Convolutional Two-Stream Network Fusion for Video Action Recognition

Christoph Feichtenhofer
Graz University of Technology
feichtenhofer@tugraz.at

Axel Pinz
Graz University of Technology
axel.pinz@tugraz.at

Andrew Zisserman
University of Oxford
az@robots.ox.ac.uk

Abstract

Recent applications of Convolutional Neural Networks (ConvNets) for human action recognition in videos have proposed different solutions for incorporating the appearance and motion information. We study a number of ways of fusing ConvNet towers both spatially and temporally in order to best take advantage of this spatio-temporal information. We make the following findings: (i) that rather than fusing at the softmax layer, a spatial and temporal network can be fused at a convolution layer without loss of performance, but with a substantial saving in parameters; (ii) that it is better to fuse such networks spatially at the last convolutional layer than earlier, and that additionally fusing at the class prediction layer can boost accuracy; finally (iii) that pooling of abstract convolutional features over spatiotemporal neighbourhoods further boosts performance. Based on these studies we propose a new ConvNet architecture for spatiotemporal fusion of video snippets, and evaluate its performance on standard benchmarks where this architecture achieves state-of-the-art results. Our code and models are available at http://www.robots.ox.ac.uk/ vgg/software/two_stream_action

1. Introduction

Action recognition in video is a highly active area of research with state of the art systems still being far from human performance. As with other areas of computer vision, recent work has concentrated on applying Convolutional Neural Networks (ConvNets) to this task, with progress over a number of strands: learning local spatiotemporal filters [11, 28, 30]), incorporating optical flow snippets [22], and modelling more extended temporal sequences [6, 17].

However, action recognition has not yet seen the substantial gains in performance that have been achieved in other areas by ConvNets, e.g. image classification [12, 23, 27], human face recognition [21], and human pose estimation [29]. Indeed the current state of the art performance [30, 34] on standard benchmarks such as UCF-101 [24] and HMDB51 [13] is achieved by a combination of ConvNets and a Fisher Vector encoding [20] of hand-crafted
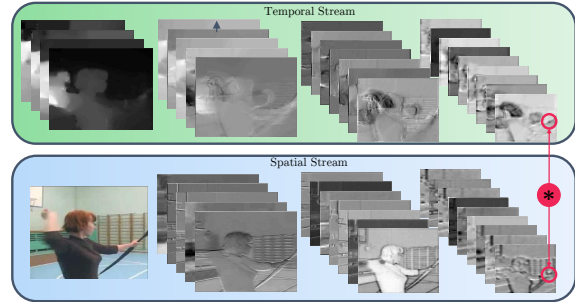

Figure 1. Example outputs of the first three convolutional layers from a two-stream ConvNet model [22]. The two networks separately capture spatial (appearance) and temporal information at a fine temporal scale. In this work we investigate several approaches to fuse the two networks over space and time.

features (such as HOF [14] over dense trajectories [33]).

Part of the reason for this lack of success is probably that current datasets used for training are either too small or too noisy (we return to this point below in related work). Compared to image classification, action classification in video has the additional challenge of variations in motion and viewpoint, and so might be expected to require *more* training examples than that of ImageNet (1000 per class) – yet UCF-101 has only 100 examples per class. Another important reason is that current ConvNet architectures are not able to take full advantage of temporal information and their performance is consequently often dominated by spatial (appearance) recognition.

As can be seen from Fig. 1, some actions can be identified from a still image from their appearance alone (archery in this case). For others, though, individual frames can be ambiguous, and motion cues are necessary. Consider, for example, discriminating walking from running, yawning from laughing, or in swimming, crawl from breast-stroke. The two-stream architecture [22] incorporates motion information by training separate ConvNets for both appearance in still images and stacks of optical flow. Indeed, this work showed that optical flow information alone was sufficient to discriminate most of the actions in UCF101.

Nevertheless, the two-stream architecture (or any previous method) is not able to exploit two very important cues for action recognition in video: (i) recognizing what is mov-

ing where, i.e. registering appearance recognition (spatial cue) with optical flow recognition (temporal cue); and (ii) how these cues evolve over time.

Our objective in this paper is to rectify this by developing an architecture that is able to fuse spatial and temporal cues at several levels of granularity in feature abstraction, and with spatial as well as temporal integration. In particular, Sec. 3 investigates three aspects of fusion: (i) in Sec. 3.1 *how* to fuse the two networks (spatial and temporal) taking account of *spatial* registration? (ii) in Sec. 3.2 *where* to fuse the two networks? And, finally in Sec. 3.3 (iii) how to fuse the networks *temporally*? In each of these investigations we select the optimum outcome (Sec. 4) and then, putting this together, propose a novel architecture (Sec. 3.4) for spatiotemporal fusion of two stream networks that achieves state of the art performance in Sec. 4.6.

We implemented our approach using the MatConvNet toolbox [31] and made our code publicly available at https://github.com/feichtenhofer/twostreamfusion

2. Related work

Several recent work on using ConvNets for action recognition in temporal sequences have investigated the question of how to go beyond simply using the framewise appearance information, and exploit the temporal information. A natural extension is to stack consecutive video frames and extend 2D ConvNets into time [10] so that the first layer learns spatiotemporal features. [11] study several approaches for temporal sampling, including early fusion (letting the first layer filters operate over frames as in [10]), slow fusion (consecutively increasing the temporal receptive field as the layers increase) and late fusion (merging fully connected layers of two separate networks that operate on temporally distant frames). Their architecture is not particularly sensitive to the temporal modelling, and they achieve similar levels of performance by a purely spatial network, indicating that their model is not gaining much from the temporal information.

The recently proposed C3D method [30] learns 3D ConvNets on a limited temporal support of 16 consecutive frames with all filter kernels of size 3×3×3. They report better performance than [11] by letting all filters operate over space and time. However, their network is considerably deeper than [10, 11] with a structure similar to the very deep networks in [23]. Another way of learning spatiotemporal relationships is proposed in [26], where the authors factorize 3D convolution into a 2D spatial and a 1D temporal convolution. Specifically, their temporal convolution is a 2D convolution over time as well as the feature channels and is only performed at higher layers of the network.

[17] compares several temporal feature pooling architectures to combine information across longer time periods. They conclude that temporal pooling of convolutional layers performs better than slow, local, or late pooling, as well as temporal convolution. They also investigate ordered sequence modelling by feeding the ConvNet features into a recurrent network with Long Short-Term Memory (LSTM) cells. Using LSTMs, however did not give an improvement over temporal pooling of convolutional features.

The most closely related work to ours, and the one we extend here, is the two-stream ConvNet architecture proposed in [22]. The method first decomposes video into spatial and temporal components by using RGB and optical flow frames. These components are fed into separate deep ConvNet architectures, to learn spatial as well as temporal information about the appearance and movement of the objects in a scene. Each stream is performing video recognition on its own and for final classification, softmax scores are combined by late fusion. The authors compared several techniques to align the optical flow frames and concluded that simple stacking of $L = 10$ horizontal and vertical flow fields performs best. They also employed multitask learning on UCF101 and HMDB51 to increase the amount of training data and improve the performance on both. To date, this method is the most effective approach of applying deep learning to action recognition, especially with limited training data. The two-stream approach has recently been employed into several action recognition methods [4, 6, 7, 17, 25, 32, 36].

Also related to our work is the bilinear method [15] which correlates the output of two ConvNet layers by performing an outer product at each location of the image. The resulting bilinear feature is pooled across all locations into an orderless descriptor. Note that this is closely related to second-order pooling [2] of hand-crafted SIFT features.

In terms of datasets, [11] introduced the Sports-1M dataset which has a large number of videos (≈1M) and classes (487). However, the videos are gathered automatically and therefore are not free of label noise. Another large scale dataset is the THUMOS dataset [8] that has over 45M frames. Though, only a small fraction of these actually contain the labelled action and thus are useful for supervised feature learning. Due to the label noise, learning spatiotemporal ConvNets still largely relies on smaller, but temporally consistent datasets such as UCF101 [24] or HMDB51 [13] which contain short videos of actions. This facilitates learning, but comes with the risk of severe overfitting to the training data.

3. Approach

We build upon the the two-stream architecture in [22]. This architecture has two main drawbacks: (i) it is not able to learn the pixel-wise correspondences between spatial and temporal features (since fusion is only on the classification scores), and (ii) it is limited in temporal scale as the spatial ConvNet operates only on single frames and the temporal

ConvNet only on a stack of L temporally adjacent optical flow frames (*e.g.* $L = 10$). The implementation of [22] addressed the latter problem to an extent by temporal pooling across regularly spaced samples in the video, but this does not allow the modelling of temporal evolution of actions.

3.1. Spatial fusion

In this section we consider different architectures for fusing the two stream networks. However, the same issues arise when spatially fusing any two networks so are not tied to this particular application.

To be clear, our intention here is to fuse the two networks (at a particular convolutional layer) such that channel responses at the same pixel position are put in correspondence. To motivate this, consider for example discriminating between the actions of brushing teeth and brushing hair. If a hand moves periodically at some spatial location then the temporal network can recognize that motion, and the spatial network can recognize the location (teeth or hair) and their combination then discriminates the action.

This spatial correspondence is easily achieved when the two networks have the same spatial resolution at the layers to be fused, simply by overlaying (stacking) layers from one network on the other (we make this precise below). However, there is also the issue of which *channel* (or channels) in one network *corresponds* to the *channel* (or channels) of the other network.

Suppose for the moment that different channels in the spatial network are responsible for different facial areas (mouth, hair, etc), and one channel in the temporal network is responsible for periodic motion fields of this type. Then, after the channels are stacked, the filters in the subsequent layers must learn the correspondence between these appropriate channels (*e.g.* as weights in a convolution filter) in order to best discriminate between these actions.

To make this more concrete, we now discuss a number of ways of fusing layers between two networks, and for each describe the consequences in terms of correspondence.

A fusion function $f : \mathbf{x}_t^a, \mathbf{x}_t^b, \rightarrow \mathbf{y}_t$ fuses two feature maps $\mathbf{x}_t^a \in \mathbb{R}^{H \times W \times D}$ and $\mathbf{x}_t^b \in \mathbb{R}^{H' \times W' \times D'}$, at time t, to produce an output map $\mathbf{y}_t \in \mathbb{R}^{H'' \times W'' \times D''}$, where W, H and D are the width, height and number of channels of the respective feature maps. When applied to feedforward ConvNet architectures, consisting of convolutional, fully-connected, pooling and nonlinearity layers, f can be applied at different points in the network to implement *e.g.* early-fusion, late-fusion or multiple layer fusion. Various fusion functions f can be used. We investigate the following ones in this paper, and, for simplicity, assume that $H = H' = H''$, $W = W' = W''$, $D = D'$, and also drop the t subscript.

Sum fusion. $\mathbf{y}^{\mathrm{sum}} = f^{\mathrm{sum}}(\mathbf{x}^a, \mathbf{x}^b)$ computes the sum of the two feature maps at the same spatial locations i, j and

feature channels d:

$$y_{i,j,d}^{\mathrm{sum}} = x_{i,j,d}^a + x_{i,j,d}^b, \tag{1}$$

where $1 \leq i \leq H, 1 \leq j \leq W, 1 \leq d \leq D$ and $\mathbf{x}^a, \mathbf{x}^b, \mathbf{y} \in \mathbb{R}^{H \times W \times D}$.

Since the channel numbering is arbitrary, sum fusion simply defines an arbitrary correspondence between the networks. Of course, subsequent learning can employ this arbitrary correspondence to its best effect, optimizing over the filters of each network to make this correspondence useful.

Max fusion. $\mathbf{y}^{\max} = f^{\max}(\mathbf{x}^a, \mathbf{x}^b)$ similarly takes the maximum of the two feature map:

$$y_{i,j,d}^{\max} = \max\{x_{i,j,d}^a, x_{i,j,d}^b\}, \tag{2}$$

where all other variables are defined as above (1).

Similarly to sum fusion, the correspondence between network channels is again arbitrary.

Concatenation fusion. $\mathbf{y}^{\mathrm{cat}} = f^{\mathrm{cat}}(\mathbf{x}^a, \mathbf{x}^b)$ stacks the two feature maps at the same spatial locations i, j across the feature channels d:

$$y_{i,j,2d}^{\mathrm{cat}} = x_{i,j,d}^a \qquad y_{i,j,2d-1}^{\mathrm{cat}} = x_{i,j,d}^b, \tag{3}$$

where $\mathbf{y} \in \mathbb{R}^{H \times W \times 2D}$.

Concatenation does not define a correspondence, but leaves this to subsequent layers to define (by learning suitable filters that weight the layers), as we illustrate next.

Conv fusion. $\mathbf{y}^{\mathrm{conv}} = f^{\mathrm{conv}}(\mathbf{x}^a, \mathbf{x}^b)$ first stacks the two feature maps at the same spatial locations i, j across the feature channels d as above (3) and subsequently convolves the stacked data with a bank of filters $\mathbf{f} \in \mathbb{R}^{1 \times 1 \times 2D \times D}$ and biases $b \in \mathbb{R}^D$

$$\mathbf{y}^{\mathrm{conv}} = \mathbf{y}^{\mathrm{cat}} * \mathbf{f} + b, \tag{4}$$

where the number of output channels is D, and the filter has dimensions $1 \times 1 \times 2D$. Here, the filter $\mathbf{f}$ is used to reduce the dimensionality by a factor of two and is able to model weighted combinations of the two feature maps $\mathbf{x}^a, \mathbf{x}^b$ at the same spatial (pixel) location. When used as a trainable filter kernel in the network, $\mathbf{f}$ is able to *learn* correspondences of the two feature maps that minimize a joint loss function. For example, if $\mathbf{f}$ is learnt to be the concatenation of two permuted identity matrices $\mathbf{1}' \in \mathbb{R}^{1 \times 1 \times D \times D}$, then the ith channel of the one network is only combined with the ith channel of the other (via summation).

Note that if there is no dimensionality reducing conv-layer injected after concatenation, the number of input channels of the upcoming layer is $2D$.

Bilinear fusion. $\mathbf{y}^{\mathrm{bil}} = f^{\mathrm{bil}}(\mathbf{x}^a, \mathbf{x}^b)$ computes a matrix outer product of the two features at each pixel location, followed by a summation over the locations:

$$\mathbf{y}^{\mathrm{bil}} = \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbf{x}_{i,j}^{a\top} \mathbf{x}_{i,j}^b. \tag{5}$$

The resulting feature $\mathbf{y}^{\text{bil}} \in \mathbb{R}^{D^2}$ captures multiplicative interactions at corresponding spatial locations. The main drawback of this feature is its high dimensionality. To make bilinear features usable in practice, it is usually applied at ReLU5, the fully-connected layers are removed [15] and power- and $L2$-normalisation is applied for effective classification with linear SVMs.

The advantage of bilinear fusion is that every channel of one network is combined (as a product) with every channel of the other network. However, the disadvantage is that all spatial information is marginalized out at this point.

Discussion: These operations illustrate a range of possible fusion methods. Others could be considered, for example: taking the pixel wise product of channels (instead of their sum or max), or the (factorized) outer product without sum pooling across locations [18].

Injecting fusion layers can have significant impact on the number of parameters and layers in a two-stream network, especially if only the network which is fused into is kept and the other network tower is truncated, as illustrated in Fig. 2 (left). Table 1 shows how the number of layers and parameters are affected by different fusion methods for the case of two VGG-M-2048 models (used in [22]) containing five convolution layers followed by three fully-connected layers each. Max-, Sum and Conv-fusion at ReLU5 (after the last convolutional layer) removes nearly *half* of the parameters in the architecture as only one tower of fully-connected layers is used after fusion. Conv fusion has slightly more parameters (97.58M) compared to sum and max fusion (97.31M) due to the additional filter that is used for channel-wise fusion and dimensionality reduction. Many more parameters are involved in concatenation fusion, which does not involve dimensionality reduction after fusion and therefore doubles the number of parameters in the first fully connected layer. In comparison, sum-fusion at the softmax layer requires all layers (16) and parameters (181.4M) of the two towers.

In the experimental section (Sec. 4.2) we evaluate and compare the performance of each of these possible fusion methods in terms of their classification accuracy.

3.2. Where to fuse the networks

As noted above, fusion can be applied at any point in the two networks, with the only constraint that the two input maps $\mathbf{x}_t^a \in \mathbb{R}^{H \times W \times D}$ and $\mathbf{x}_t^b \in \mathbb{R}^{H' \times W' \times D}$, at time t, have the same spatial dimensions; *i.e.* $H = H'$, $W = W'$. This can be achieved by using an "upconvolutional" layer [38], or if the dimensions are similar, upsampling can be achieved by padding the smaller map with zeros.

Table 2 compares the number of parameters for fusion at different layers in the two networks for the case of a VGG-M model. Fusing after different conv-layers has roughly the same impact on the number of parameters, as most of these
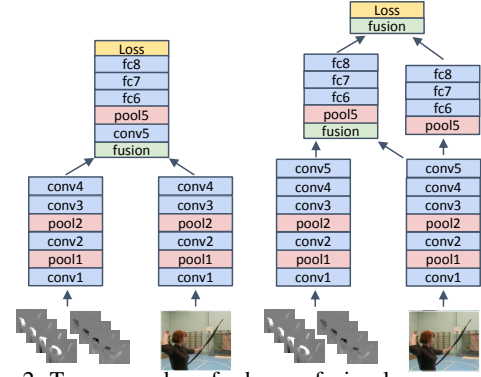

Figure 2. Two examples of where a fusion layer can be placed. The left example shows fusion after the fourth conv-layer. Only a single network tower is used from the point of fusion. The right figure shows fusion at two layers (after conv5 and after fc8) where both network towers are kept, one as a hybrid spatiotemporal net and one as a purely spatial network.

are stored in the fully-connected layers. Two networks can also be fused at two layers, as illustrated in Fig. 2 (right). This achieves the original objective of pixel-wise registration of the channels from each network (at conv5) but does not lead to a reduction in the number of parameters (by half if fused only at conv5, for example). In the experimental section (Sec. 4.3) we evaluate and compare both the performance of fusing at different levels, and fusing at multiple layers simultaneously.
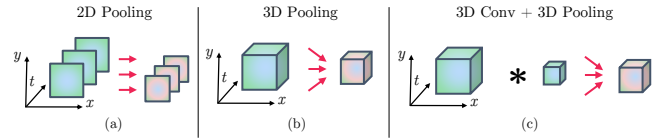

Figure 3. Different ways of fusing temporal information. (a) 2D pooling ignores time and simply pools over spatial neighbourhoods to individually shrink the size of the feature maps for each temporal sample. (b) 3D pooling pools from local spatiotemporal neighbourhoods by first stacking the feature maps across time and then shrinking this spatiotemporal cube. (c) 3D conv + 3D pooling additionally performs a convolution with a fusion kernel that spans the feature channels, space and time before 3D pooling.

3.3. Temporal fusion

We now consider techniques to combine feature maps $\mathbf{x}_t$ over time t, to produce an output map $\mathbf{y}_t$. One way of processing temporal inputs is by averaging the network predictions over time (as used in [22]). In that case the architecture only pools in 2D (xy); see Fig. 3(a).

Now consider the input of a temporal pooling layer as feature maps $\mathbf{x} \in \mathbb{R}^{H \times W \times T \times D}$ which are generated by stacking spatial maps across time $t = 1 \ldots T$.

3D Pooling: applies max-pooling to the stacked data within a 3D pooling cube of size $W' \times H' \times T'$. This is a straightforward extension of 2D pooling to the temporal domain, as illustrated in Fig. 3(b). For example, if three
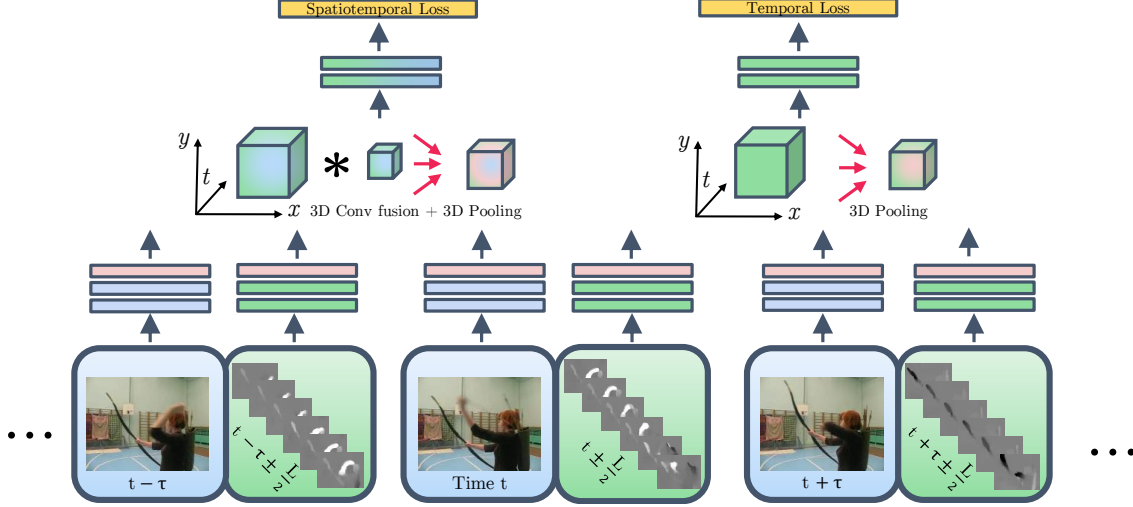
Figure 4. Our spatiotemporal fusion ConvNet applies two-stream ConvNets, that capture short-term information at a fine temporal scale $(t \pm \frac{L}{2})$, to temporally adjacent inputs at a coarse temporal scale $(t + T\tau)$. The two streams are fused by a 3D filter that is able to learn correspondences between highly abstract features of the spatial stream (blue) and temporal stream (green), as well as local weighted combinations in x, y, t. The resulting features from the fusion stream and the temporal stream are 3D-pooled in space and time to learn spatiotemporal (top left) and purely temporal (top right) features for recognising the input video.

temporal samples are pooled, then a $3 \times 3 \times 3$ max pooling could be used across the three stacked corresponding channels. Note, there is no pooling *across* different channels.

3D Conv + Pooling: first convolves the four dimensional input $\mathbf{x}$ with a bank of D' filters $\mathbf{f} \in \mathbb{R}^{W'' \times H'' \times T'' \times D \times D'}$ and biases $b \in \mathbb{R}^D$

$$\mathbf{y} = \mathbf{x}_t * \mathbf{f} + b, \qquad (6)$$

as *e.g.* in [30], followed by 3D pooling as described above. This method is illustrated in Fig. 3(c). The filters $\mathbf{f}$ are able to model weighted combinations of the features in a local spatio-temporal neighborhood using kernels of size $W'' \times H'' \times T'' \times D$. Typically the neighborhood is $3 \times 3 \times 3$ (spatial $\times$ temporal).

Discussion. The authors of [17] evaluate several additional methods to combine two-stream ConvNets over time. They find temporal max-pooling of convolutional layers among the top performers. We generalize max-pooling here to 3D pooling that provides invariance to small changes of the features' position over time. Further, 3D conv allows spatio-temporal filters to be learnt [28, 30]. For example, the filter could learn to center weight the central temporal sample, or to differentiate in time or space.

3.4. Proposed architecture

We now bring together the ideas from the previous sections to propose a new spatio-temporal fusion architecture and motivate our choices based on our empirical evaluation in Sec. 4. The choice of the spatial fusion method, layer and temporal fusion is based on the experiments in sections 4.2, 4.3 and 4.5, respectively.

Our proposed architecture (shown in Fig. 4) can be viewed as an extension of the architecture in Fig. 2 (left) over time. We fuse the two networks, at the last convolutional layer (after ReLU) *into* the spatial stream to convert it into a spatiotemporal stream by using 3D Conv fusion followed by 3D pooling (see Fig. 4, left). Moreover, we *do not truncate* the temporal stream and also perform 3D Pooling in the temporal network (see Fig. 4, right). The losses of both streams are used for training and during testing we average the predictions of the two streams. In our empirical evaluation (Sec. 4.6) we show that keeping both streams performs slightly better than truncating the temporal stream after fusion.

Having discussed how to fuse networks over time, we discuss here the issue of how often to sample the temporal sequence. The temporal fusion layer receives T temporal chunks that are τ frames apart; *i.e.* the two stream towers are applied to the input video at time $t, t+\tau, \ldots t+T\tau$. As shown in Fig. 4 this enables us to capture short scale $(t \pm \frac{L}{2})$ temporal features at the input of the temporal network (*e.g.* the drawing of an arrow) and put them into context over a longer temporal scale $(t + T\tau)$ at a higher layer of the network (*e.g.* drawing an arrow, bending a bow, and shooting an arrow).

Since the optical flow stream has a temporal receptive field of $L = 10$ frames, the architecture operates on a total temporal receptive field of $T \times L$. Note that $\tau < L$ results in overlapping inputs for the temporal stream, whereas $\tau \geq L$ produces temporally non-overlapping features.

After fusion, we let the 3D pooling operate on T spatial feature maps that are τ frames apart. As features may change their spatial position over time, combining spatial

and temporal pooling to 3D pooling makes sense. For example, the output of a VGG-M network at conv5 has an input stride of 16 pixels and captures high level features from a receptive field of 139×139 pixels. Spatiotemporal pooling of conv5 maps that are τ frames distant in time can therefore capture features of the same object, even if they slightly move.

3.5. Implementation details

Two-Stream architecture. We employ two pre-trained ImageNet models. First, for sake of comparison to the original two-stream approach [22], the VGG-M-2048 model [3] with 5 convolutional and 3 fully-connected layers. Second, the very deep VGG-16 model [23] that has 13 convolutional and 3 fully-connected layers. We first separately train the two streams as described in [22], but with some subtle differences: We do not use RGB colour jittering; Instead of decreasing the learning rate according to a fixed schedule, we lower it after the validation error saturates; For training the spatial network we use lower dropout ratios of 0.85 for the first two fully-connected layers. Even lower dropout ratios (up to 0.5) did not decrease performance significantly.

For the temporal net, we use optical flow [1, 37] stacking with $L = 10$ frames [22]. We also initialised the temporal net with a model pre-trained on ImageNet, since this generally facilitates training speed without a decrease in performance compared to our model trained from scratch. The network input is rescaled beforehand, so that the smallest side of the frame equals 256. We also pre-compute the optical flow before training and store the flow fields as JPEG images (with clipping of displacement vectors larger than 20 pixels). We do not use batch normalization [9].

Two-Stream ConvNet fusion. For fusion, these networks are finetuned with a batch size of 96 and a learning rate starting from 10^{-3} which is reduced by a factor of 10 as soon as the validation accuracy saturates. We only propagate back to the injected fusion layer, since full back-propagation did not result in an improvement. In our experiments we only fuse between layers with the same output resolution; except for fusing a VGG-16 model at ReLU5_3 with a VGG-M model at ReLU5, where we pad the slightly smaller output of VGG-M (13×13, compared to 14×14) with a row and a column of zeros. For Conv fusion, we found that careful initialisation of the injected fusion layer (as in (4)) is very important. We compared several methods and found that initialisation by identity matrices (to sum the two networks) performs as well as random initialisation.

Spatiotemporal architecture. For our final architecture described in Sec. 3.4, the 3D Conv fusion kernel $\mathbf{f}$ has dimension $3 \times 3 \times 3 \times 1024 \times 512$ and $T = 5$, i.e. the spatio-temporal filter has dimension $H'' \times W'' \times T'' = 3 \times 3 \times 3$, the $D = 1024$ results from concatenating the ReLU5 from the spatial and temporal streams, and the $D' = 512$ matches

the number of input channels of the following FC6 layer.

The 3D Conv filters are also initialised by stacking two identity matrices for mapping the 1024 feature channels to 512. Since the activations of the temporal ConvNet at the last convolutional layer are roughly 3 times lower than its appearance counterpart, we initialise the temporal identity matrix of $\mathbf{f}$ by a factor of 3 higher. The spatiotemporal part of $\mathbf{f}$ is initialised using a Gaussian of size $3 \times 3 \times 3$ and $\sigma = 1$. Further, we do not fuse at the prediction layer during training, as this would bias the loss towards the temporal architecture, because the spatiotemporal architecture requires longer to adapt to the fused features.

Training 3D ConvNets is even more prone to overfitting than the two-stream ConvNet fusion, and requires additional augmentation as follows. During finetuning, at each training iteration we sample the $T = 5$ frames from each of the 96 videos in a batch by randomly sampling the starting frame, and then randomly sampling the temporal stride (τ) $\in [1, 10]$ (so operating over a total of between 15 and 50 frames). Instead of cropping a fixed sized 224×224 input patch, we randomly jitter its width and height by $\pm 25\%$ and rescale it to 224×224. The rescaling is chosen randomly and may change the aspect-ratio. Patches are only cropped at a maximum of 25% distance from the image borders (relative to the width and height). Note, the position (and size, scale, horizontal flipping) of the crop is randomly selected in the first frame (of a multiple-frame-stack) and then the same spatial crop is applied to all frames in the stack.

Testing. Unless otherwise specified, only the $T = 5$ frames (and their horizontal flips) are sampled, compared to the 25 frames in [22], to foster fast empirical evaluation. In addition we employ fully convolutional testing where the entire frame is used (rather than spatial crops).

4. Evaluation

4.1. Datasets and experimental protocols

We evaluate our approach on two popular action recognition datasets. First, UCF101 [24], which consists of 13320 action videos in 101 categories. The second dataset is HMDB51 [13], which contains 6766 videos that have been annotated for 51 actions. For both datasets, we use the provided evaluation protocol and report the mean average accuracy over the three splits into training and test data.

4.2. How to fuse the two streams spatially?

For these experiments we use the same network architecture as in [22]; i.e. two VGG-M-2048 nets [3]. The fusion layer is injected at the last convolutional layer, after rectification, *i.e.* its input is the output of ReLU5 from the two streams. This is chosen because, in preliminary experiments, it provided better results than alternatives such as the non-rectified output of conv5. At that point the features are

Fusion Method	Fusion Layer	Acc.	#layers	#parameters
Sum [22]	Softmax	85.6%	16	181.42M
Sum (ours)	Softmax	85.94%	16	181.42M
Max	ReLU5	82.70%	13	97.31M
Concatenation	ReLU5	83.53%	13	172.81M
Bilinear [15]	ReLU5	85.05%	10	6.61M+SVM
Sum	ReLU5	85.20%	13	97.31M
Conv	ReLU5	85.96%	14	97.58M

Table 1. Performance comparison of different spatial fusion strategies (Sec. 3.1) on UCF101 (split 1). Sum fusion at the softmax layer corresponds to averaging the two networks predictions and therefore includes the parameters of both 8-layer VGG-M models. Performing fusion at ReLU5 using Conv or Sum fusion does not significantly lower classification accuracy. Moreover, this requires only half of the parameters in the softmax fusion network. Concatenation has lower performance and requires twice as many parameters in the FC6 layer (as Conv or Sum fusion). Only the bilinear combination enjoys much fewer parameters as there are no FC layers involved; however, it has to employ an SVM to perform comparably.

Fusion Layers	Accuracy	#layers	#parameters
ReLU2	82.25%	11	91.90M
ReLU3	83.43%	12	93.08M
ReLU4	82.55%	13	95.48M
ReLU5	85.96%	14	97.57M
ReLU5 + FC8	86.04%	17	181,68M
ReLU3 + ReLU5 + FC6	81.55%	17	190,06M

Table 2. Performance comparison for Conv fusion (4) at different fusion layers. An earlier fusion (than after conv5) results in weaker performance. Multiple fusions also lower performance if early layers are incorporated (last row). Best performance is achieved for fusing at ReLU5 or at ReLU5+FC8 (but with nearly double the parameters involved).

Model	UCF101 (split 1)		HMDB51 (split 1)	
	VGG-M-2048	VGG-16	VGG-M-2048	VGG-16
Spatial	74.22%	82.61%	36.77%	47.06%
Temporal	82.34%	86.25%	51.50%	55.23%
Late Fusion	85.94%	90.62%	54.90%	58.17%

Table 3. Performance comparison of deep (VGG-M-2048) vs. very deep (VGG-16) Two-Stream ConvNets on the UCF101 (split1) and HMDB51 (split1). Late fusion is implemented by averaging the prediction layer outputs. Using deeper networks boosts performance at the cost of computation time.

already highly informative while still providing coarse location information. After the fusion layer a single processing stream is used.

We compare different fusion strategies in Table 1 where we report the average accuracy on the first split of UCF101. We first observe that our performance for softmax averaging (85.94%) compares favourably to the one reported in [22]. Second we see that Max and Concatenation perform considerably lower than Sum and Conv fusion. Conv fusion performs best and is slightly better than Bilinear fusion and simple fusion via summation. For the reported Conv-fusion result, the convolution kernel $\mathbf{f}$ is initialised by identity matrices that perform summation of the two feature maps. Initialisation via random Gaussian noise ends up at a similar performance 85.59% compared to identity matrices (85.96%), however, at a much longer training time. This is interesting, since this, as well as the high result of Sum-fusion, suggest that simply summing the feature maps is already a good fusion technique and learning a randomly initialised combination does not lead to significantly different/better results.

For all the fusion methods shown in Table 1, fusion at FC layers results in lower performance compared to ReLU5, with the ordering of the methods being the same as in Table 1, except for bilinear fusion which is not possible at FC layers. Among all FC layers, FC8 performs better than FC7 and FC6, with Conv fusion at 85.9%, followed by Sum fusion at 85.1%. We think the reason for ReLU5 performing slightly better is that at this layer spatial correspondences between appearance and motion are fused, which would have already been collapsed at the FC layers [16].

4.3. Where to fuse the two streams spatially?

Fusion from different layers is compared in Table 2. Conv fusion is used and the fusion layers are initialised

by an identity matrix that sums the activations from previous layers. Interestingly, fusing and truncating one net at ReLU5 achieves around the same classification accuracy on the first split of UCF101 (85.96% vs 86.04%) as an additional fusion at the prediction layer (FC8), but at a much lower number of total parameters (97.57M vs 181.68M). Fig. 2 shows how these two examples are implemented.

4.4. Going from deep to very deep models

For computational complexity reasons, all previous experiments were performed with two VGG-M-2048 networks (as in [22]). Using deeper models, such as the very deep networks in [23] can, however, lead to even better performance in image recognition tasks [5, 15, 27]. Following that, we train a 16 layer network, VGG-16, [23] on UCF101 and HMDB51. All models are pretrained on ImageNet and separately trained for the target dataset, except for the temporal HMDB51 networks which are initialised from the temporal UCF101 models. For VGG-16, we use TV-L1 optical flow [37] and apply a similar augmentation technique as for 3D ConvNet training (described in Sec. 3.5) that samples from the image corners and its centre [35]. The learning rate is set to 50^{-4} and decreased by a factor of 10 as soon as the validation objective saturates.

The comparison between deep and very deep models is shown in Table 3. On both datasets, one observes that going to a deeper spatial model boosts performance significantly (8.11% and 10.29%), whereas a deeper temporal network yields a lower accuracy gain (3.91% and 3.73%).

Fusion Method	Pooling	Fusion Layers	UCF101	HMDB51
2D Conv	2D	ReLU5 +	89.35%	56.93%
2D Conv	3D	ReLU5 +	89.64%	57.58%
3D Conv	3D	ReLU5 +	90.40%	58.63%

Table 4. Spatiotemporal two-stream fusion on UCF101 (split1) and HMDB51 (split1). The models used are VGG-16 (spatial net) and VGG-M (temporal net). The "+" after a fusion layer indicates that both networks and their loss are kept after fusing, as this performs better than truncating one network. Specifically, at ReLU5 we fuse from the temporal net into the spatial network, then perform either 2D or 3D pooling at Pool5 and compute a loss for each tower. During testing, we average the FC8 predictions for both towers.

4.5. How to fuse the two streams temporally?

Different temporal fusion strategies are shown in Table 4. In the first row of Table 4 we observe that conv fusion performs better than averaging the softmax output (*cf.* Table 3). Next, we find that applying 3D pooling instead of using 2D pooling after the fusion layer increases performance on both datasets, with larger gains on HMDB51. Finally, the last row of Table 4 lists results for applying a 3D filter for fusion which further boosts recognition rates.

4.6. Comparison with the state-of-the-art

Finally, we compare against the state-of-the-art over all three splits of UCF101 and HMDB51 in Table 5. We use the same method as shown above, *i.e.* fusion by 3D Conv and 3D Pooling (illustrated in Fig. 4). For testing we average 20 temporal predictions from each network by densely sampling the input-frame-stacks and their horizontal flips. One interesting comparison is to the original two-stream approach [22], we improve by 3% on UCF101 and HMDB51 by using a VGG-16 spatial (S) network and a VGG-M temporal (T) model, as well as by 4.5% (UCF) and 6% (HMDB) when using VGG-16 for both streams. Another interesting comparison is against the two-stream network in [17], which employs temporal conv-pooling after the last dimensionality reduction layer of a GoogLeNet [27] architecture. They report 88.2% on UCF101 when pooling over 120 frames and 88.6% when using an LSTM for pooling. Here, our result of 92.5% clearly underlines the importance of our proposed approach. Note also that using a single stream after temporal fusion achieves 91.8%, compared to maintaining two streams and achieving 92.5%, but with far fewer parameters and a simpler architecture.

As a final experiment, we explore what benefit results from a late fusion of hand-crafted IDT features [33] with our representation. We simply average the SVM scores of the FV-encoded IDT descriptors (*i.e.* HOG, HOF, MBH) with the predictions (taken before softmax) of our ConvNet representations. The resulting performance is shown in Table 6. We achieve 93.5% on UCF101 and 69.2% HMDB51. This state-of-the-art result illustrates that there is still a de-

Method	UCF101	HMDB51
Spatiotemporal ConvNet [11]	65.4%	-
LRCN [6]	82.9%	-
Composite LSTM Model [25]	84.3%	44.0
C3D [30]	85.2%	-
Two-Stream ConvNet (VGG-M) [22]	88.0%	59.4%
Factorized ConvNet [26]	88.1%	59.1%
Two-Stream Conv Pooling [17]	88.2%	-
Two-Stream ConvNet (VGG-16, [35])	91.4%	58.5%
Two-Stream ConvNet (VGG-16, ours)	91.7%	58.7%
Ours (S:VGG-16, T:VGG-M)	90.8%	62.1%
Ours (S:VGG-16, T:VGG-16, single tower after fusion)	91.8%	64.6%
Ours (S:VGG-16, T:VGG-16)	92.5%	65.4%

Table 5. Mean classification accuracy of best performing ConvNet approaches over three train/test splits on HMDB51 and UCF101. For our method we list the models used for the spatial (S) and temporal (T) stream.

	UCF101	HMDB51
IDT+higher dimensional FV [19]	87.9%	61.1%
C3D+IDT [30]	90.4%	-
TDD+IDT [34]	91.5%	65.9%
Ours+IDT (S:VGG-16, T:VGG-M)	92.5%	67.3%
Ours+IDT (S:VGG-16, T:VGG-16)	93.5%	69.2%

Table 6. Mean classification accuracy on HMDB51 and UCF101 for approaches that use IDT features [33].

gree of complementary between hand-crafted representations and our end-to-end learned ConvNet approach.

5. Conclusion

We have proposed a new spatiotemporal architecture for two stream networks with a novel convolutional fusion layer between the networks, and a novel temporal fusion layer (incorporating 3D convolutions and pooling). The new architecture does not increase the number of parameters significantly over previous methods, yet exceeds the state of the art on two standard benchmark datasets. Our results suggest the importance of learning correspondences between highly abstract ConvNet features both spatially and temporally. One intriguing finding is that there is still such an improvement by combining ConvNet predictions with FV-encoded IDT features. We suspect that this difference may vanish in time given far more training data, but otherwise it certainly indicates where future research should attend.

Finally, we return to the point that current datasets are either too small or too noisy. For this reason, some of the conclusions in this paper should be treated with caution.

References

[1] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *Proc. ECCV*, 2004.

[2] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *Proc. ECCV*, 2012.

[3] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proc. BMVC.*, 2014.

[4] G. Chron, I. Laptev, and C. Schmid. P-CNN: Pose-based CNN features for action recognition. In *Proc. ICCV*, 2015.

[5] M. Cimpoi, S. Maji, and A. Vedaldi. Deep filter banks for texture recognition and segmentation. In *Proc. CVPR*, 2015.

[6] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proc. CVPR*, 2015.

[7] G. Gkioxari and J. Malik. Finding action tubes. In *Proc. CVPR*, 2015.

[8] A. Gorban, H. Indrees, Y. Jiang, A. R. Zamir, I. Laptev, M. Shah, and R. Sukthankar. Thumos challenge: Action recognition with a large number of classes. http://wwwthumos.info/, 2015.

[9] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015.

[10] S. Ji, W. Xu, M. Yang, and K. Yu. 3D convolutional neural networks for human action recognition. *IEEE PAMI*, 35(1):221–231, 2013.

[11] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classication with convolutional neural networks. In *Proc. CVPR*, 2014.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[13] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *Proc. ICCV*, 2011.

[14] I. Laptev, M. Marszałek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *Proc. CVPR*, 2008.

[15] T.-Y. Lin, A. RoyChowdhury, and S. Maji. Bilinear CNN models for fine-grained visual recognition. In *Proc. ICCV*, 2015.

[16] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proc. CVPR*, 2015.

[17] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Proc. CVPR*, 2015.

[18] J. Oh, X. Guo, H. Lee, S. Singh, and R. Lewis. Action-conditional video prediction using deep networks in atari game. In *NIPS*, 2015.

[19] X. Peng, L. Wang, X. Wang, and Y. Qiao. Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice. *CoRR*, abs/1405.4506, 2014.

[20] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher kernel for large-scale image classification. In *Proc. ECCV*, 2010.

[21] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proc. CVPR*, 2015.

[22] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. ICLR*, 2014.

[24] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human actions calsses from videos in the wild. Technical Report CRCV-TR-12-01, UCF Center for Research in Computer Vision, 2012.

[25] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using LSTMs. In *Proc. ICML*, 2015.

[26] L. Sun, K. Jia, D.-Y. Yeung, and B. Shi. Human action recognition using factorized spatio-temporal convolutional networks. In *Proc. ICCV*, 2015.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. CVPR*, 2015.

[28] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *Proc. ECCV*, 2010.

[29] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. In *Proc. CVPR*, 2015.

[30] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3D convolutional networks. In *Proc. ICCV*, 2015.

[31] A. Vedaldi and K. Lenc. MatConvNet – convolutional neural networks for MATLAB. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.

[32] S. Venugopalan, M. Rohrbach, R. Mooney, T. Darrell, and K. Saenko. Sequence to sequence video to text. In *Proc. ICCV*, 2015.

[33] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Proc. ICCV*, 2013.

[34] L. Wang, Y. Qiao, and X. Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *Proc. CVPR*, 2015.

[35] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao. Towards good practices for very deep two-stream convnets. *arXiv preprint arXiv:1507.02159*, 2015.

[36] P. Weinzaepfel, Z. Harchaoui, and C. Schmid. Learning to track for spatio-temporal action localization. In *Proc. ICCV*, 2015.

[37] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime TV-L1 optical flow. In *Proc. DAGM*, pages 214–223, 2007.

[38] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proc. ECCV*, 2014.