THE 3TCONV: AN INTRINSIC APPROACH TO EXPLAINABLE 3D CNNS

Anonymous authors
Paper under double-blind review

ABSTRACT

Current deep learning architectures that make use of the 3D convolution (3DConv) achieve state-of-the-art results on action recognition benchmarks. However, the 3DConv does not easily lend itself to explainable model decisions. To this end we introduce a novel and intrinsic approach, whereby all the aspects of the 3DConv are rendered explainable. Our approach proposes the temporally factorized 3D convolution (3TConv) as an interpretable alternative to the regular 3DConv. In a 3TConv the 3D convolutional filter is obtained by learning a 2D filter and a set of temporal transformation parameters, resulting in a sparse filter requiring less parameters. We demonstrate that 3TConv learns temporal transformations that afford a direct interpretation by analyzing the transformation parameter statistics on a model level. Our experiments show that in the low-data regime the 3TConv outperforms 3DConv and R(2+1)D while containing up to 77% less parameters.

1 INTRODUCTION

Understanding the inner workings of ConvNets is important when they are used to make actionable decisions or when humans have to make actionable decisions based on lower-level decisions from decision support systems. Three-dimensional convolutional networks (3DConvNets) (Baccouche et al., 2011) are a natural extension of 2DConvNets and have been investigated for the processing of spatio-temporal data, e.g., action recognition in video (Ji et al., 2012; Tran et al., 2015; Varol et al., 2017; Carreira & Zisserman, 2017). While some of these models achieve state-of-the-art results in video action recognition benchmarks, the temporal aspect of their inner workings remains difficult to interpret. Given that a 3DConv filter is simply the 3D extension of the 2DConv filter, one could apply visualization methods that are suitable for 2DConvNets to the individual slices along the temporal axis of a 3DConv filter (Carreira & Zisserman, 2017; Anders et al., 2019; Yang et al., 2018). This approach is effective to the degree that we can gain insight into what the model learns in terms of spatial features. However, these methods do not provide a meaningful insight into the temporal dynamics that the model takes into consideration.

Activation-maximization (Erhan et al., 2009; Simonyan et al., 2013; Olah et al., 2017) visualizes important features in an arbitrary layer of a DNN by optimizing a randomly initialized input such that the activation of the chosen neuron in a layer is maximized. One big obstacle to this approach is that the resulting visualizations can be difficult to recognize and are subject to interpretation. Applying such methods on 3DConvs will likely exacerbate these problems given that the search space for optimization is at least one order of magnitude larger (depending on the number of frames in the video). Saliency extraction methods reveal which input region causes high activations in specific network components (Simonyan et al., 2013; Montavon et al., 2017; Zhou et al., 2016; Selvaraju et al., 2017). Applying this method to a 3DConv results in a saliency sequence that can be applied to the input video to reveal spatial components that cause a channel to have a high output. Concrete examples can be seen in Anders et al. (2019), where saliency extraction analysis (Simonyan et al., 2013) and deep Taylor decomposition and layerwise relevance propagation (Montavon et al., 2017) are used on a 3DConvNet to analyze model predictions. Yang et al. (Yang et al., 2018) explain model predictions by adapting CAM (Zhou et al., 2016) and Grad-CAM (Selvaraju et al., 2017) for their 3DConvNet. However, the results of the CAM methods completely depend on resolution of the activation maps in the last convolutional layer. This is in turn dependent on the specific network architecture. While saliency extraction methods return less subjective spatial representations in the

input space, compared to activation-maximization, their application to the temporal domain leaves much to be desired.

To bridge this gap, we propose a novel approach through which the model learns temporal parameters that afford direct interpretation. Temporal parameterization provides a novel way of visualizing and understanding the temporal dynamics learned by a 3TConvNet. These temporal parameters are multi-functional:
• **Intrinsic interpretability**: The 3DConv filter is factorized into a 2D spatial filter and corresponding temporal parameters that transform the 2D filter into a 3D filter. The 3TConvNet learns explicit temporal affine transformations whose values can be plotted in an interpretable graph.
• **Complement existing 2D methods**: Increase interpretation ability when used in combination with existing 2D methods.
• **Increased performance**: When learning from smaller datasets, 3TConvNet outperforms alternatives such as 3DConvNet and R(2+1)D.
• **Fewer parameters**: The approach reduces the number of parameters in the model up to 77%.
• **Transfer learning**: Additionally, the parameterization provides an intuitive way to reuse weights from pretrained 2DConvNets.

2 Related work

Factorization. Previously, Tran et al. (2018) have factorized the individual 3D convolutional filters into separate spatial and temporal components called R(2+1)D blocks. This creates two specific learning phases: a spatial feature learning phase and a temporal feature learning phase guided by one weight per temporal dimension. Sun et al. (2015) proposes factorized spatio-temporal ConvNets which factorizes a 3DConvNet at the layer level, resulting in a network where the lower layers are 2DConv layers and the higher layers are 1DConv layers. Similarly, Qiu et al. (2017) develop the Pseudo-3D Residual Net approach which factorizes 3DConvs by first constructing a 2DConv component and then a 1DConv component in a bottleneck fashion. What these approaches have in common is that the input is first processed by 2DConvs followed by 1DConvs. Our method is distinctive in the sense that there are no two separate stages of processing the input. The 3DConv operation remains intact. The novelty we propose lies in the procedure used to obtain the weights for the kernel. In other words, the factorization is performed on the 3D kernel values themselves rather than on the operation. From this point of view, R(2+1)D, and the other decomposition methods, are quite different compared to our method.
Symmetry transformations. In our work we wish to exploit symmetries that occur through time by learning the optimal affine transformations to generate 3D kernels. Several previous works have introduced a conceptually similar idea, however, applied only to the spatial dimensions. Jaderberg et al. (2015) apply affine transformations to features extracted from 2D filters. Our method differs in the sense that it is used to obtain sequentially build a 3D filter, while in Jaderberg et al. (2015) the affine transformations are used to obtain spatially robust feature representations in 2DConvNets. Group equivariant convolutions (Cohen & Welling, 2016) make use of symmetry groups containing reflection and rotation to extend translational spatial symmetry. The goal is to learn less redundant convolutional filters in the spatial domain. Unlike these methods, our 3TConv uses affine transformations to model symmetries in the kernel values themselves, rather than in the input.

3 Methods

One of the main sources of temporal variability in video streams is the optical flow due to the motion of the camera and of the background. In addition, important objects in the foreground can also exhibit temporal motion patterns that can be captured by affine transformations. These movements induce a constant optical flow that can be modeled as a global affine transformation of the frames. This suggests an affine parameterization of the transformation function in terms of translations, rotations and scaling parameters.

To clearly demonstrate the procedure we will build our way up to a consistent notation starting from the 2DConv.

2DConv[1]
The input is an image $\boldsymbol{X} \in \mathbb{R}^{h \times w}$, where h and w denote the height and width of the input. Convolutional kernel $\boldsymbol{K} \in \mathbb{R}^{n \times m} : n \leq h, m \leq w$, where n and m denote the height and width dimensions of the kernel. Output $\boldsymbol{Y} \in \mathbb{R}^{h' \times w'}$, where $h' = \frac{h-n+2p}{s} + 1$, $w' = \frac{w-m+2p}{s} + 1$. s and p denote the stride and padding. In the rest of the examples we fix $s = 1$ and $p = 0$, hence, $h' = h - n + 1$ and $w' = w - m + 1$. The result of $\boldsymbol{Y} = \boldsymbol{X} * \boldsymbol{K}$, $*$ indicating the convolution operation, where for each element $Y_{\alpha,\beta} \in \boldsymbol{Y}$, the calculations can be written as:

$$Y_{\alpha,\beta} = \sum_{i=1}^{n} \sum_{j=1}^{m} K_{i,j} X_{i+\alpha-1, j+\beta-1} \tag{1}$$

3DConv
We now introduce the temporal dimension into our calculations. The input is an image sequence $\mathbf{X} \in \mathbb{R}^{h \times w \times d}$, where d denotes the time. d corresponds with the number of frames in the image sequence. Convolutional kernel $\mathbf{K} \in \mathbb{R}^{n \times m \times l} : l \leq d$, where l denotes the temporal dimension of the kernel. Output $\mathbf{Y} \in \mathbb{R}^{h' \times w' \times d'}$, where $d' = d - l + 1$. Similar to the 2D case, we can write the per-element result $\mathbf{Y} = \mathbf{X} * \mathbf{K}$ as:

$$Y_{\alpha,\beta,\gamma} = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{b=1}^{l} K_{i,j,b} X_{i+\alpha-1, j+\beta-1, b+\gamma-1} \tag{2}$$

3TConv
For the 3TConv we introduce a transformation function $f(\mathbf{K}_{:,:,1}, \mathbf{\Theta})$ where kernel $\mathbf{K}_{:,:,1} \in \mathbb{R}^{n \times m \times 1}$ and $\mathbf{\Theta} \in \mathbb{R}^{n \times m \times (l-1)}$. Each slice $\mathbf{\Theta}_{:,:,b}$ corresponds to an affine transformation matrix that is derived from the affine transformation parameters scale s, rotation r and translation t_x, t_y[2]. Using function f we obtain sparse kernel $\mathbf{K}$ where each consecutive slice is a matrix multiplication of the previous slice with the corresponding transformation matrix:

$$\begin{aligned}
\mathbf{K}_{:,:,2} &= f(\mathbf{K}_{:,:,1}, \mathbf{\Theta}_{:,:,1}) = \mathbf{K}_{:,:,1} \mathbf{\Theta}_{:,:,1} \\
\mathbf{K}_{:,:,3} &= f(\mathbf{K}_{:,:,2}, \mathbf{\Theta}_{:,:,2}) = \mathbf{K}_{:,:,1} \mathbf{\Theta}_{:,:,1} \mathbf{\Theta}_{:,:,2} \\
&\vdots \\
\mathbf{K}_{:,:,l} &= f(\mathbf{K}_{:,:,l-1}, \mathbf{\Theta}_{:,:,l-1}) = \mathbf{K}_{:,:,1} \mathbf{\Theta}_{:,:,1} \mathbf{\Theta}_{:,:,2} ... \mathbf{\Theta}_{:,:,l-1}
\end{aligned} \tag{3}$$

This results in $\mathbf{K} = \{\mathbf{K}_{:,:,1}, \mathbf{K}_{:,:,2}, ..., \mathbf{K}_{:,:,l}\}$. Notice that the depth of $\mathbf{\Theta}$ is $l - 1$ and not l. In order to get l number of slices in the final kernel given that we start with $\mathbf{K}_{:,:,1}$, we need $l - 1$ number of transformations to derive until $\mathbf{K}_{:,:,l}$.

We can summarize the 3TConv as:

$$f(\mathbf{K}_{:,:,1}, \mathbf{\Theta}) = \left(\prod_{b=1}^{l-1} \mathbf{\Theta}_{:,:,b} \right) \mathbf{K}_{:,:,1} \tag{4}$$

Substituting the 3TConv into Equation 2, the full 3TConv calculation is written as:

$$Y_{\alpha,\beta,\gamma} = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{b=1}^{l} \left(\prod_{z=1}^{b} \Theta_{i,j,z} \right) K_{i,j,1} X_{i+\alpha-1, j+\beta-1, b+\gamma-1} \tag{5}$$

3.1 Optimization procedure

When a 3TConvNet is first initialized, the temporal parameters are initialized as the identity mapping: $s = 1$, $r = 0$, $x = 0$ and $y = 0$. The initial kernel slice $\mathbf{K}_{:,:,1}$ is implemented as a 2D matrix,

[1]For the sake of clarity in this definition and those that follow, we will assume that the channels in the input are equal to one.

[2]Due to space limitations the derivation is given in the Appendix A

initialized according to a normal distribution. Thus in the initial kernel all slices are equal: $\mathbf{K}_{:,:,1} = \mathbf{K}_{:,:,2} = ... = \mathbf{K}_{:,:,l}$. In each 3TConv layer, a layer containing k number of kernels, each kernel is comprised of the 2D slice $\mathbf{K}_{:,:,1}$ and a unique set of temporal parameters $\{s, r, t_x, t_y\}$. During training the set of temporal parameters is updated for each kernel along with the the 2D matrix implementation of kernel slice $\mathbf{K}_{:,:,1}$. Compared to the 3DConv, the 3TConv requires a much lower learning rate due to the sensitivity of the temporal parameters. As a result the training procedure is three to four times slower than the 3DConv. Theoretically it should be possible to achieve a more efficient implementation of the 3TConv and we reserve the investigation of this matter for future work.

3.2 Explaining temporal dynamics with 3TConv

As mentioned before, the affine transformation matrix slices $\mathbf{\Theta}_{:,:,b}$ is derived from the set of parameters $\{s, r, t_x, t_y\}$. These parameters can be extracted as-is from a trained 3TConvNet, see Appendix A. The s parameter is the scaling factor relative to 1. The larger s, the bigger the resulting transformation. Applied on an image this has the effect of zooming in or out. The r parameter is the rotation measured in degrees where a positive value of r indicates a counter-clockwise rotation. In the resulting plots we multiply $-1 \times r$ such that a positive value indicate a clockwise rotation. The translation parameters in their raw form indicate what percentage of the image has translated. To obtain the amount of translation in pixel units we need to multiply by the width W and height H of the image: $px_x = t_x W$ and $px_y = t_y H$.

4 Experiments

In the following experiments we will show several examples demonstrating how 3TConvs can be used for explainability. We urge the reader to consider that the uses of 3TConvs are not limited to these examples and that our novel parameterization approach affords concrete and objective values previously unavailable.

Modified versions of ResNet18 (He et al., 2016) and GoogLeNet (Szegedy et al., 2015) are used. In both architectures the 2DConv filters are replaced by 3DConv and 3TConv filters. The networks are trained on the Jester dataset (Materzynska et al., 2019) to classify 27 different hand-gestures and the UCF101 dataset (Soomro et al., 2012) to classify 101 human activities in various scenarios. Implementation was done in PyTorch (Paszke et al., 2019) and for the experiments using transfer learning, model weights are obtained from pretrained GoogLeNet and ResNet18 models from the torchvision model zoo. Details about data pre-processing and model training can be found in Appendix C.

4.1 Baseline: Applying 2D visualization methods to 3D- and 3TConv

Next, we applied the saliency extraction method from Simonyan et al. (2013) and activation-maximization method from Olah et al. (2017). The goal is to demonstrate how 2DConv visualization methods translate to the 3DConv and 3TConv domain as a baseline for comparing the interpretation when temporal parameters are available. We also made a 3D implementation of Grad-CAM (Selvaraju et al., 2017), however, we soon discovered that this method was not suitable to our architectures due to insufficient temporal resolution in the resulting activation maps.

Figure 1 depicts the visualizations. In the saliency extraction results we can see high-activation responses for specific channels. The results indicate that sometimes the models are picking up spatial components that are both recognizable to humans and that are relevant to classification, such as areas of the hands. However, for both 3D and 3T models there are also instances where it is not clear why the highlighted spatial components are relevant to the target class, i.e., when the model is looking at the background.

As mentioned in Section 2, the application of activation-maximization can lead to visual patterns that are difficult to interpret. Figure 1 indeed shows that it is difficult to relate the patterns observed in the images to visual aspects of objects that are present in the frames. However, to some extent, the patterns in Figure 1B correspond better to the saliency region that is indicated by the saliency extraction results. For example, channel 121 is extracting features from the hand and the maximized

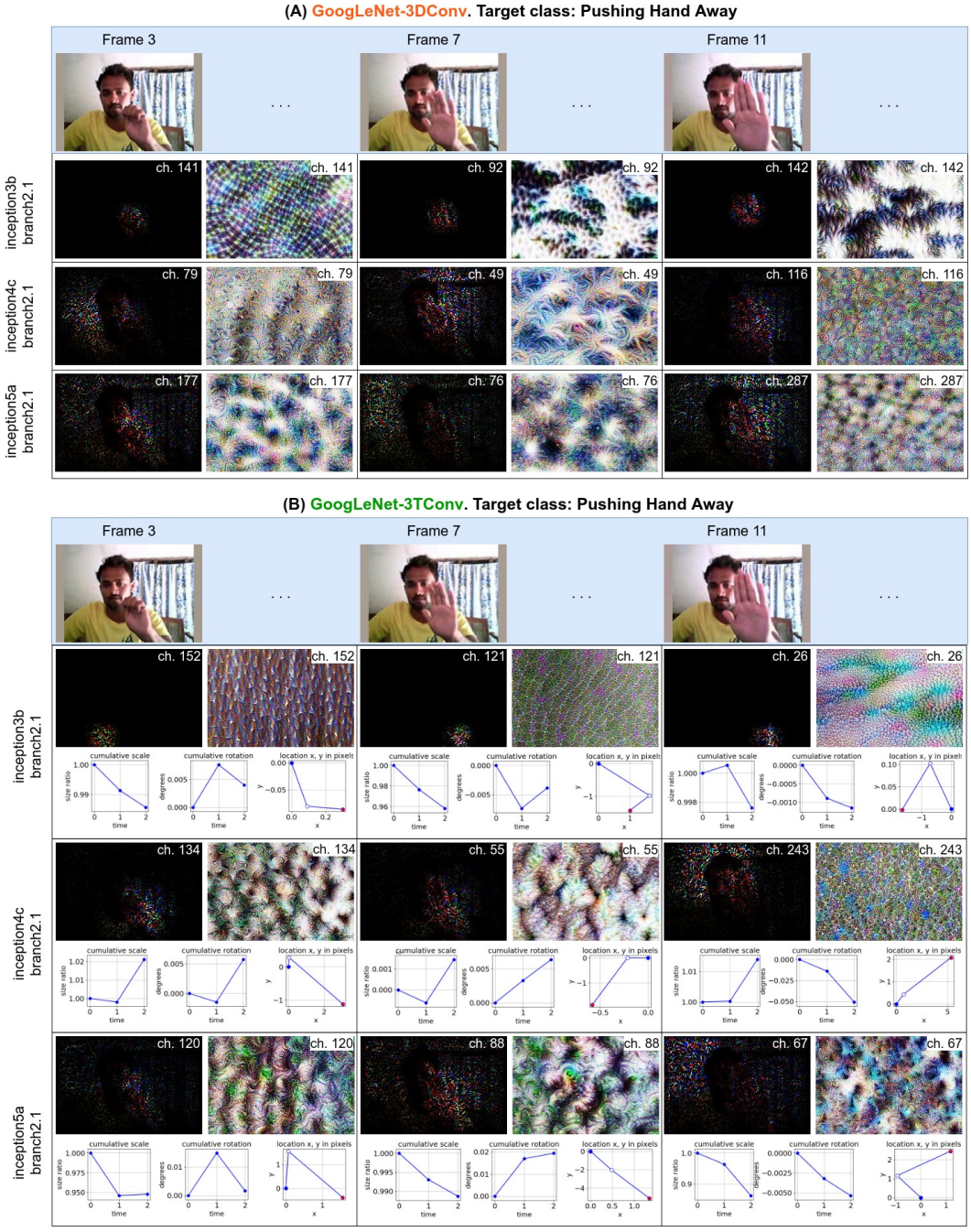

Figure 1: (A+B) 2D visualization methods, saliency extraction method (left) and activation-maximization (right), are applied to a 3DConv and a 3TConv. The 2D results are somewhat subjective to interpretation, especially the activation-maximization. For saliency extraction it is not always clear why the network is finding that particular part of the image salient. In (B) we extract the temporal parameters and use them to enrich the interpretation of the 2D results. For each channel a combined explanation is generated: the saliency extraction method (left), the activation-maximization (right) and the set of temporal parameters are plotted (bottom). In the right-most plot indicating the translation parameters, blue denotes the starting position and red denotes the ending position. For the chosen video where the target class is "Pushing Hand Away" we can see that the high-activation channels contain translation parameters that correspond with what is happening in the original video. We also notice a trend where temporal parameters in the higher layers have larger values and are easier to interpret. It is likely that if a 3TConv layer is used as the output layer, that the temporal parameters would directly correspond with the main actions in the target class.

patterns resembles skin cells when looked at from up close. While the interpretation of these patterns is subjective, in any case, no insight can be achieved about the temporal qualities that the model may extract from the data. Additional visualizations are linked in Appendix C.

4.2 COMBINING TEMPORAL TRANSFORMATIONS WITH 2D VISUALIZATIONS

Results are shown in Figure 1B. In each channel that we investigate, the temporal parameters are applicable to features extracted using that particular channel, similar to how a channel with a 2DConv extracts a specific pattern from the input. When we apply saliency extraction method, we can see which channel was most activated by features in the input.

Since we are operating in a 2D medium (paper) and we are trying to explain a 3D phenomenon, it is difficult to accurately describe the movements in the original video, the reader can view the input video here. The video has been slowed down, truncated to 15 frames and annotated with the frame numbers for the purposes of viewing.

Starting from the inception3b branch2.1, we can see that the model is focusing on relatively small regions, as indicated by the results of the saliency extraction method. Channel 121 is detecting a salient feature that overlaps with the person's wrist on frame 7. Taking the temporal parameters into account we see that this part of the hand specifically corresponds the action of moving slightly downwards. In the input video we can see that this is in line with what that part of the body is doing; as the hand gets closer to the camera, the palm of the hand tends towards a downwards motion. Looking at inception4c branch2.1: In channel 134, frame 3, the fist opens a little and moves a little bit downward towards the left. In this case the scaling and translation parameters correspond to the action in the video. In channel 55, frame 7, the hand is moving a little bit downward according to the translation parameter. This parameter also corresponds with the action in the video. The rotation parameter does not seem to be utilized much and the translation parameter is being utilized the most. We had expected that scaling would have larger values, especially for this class, where the hand if sequentially increasing in size. However, it can be the case that the translation parameters are compensating for the scaling, which also explains why they appear more developed.

Overall we get the picture that on low and mid layers, the temporal parameter values are comparatively smaller than those in the higher layer. This is expected since larger actions are made up of smaller actions. The parameter that gave the most consistent interpretation was the translation parameter, especially in channels 121, 134 and 88 we can see a clear downward-right moving trend that corresponds with the action taken in the actual video. Given these results we conclude that the higher layer parameters will be most interpretable and will correspond to the target class. If a 3TConv is used as the final layer in a ConvNet, the temporal parameters will correspond directly with the classes.

4.3 ANALYSIS OF TEMPORAL PARAMETERS ACROSS MODELS AND DATASETS

In the previous section, the temporal parameters were analyzed on a per-channel level. The temporal parameters can also be used to understand the global decision-making process of the model in interpretable and quantifiable model statistics. In Figure 2, the distributions of the learned temporal parameters of 3TConv-ResNet18 and 3TConv-GoogLeNet trained on the Jester and UCF101 datasets are visualized. This overview allows us to compare different models trained on different datasets. We immediately attain a high-level view of what each network considers important for classification for the specific datasets.

Even though the results of 3TConv suggest that temporal dynamics are more important for Jester than UCF101, it does not outperform the 'less interpretable' 3DConv on the Jester dataset. Noting that the Jester dataset differs significantly from UCF101 in terms of samples per class, fewer number of classes and less variety in dataset, this result is in line with our general empirical observations that 3TConv typically outperforms 3DConv in the low-data regime. This is likely due to 3DConv being a more complex and flexible model and better at exploiting the availability of more data.

4.4 PERFORMANCE COMPARISON

The performance comparison is shown in Table 1. On UCF101, pretrained 3TConvNets can outperform pretrained 3DConvNets using up to 77% fewer parameters. However on the Jester dataset

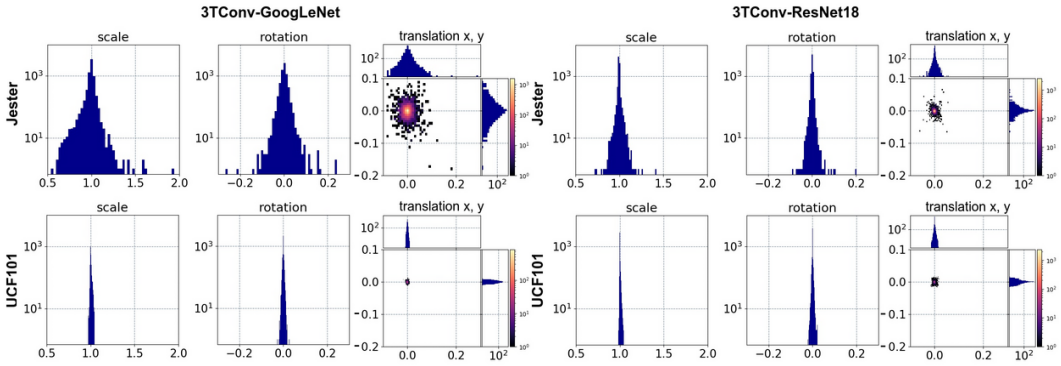

Figure 2: Distributions of the temporal parameters of pretrained 3TConv versions of GoogLeNet and ResNet18. Each network has been trained on the Jester dataset for gesture recognition and on the UCF101 dataset for action recognition. Each separate panel represents the distributions of all the learned temporal parameters across a single model after training on the indicated dataset. In each separate panel, three distributions are displayed: (i) the distribution for the scale parameter, (ii) the distribution for the rotation parameters, and (iii) the joint distribution for the translation parameters. Note that the axis scales are equal for each panel so that a direct comparison can be made, between models and well as between datasets.

3DConvNets outperform 3TConvNets. This discrepancy is explained by the fact that models with more parameters can learn from bigger datasets better than smaller or more restricted models. In preliminary experiments we have seen evidence that 3TConv consistently outperforms 3DConv in the low data regime, see Appendix B.

Table 1: Classification accuracy vs. number of model parameters

	Jester val acc %	# params	UCF101 val acc %	# params
scratch 3T-GoogLeNet	74.1	7419121	33.4	7646671
pret. 3T-GoogLeNet	84.9	7419121	**63.7**	7646671
pret. 3D-GoogLeNet	89.9	14078833	58.7	14306383
scratch. 3T-ResNet18	44.4	11222619	31.0	11260581
pret. 3T-ResNet18	74.5	11222619	61.1	11260581
pret. 3D-ResNet18	81.6	33217755	56.1	33255717
scratch R(2+1)D	**91.8**	31313976	31.4	31351938

Comparison with R(2+1)D. In order to compare our method with current state-of-the-art methods we compare it with R(2+1)D model trained from scratch on the Jester and UCF101. The results can be seen in Table 1. We can only compare the model trained from scratch due to time limitations involving the download of the Kinetics dataset for a full comparison and we reserve this for future work. From the comparison with the tabula rasa models, R(2+1)D performs exceptionally well on the Jester dataset, however, on the UCF101 dataset it is outperformed by our 3TConv using 75% fewer parameters. This is in line with our previous observations where 3TConv outperforms 3DConv in the low data regime.

4.5 Results for tabula rasa models

Table 1 also compares the classification accuracies for 3TConvNets that were either pretrained or trained from scratch. Pretrained models massively outperform tabula rasa models, showing that transfer learning has a significant impact on 3TConv performance. In the final row a comparison can be made with R(2+1)D method trained from scratch.

Figure 3 show the parameters estimated for tabula rasa models that were trained from scratch. A comparison between these results and those of the previous section shows that the difference in results

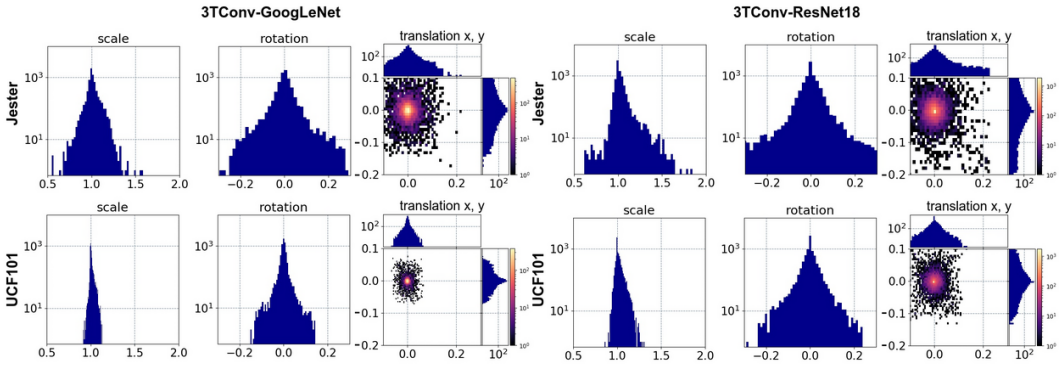

Figure 3: Distributions of the temporal parameters for the 3TConv versions of GoogLeNet and ResNet18 that were trained from scratch. Each network has been trained on the Jester dataset for gesture recognition and on the UCF101 dataset for action recognition. Each separate panel represents the distributions of all the learned temporal parameters across a single model after training on the indicated dataset. In each separate panel, three distributions are displayed: (i) the distribution for the scale parameter, (ii) the distribution for the rotation parameters, and (iii) the joint distribution for the translation parameters. Note that the axis scales are equal for each panel so that a direct comparison can be made, between models and well as between datasets.

for the Jester and UCF101 datasets becomes more pronounced for tabula rasa models. This is to be expected since models trained from scratch are not endowed with good visual features and likely develop a larger repertoire of temporal parameters to compensate for this. It seems that, on the Jester dataset, the model needs to develop a larger variety of temporal parameters to do classification. This suggests that classification on the Jester dataset is more dependent on affine motion transformations than classification on the UCF101 dataset. This is not surprising since many of the classes in UCF101 are not dependent on unique motion patterns at all. For example, the SkyDiving, Skiing, and Skijet classes can be classified based on spatial features alone. In contrast, the classes in the Jester dataset are strongly dependent on what kind of motion the person performs with their hand.

The comparison with pretrained models further reveals that the dependency of the models on the temporal parameters decreases strongly across models and datasets when initialization with pretrained weights is provided. This implies that when the model can extract good spatial features, the dependency on learning a broad range of temporal parameters decreases. This interesting phenomenon warrants further investigation.

5 Discussion

While using the affine parameterization might incur additional complexity during optimization, we have empirically observed in our initial experiments that affine parameterizations performs on par with free parameterizations, that is, when the transformations are not restricted to affine transformations. It is an open question to what extent parameterization limits the expressiveness of the 3TConv and which kind of parameterization can lead to more interpretable parameters. An idea that we have also not explored is whether temporal parameters can be extracted from trained 3DConvs and if insight about the model can be achieved by analyzing the extracted parameters. An in depth analysis of the interpretability of the temporal parameters at various layers and how the values change from lower to higher layers should most definitely be conducted given that in this paper we have observed trends that support the hypothesis that last layer parameters should correspond better to class actions. One approach for achieving this is to analyze the final layer featuremaps in a conceptually similar manner as Grad-CAM, however, a solution for the lack of temporal resolution needs to be found.

References

Christopher J Anders, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Understanding patch-based learning of video data by explaining predictions. In *Explainable AI: Interpreting,*

Explaining and Visualizing Deep Learning, pp. 297–309. Springer, 2019.

Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential deep learning for human action recognition. In *International workshop on human behavior understanding*, pp. 29–39. Springer, 2011.

Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6299–6308, 2017.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pp. 2990–2999, 2016.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pp. 2017–2025, 2015.

Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

Joanna Materzynska, Guillaume Berger, Ingo Bax, and Roland Memisevic. The jester dataset: A large-scale video dataset of human gestures. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Zhaofan Qiu, Ting Yao, and Tao Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *proceedings of the IEEE International Conference on Computer Vision*, pp. 5533–5541, 2017.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

Lin Sun, Kui Jia, Dit-Yan Yeung, and Bertram E Shi. Human action recognition using factorized spatio-temporal convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 4597–4605, 2015.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 4489–4497, 2015.

Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. 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.

Gül Varol, Ivan Laptev, and Cordelia Schmid. Long-term temporal convolutions for action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1510–1517, 2017.

Chengliang Yang, Anand Rangarajan, and Sanjay Ranka. Visual explanations from deep 3d convolutional neural networks for alzheimer's disease classification. In *AMIA Annual Symposium Proceedings*, volume 2018, pp. 1571. American Medical Informatics Association, 2018.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

A DERIVATION AFFINE TRANSFORMATION MATRIX FROM PARAMETERS

In Equation 4 we had: $f(\mathbf{K}_{:,:,1}, \mathbf{\Theta}) = \left(\prod_{b=1}^{l-1} \mathbf{\Theta}_{:,:,b}\right)\mathbf{K}_{:,:,1}$.

First the matrix $\boldsymbol{M}$ is obtained from $\{s, r, t_x, t_y\}$:

$$\boldsymbol{M} = \begin{bmatrix} s\cos r & -s\sin r & t_x s\cos r - t_y s\sin r \\ s\sin r & s\cos r & t_x s\sin r + t_y s\cos r \end{bmatrix} \tag{6}$$

Next, sampling grid $\mathbf{G} \in \mathbb{R}^{n\times m\times 2}$ is created, where for each index $\boldsymbol{G}_{i,j,b}$ the value is computed as:

$$\begin{bmatrix} \boldsymbol{G}_{i,j,0} \\ \boldsymbol{G}_{i,j,1} \end{bmatrix} = \boldsymbol{M} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} \tag{7}$$

where $i \in \{1, ..., n\}$ and $j \in \{1, ..., m\}$. The sampling grid is then used to transform the values from $\mathbf{K}_{:,:,1}$ to $\mathbf{K}_{:,:,2}$:

$$\boldsymbol{K}_{i,j,2} = \boldsymbol{K}_{i,j,1}\Theta_{i,j,1} \tag{8}$$

Where

$$\Theta_{i,j,b} = \max(0, 1 - |\boldsymbol{G}_{i,j,0} - i|)\max(0, 1 - |\boldsymbol{G}_{i,j,0} - j|) \tag{9}$$

In this case we use bilinear interpolation for the transformation.

Compared to a regular 3D kernel which has nml parameters, the 3T kernel only has $nm + 4(l-1)$ parameters.

B PERFORMANCE IN THE LOW DATA REGIME

In a preliminary experiment we created a toy dataset and performed an experiment in which we compared the performance of the models based on how many samples per class the models had access to during training. The results can be seen in Figure 5 and an example from the toy dataset can be seen in Figure . We can see that 3TConv significantly outperforms 3DConv on the low data regime.

C TRAINING DETAILS

Link additional visualizations:

- Additional visualizations

C.1 DATA PRE-PROCESSING

Table 2: Overview of metadata after the data has been processed. UCF101 split 1 was used for train and test.

dataset	classes	height × width	frames	vids in train	vids in validation	vids in test
Jester	27	150 × 224	30	118562	7393	7394
UCF101	101	168 × 224	30	9537	N/A	3783

The models that we used are pre-trained on the ImageNet dataset and were acquired from the torchvision model zoo. In order to make transfer learning possible, resizing and normalization of the data was needed.

The Jester dataset is available for download here: https://20bn.com/datasets/jester. The UCF101 dataset is available here: https://www.crcv.ucf.edu/data/UCF101.php.

For the UCF101 dataset we used split number 1 for the training split and the test split. The test split was then partitioned in half such that the first half of the test split can be used as a validation data split.

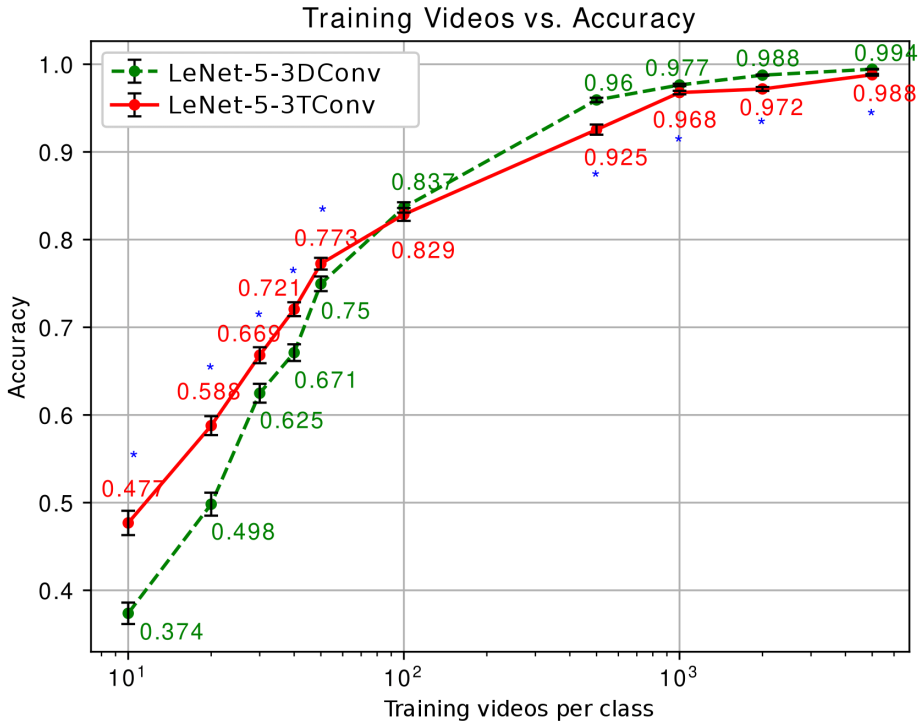

Figure 4: Comparison LeNet-5-3D and LeNet-5-3T on the video-MNIST dataset, while varying the number of samples per class that the models had access to during training.

Resize. In the Jester dataset each frame of a video was resized to a width of 224 pixel and a height of 150 pixels. For the UCF101 dataset split 1, each frame of a video was resized to a width of 224 pixels and a height of 168 pixels.

Frames. For both datasets the number of frames for each video is set to 30 frames. When the length of the original video is greater than 30 frames, frames were dropped such that the interval between dropped frames is equal. If the original lenth was less than 30 frames, only the first and last frames are duplicated.

Normalization. The videos are originally encoded as RGB values tuples between 0 and 255. After loading as such, we scale the values for each value down between 0 and 1 by dividing by 255. Then, for each channel, we subtract and divide by the ImageNet means and standard deviations respectively.

C.2 Model initialization and training

Replacing 2DConvs. In our models, the 2DConv is replaced with 3DConvs and 3TConvs. For both 3DConvs as well as 3TConvs, the temporal depth of the filter is equal to the height and width, i.e, we use unilateral 3D filters. Let us take a look at an example where we replace a 5×5 2DConv filter. For the 3DConv case we replace it with a $5 \times 5 \times 5$ filter. For the 3TConv case we replace it with a 3TConv filter consisting of a 5×5 weight $\mathbf{K}_{:,:,1}$ and a set of $4 \times \{s, r, t_x, t_y\}$ temporal parameters. Note that it is 4 and not 5 because we only need to perform 4 transformations to yield the $5 \times 5 \times 5$ 3D filter from the initial set of weights $\mathbf{K}_{:,:,1}$. However, for both 3DConvNet and 3TConvNet, a 1×1 Conv is always replaced by a $1 \times 1 \times 1$ 3DConv.

Weight transfer. For 3DConv we perform transfer learning from 2D to 3D in the same way as in Carreira & Zisserman (2017). That is, stack the 2D filter weights weights along the temporal

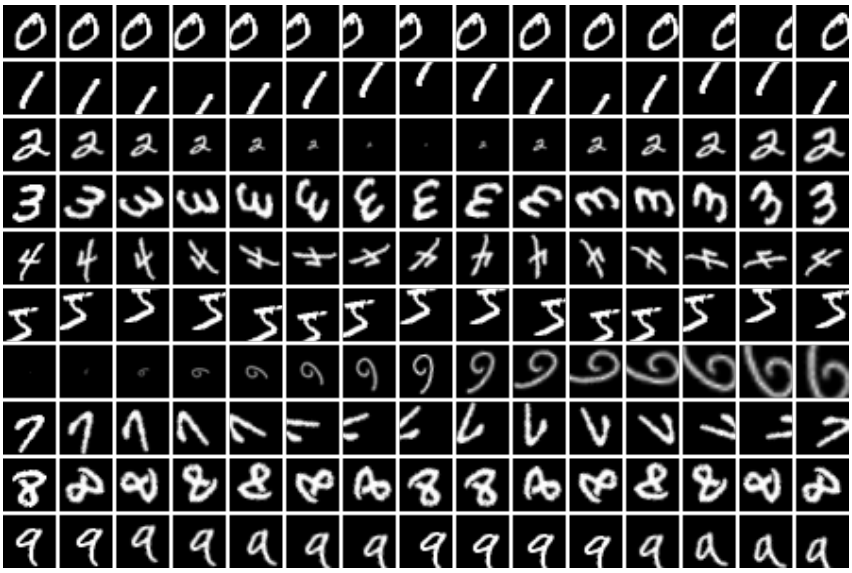

Figure 5: An example of the toy dataset that we created called Video-MNIST. Ten classes in total, each has a different appearance and dynamic behavior. Each sequence contains 30 frames showing an affine transformation on a single original digit moving in a 28×28 pixel frame. The class-specific affine transformations are restricted to scale, rotation and x, y translations.

axis and then divide the values by the length of the temporal axis, e.g., divide by 5 according to the example in the previous paragraph. For the 3TConv the 2D set of weights are simply copied over to K_1 without any modifications. The temporal parameters are initialized with the identity, i.e., $s = 1$, $r = 0$, $t_x = 0$, $t_y = 0$, such that the resulting 3D filter is equal to stacking the 2D weights along the temporal axis. We performed experiments where $\mathbf{K}_{:,:,1}$ is divided by the length of the temporal axis, just like in Carreira & Zisserman (2017), however we found that initializing with unmodified copied 2D weights led to better classification performance on both datasets.

Training. For the Jester dataset, the classes are heavily skewed towards the "Doing other things" class. To correct for this class imbalance we implemented a weighted loss. For both datasts, the models were trained using ADAM and cross-entropy loss. The learning rate for the 3DConvNets was $5e-6$ and the learning rate for the 3TConvNets was $5e-5$. These learning rates were determined experimentally. Models were trained using varying batchsizes of between 10 and 30 samples per batch depending on the availability of memory on our GPUs. We trained each model on a single GPU which took on average 3 days to train on the Jester dataset and 1 day to train on the UCF101 dataset. We used Nvidia GeForce RTX 2080 Ti GPUs. The networks train until the early stopping condition is satisfied, that is, when the validation accuracy starts decreases and the validation loss. This condition is checked for every 5 epochs. Finally, the model that performed best on the validation dataset across the epochs is saved and used in our analyses later. Given that we investigate single models we only train each model once.

Hyper-parameters. Learning rate is informed by the empirical process of trying different learning rates and observing which one gives best results. The range tried for each network was from 0.1 to 5e-7. Batch sizes were chosen such that a good balance was achieved between GPU memory allocation and running times. Batch sizes vary between 16 and 32. Number of epochs are decided by early stopping which is described above. Other parameters such as the number of hidden layers, channels per layer and activation functions adopted from the original model configuration as implemented by the torch model zoo.

Table 3: Training details for all models and datasets.

method	**architecture**	**dataset**	**pre-trained**	**learning rate**	**batch size**	**epochs trained**	**training acc** %	**validation acc** %
3TConv	ResNet18	Jester	yes	5e-5	32	24	98.7	74.5
3TConv	ResNet18	UCF101	yes	5e-5	30	46	100	61.1
3TConv	GoogLeNet	Jester	yes	5e-5	20	29	99.4	84.9
3TConv	GoogLeNet	UCF101	yes	5e-5	25	13	100	63.7
3TConv	ResNet18	Jester	no	5e-5	32	28	71.6	44.4
3TConv	ResNet18	UCF101	no	5e-5	16	37	91.3	31.0
3TConv	GoogLeNet	Jester	no	5e-5	20	34	98.5	74.1
3TConv	GoogLeNet	UCF101	no	5e-5	18	29	100	33.4
3DConv	ResNet18	Jester	yes	5e-6	32	46	99.8	81.6
3DConv	ResNet18	UCF101	yes	5e-6	30	41	100	56.1
3DConv	GoogLeNet	Jester	yes	5e-6	19	26	99.9	89.9
3DConv	GoogLeNet	UCF101	yes	5e-6	20	55	100	58.7