 2000, 5000$. Model selection happens for each of the number of videos separately. The models trained on 10 videos are different from the models trained on 20 videos. After model selection based on the validation split the models from the best run perform inference on the test split.

4 Results

In Table 3 and Figure 3 we can see that the that our method outperforms the conventional 3D convolution significantly in the low data regime. However, when we have ample training data the conventional 3D convolution outperforms our method, as is to be expected. It is worth mentioning that, in general, our method uses fewer parameters and still achieves reasonable results in all settings.

Table 2. Detailed results LeNet-5-3D vs. LeNet-5-3DTTN on Video-MNIST.

	LeNet-5-3D			**LeNet-5-3DTTN**		
train videos	mean accuracy	standard error	model parameters	mean accuracy	standard error	model parameters
10	0.374	0.0122	375668	**0.477**	0.0139	444966
20	0.498	0.0131	496430	**0.588**	0.0108	348612
30	0.625	0.0107	496430	**0.668**	0.0091	444966
40	0.671	0.0095	496430	**0.721**	0.0079	348612
50	0.750	0.0083	496430	**0.773**	0.0067	444966
100	**0.837**	0.0059	496430	0.829	0.0073	348612
500	**0.960**	0.0025	496430	0.925	0.0058	222940
1000	**0.976**	0.0014	496430	0.968	0.0020	254058
2000	**0.988**	0.0007	496430	0.972	0.0022	222940
5000	**0.994**	0.0003	496430	0.988	0.0010	222940

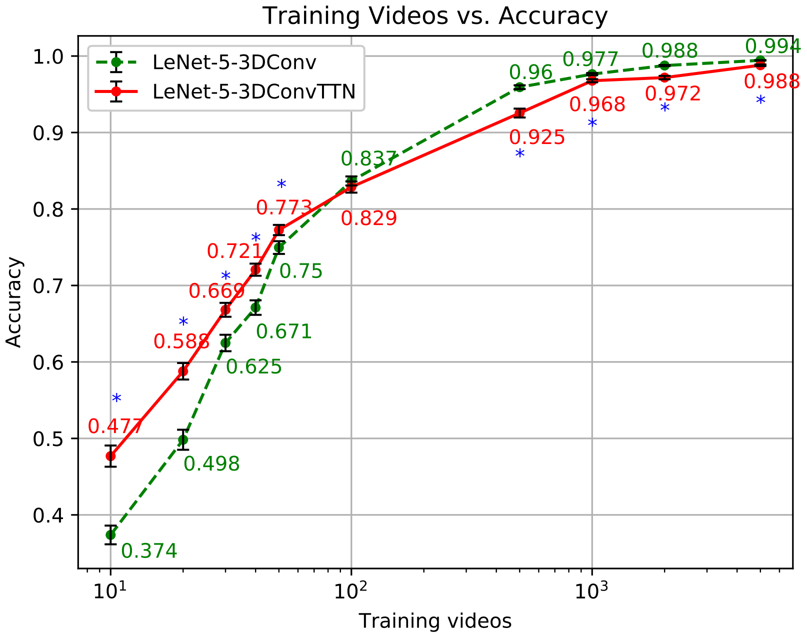

Fig. 3. A comparison of the model using regular 3D convolutions with the model using our factorized 3D convolutions on the entire test split of Video-MNIST. An asterisk denotes if the difference between models is significant.

5 Conclusion

We propose a novel factorization method for 3D convolutional kernels. Our method factorizes the 3D kernel along the temporal dimension and provides a way to learn the 3D kernel through transformations of a 2D kernel, thereby greatly reducing the number of parameters needed. We demonstrate that our method significantly outperforms the conventional 3D convolution in the low data regime (10 to 50 training videos), yielding 0.58 vs. 0.65 on average for the LeNet-5-3D and the LeNet-5-3DTTN respectively. Additionally our model achieves competitive results in the high data regime ($>$ 100), with 0.95 vs. 0.94 on average for the LeNet-5-3D and the LeNet-5-3DTTN respectively, using up to 45% fewer parameters. Hence, 3DTTNs provide a useful building block when estimating models for video processing in the low data regime. In future work, we will explore in which real-world problem settings 3DTTNs outperform their nonfactorized counterparts.

References

1. Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., Baskurt, A.: Sequential deep learning for human action recognition. In: International workshop on human behavior understanding. pp. 29–39. Springer (2011)
2. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT Press (2016)
3. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
4. Jaderberg, M., Simonyan, K., Zisserman, A., et al.: Spatial transformer networks. In: Advances in neural information processing systems. pp. 2017–2025 (2015)
5. Ji, S., Xu, W., Yang, M., Yu, K.: 3d convolutional neural networks for human action recognition. IEEE transactions on pattern analysis and machine intelligence **35**(1), 221–231 (2012)
6. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 1725–1732 (2014)
7. Lea, C., Reiter, A., Vidal, R., Hager, G.D.: Segmental spatiotemporal cnns for fine-grained action segmentation. In: European Conference on Computer Vision. pp. 36–52. Springer (2016)
8. Lea, C., Vidal, R., Reiter, A., Hager, G.D.: Temporal convolutional networks: A unified approach to action segmentation. In: European Conference on Computer Vision. pp. 47–54. Springer (2016)
9. Lecun, Y.: Generalization and network design strategies. In: Connectionism in perspective. Elsevier (1989)
10. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
11. Qiu, Z., Yao, T., Mei, T.: Learning spatio-temporal representation with pseudo-3d residual networks. In: proceedings of the IEEE International Conference on Computer Vision. pp. 5533–5541 (2017)
12. Sun, L., Jia, K., Yeung, D.Y., Shi, B.E.: Human action recognition using factorized spatio-temporal convolutional networks. In: Proceedings of the IEEE international conference on computer vision. pp. 4597–4605 (2015)
13. Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., Paluri, M.: A closer look at spatiotemporal convolutions for action recognition. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 6450–6459 (2018)
14. Varol, G., Laptev, I., Schmid, C.: Long-term temporal convolutions for action recognition. IEEE transactions on pattern analysis and machine intelligence **40**(6), 1510–1517 (2017)
15. Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., Toderici, G.: Beyond short snippets: Deep networks for video classification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4694–4702 (2015)