REPRESENTATION FLOW FOR ACTION RECOGNITION

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we propose a convolutional layer inspired by optical flow algorithms to learn motion representations. Our *representation flow* layer is a fully-differentiable layer designed to capture the 'flow' of any representation channel within a convolutional neural network for action recognition. Its parameters for iterative flow optimization are learned in an end-to-end fashion together with the other model parameters, maximizing the action recognition performance. Furthermore, we newly introduce the concept of learning 'flow of flow' representations by stacking multiple representation flow layers. We conducted extensive experimental evaluations, confirming its advantages over previous recognition models using traditional optical flows in both computational speed and performance.

1 INTRODUCTION

Activity recognition is an important problem in computer vision with many societal applications including surveillance, robot perception, smart environment/city, and more. Use of video convolutional neural networks (CNNs) have become the standard method for this task, as they can learn more optimal representations for the problem. Two-stream networks (Simonyan & Zisserman, 2014), taking both RGB frames and optical flow as input, provide state-of-the-art results and have been extremely popular. 3-D spatio-temporal CNN models, e.g., (Carreira & Zisserman, 2017), with XYT convolutions also found that such two-stream design (RGB + optical flow) increases their accuracy. Abstracting both appearance information and explicit motion flow benefits the recognition.

However, optical flow is expensive to compute. It often requires hundreds of optimization iterations every frame, and causes learning of two separate CNN streams (i.e., RGB-stream and flow-stream). This requires significant computation cost and a great increase in the number of model parameters to learn. Further, this means that the model needs to compute optical flow every frame even during inference and run two parallel CNNs, limiting its real-time applications.

There were previous works to learn representations capturing motion information without using optical flow as input, such as motion feature networks (Lee et al., 2018) and ActionFlowNet (Ng et al., 2018). However, although they were more advantageous in terms of the number of model parameters and computation speed, they suffered from inferior performance compared to two-stream models on public datasets such as Kinetics (Kay et al., 2017) and HMDB (Kuehne et al., 2011). We hypothesize that the iterative optimization performed by optical flow methods produces an important feature that other methods fail to capture.

In this paper, we propose a CNN layer inspired by optical flow algorithms to learn motion representations for action recognition without having to compute optical flow. Our *representation flow* layer is a fully-differentiable layer designed to capture 'flow' of any representation channels within the model. Its parameters for iterative flow optimization are learned together with other model parameters, maximizing the action recognition performance. This is also done without having/training multiple network streams, reducing the number of parameters in the model. Further, we newly introduce the concept of learning 'flow of flow' representations by stacking multiple representation flow layers. We conduct extensive action classification experimental evaluation of where to compute optical flow and various hyperparameters, learning parameters, and fusion techniques.

2 RELATED WORKS

Capturing motion and temporal information has been studied for activity recognition. Early, hand-crafted approaches such as dense trajectories (Wang et al., 2011) captured motion information by tracking points through time. Many algorithms have been developed to compute optical flow as a way to capture motion in video (Fortun et al., 2015). Other works have explored learning the ordering of frames to summarize a video in a single 'dynamic image' used for activity recognition (Bilen et al., 2016).

Convolutional neural networks (CNNs) have been applied to activity recognition. Initial approaches explored methods to combine temporal information based on pooling or temporal convolution (Ng et al., 2015; Karpathy et al., 2014). Other works have explored using attention to capture sub-events of activities (Piergiovanni et al., 2017). Two-stream networks have been very popular: they take input of a single RGB frame (captures appearance information) and a stack of optical flow frames (captures motion information). Often, the two network streams of the model are separately trained and the final predictions are averaged together (Simonyan & Zisserman, 2014). There were other two-stream CNN works exploring different ways to 'fuse' or combine the motion CNN with the appearance CNN (Feichtenhofer et al., 2016b;a). There were also large 3D XYT CNNs learning spatio-temporal patterns (Carreira & Zisserman, 2017; Xie et al., 2017), enabled by large video datasets such as Kinetics (Kay et al., 2017). However, these approaches still rely on optical flow input to maximize their accuracies.

Recently, there have been works on learning motion representations. Fan et al. (2018) implemented the TVL-1 method using deep learning libraries to increase its computational speed and allow for learning some parameters. The result was fed to a two-stream CNN for the recognition. Several works explored learning a CNN to predict optical flow, which also can be used for action recognition (Dosovitskiy et al., 2015; Hui et al., 2018; Gao et al., 2018; Sun et al., 2018; Ng et al., 2018). Lee et al. (2018) shifted features from sequential frames to capture motion in a non-iterative fashion. Sun et al. (2018) proposed an optical flow guided feature (OFF) by computing the gradients of representations and temporal differences, but required RGB, optical flow and RGB differences to achieve state-of-the-art performance.

Unlike prior works, our proposed model with the *representation flow* layers relies only on the RGB input, learning much fewer parameters while correctly representing motion with the iterative optimization. It is significantly faster than the video CNNs requiring optical flow input, while still performing as good as or even better than the two-stream models. It clearly outperforms existing motion representation methods including (Fan et al., 2018) in both speed and accuracy, which we experimentally confirm.

3 APPROACH

Our method is a fully-differentiable convolutional layer inspired by optical flow algorithms. Unlike traditional optical flow methods, all the parameters of our method can be learned end-to-end, maximizing action recognition performance. Furthermore, our layer is designed to compute the 'flow' of any representation channels, instead of limiting its input to be traditional RGB frames.

3.1 REVIEW OF OPTICAL FLOW METHODS

Before describing our layer, we briefly review how optical flow is computed. Optical flow methods are based on the brightness consistency assumption. That is, given sequential images I_1, I_2, a point x, y in I_1 is located at $x + \Delta x, y + \Delta y$ in I_2, or $I_1(x, y) = I_2(x + \Delta x, y + \Delta y)$. These methods assume small movements between frames, so this can be approximated with a Taylor series: $I_2 = I_1 + \frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y$, where $\boldsymbol{u} = [\Delta x, \Delta y]$. These equations are solved for $\boldsymbol{u}$ to obtain the flow, but can only be approximated due to the two unknowns.

The standard, variational methods for approximating optical flow (e.g., Brox (Brox et al., 2004) and TVL-1 (Zach et al., 2007) methods) take sequential images I_1, I_2 as input. Variational optical flow methods estimate the flow field, $\boldsymbol{u}$, using an iterative optimization method. The tensor $\boldsymbol{u} \in \mathcal{R}^{2\times W\times H}$ is the x and y directional flow for every location in the image. Taking two sequential images as input, I_1, I_2, the methods first compute the gradient in both x and y directions: ∇I_2. The initial flow is set to 0, $\boldsymbol{u} = 0$. Then ρ, the residual, can be computed. For efficiency, the constant part of ρ, ρ_c is pre-computed:

$$\rho_c = I_2 - \nabla_x I_2 \cdot \boldsymbol{u}_x - \nabla_y I_2 \cdot \boldsymbol{u}_y - I_1 \tag{1}$$

The iterative optimization is then performed, each updating $\boldsymbol{u}$:

$$\rho = \rho_c + \nabla_x I_2 \cdot \boldsymbol{u}_x + \nabla_y I_2 \cdot \boldsymbol{u}_y \tag{2}$$

$$\boldsymbol{v} = \begin{cases} \boldsymbol{u} + \lambda\theta\nabla I_2 & \rho < -\lambda\theta|\nabla I_2|^2 \\ \boldsymbol{u} - \lambda\theta\nabla I_2 & \rho > \lambda\theta|\nabla I_2|^2 \\ \boldsymbol{u} - \rho\frac{\nabla I_2}{|I_2|^2} & \text{otherwise} \end{cases} \tag{3}$$

$$\boldsymbol{u} = \boldsymbol{v} + \theta \cdot \text{divergence}(\boldsymbol{p}) \tag{4}$$

$$\boldsymbol{p} = \frac{\boldsymbol{p} + \frac{\tau}{\theta}\nabla\boldsymbol{u}}{1 + \frac{\tau}{\theta}|\nabla\boldsymbol{u}|} \tag{5}$$

Here θ controls the weight of the TVL-1 regularization term, λ controls the smoothness of the output and τ controls the time-step. These hyperparameters are manually set. $\boldsymbol{p}$ is the dual vector fields, which are used to minimize the energy. The divergence of $\boldsymbol{p}$, or backward difference, is computed as:

$$\text{divergence}(\boldsymbol{p}) = \boldsymbol{p}_{x,i,j} - \boldsymbol{p}_{x,i-1,j} + \boldsymbol{p}_{y,i,j} - \boldsymbol{p}_{y,i,j-1} \tag{6}$$

where $\boldsymbol{p}_x$ is the x direction and $\boldsymbol{p}_y$ is the y direction, and $\boldsymbol{p}$ contains all the spatial locations in the image.

The goal is to minimize the total variational energy:

$$E = |\nabla\boldsymbol{u}| + \lambda|I_1 * u + I_1 - I_2| \tag{7}$$

Approaches run this iterative optimization for multiple input scales, from small to large, and use the previous flow estimate $\boldsymbol{u}$ to warp I_2 at the larger scale, providing a coarse-to-fine optical flow estimation. These standard approaches require multiple scales and warpings to obtain a good flow estimate, taking thousands of iterations.

3.2 Representation Flow Layer

Inspired by the optical flow algorithm, we design a fully-differentiable, learnable, convolutional representation flow layer by extending the general algorithm outlined above. The main differences are that (i) we allow the layer to capture flow of any CNN feature map, and that (ii) we learn its parameters including θ, λ, and τ as well as the divergence weights. We also make several key changes to reduce computation time: (1) we only use a single scale, (2) we do not perform any warping, and (3) we compute the flow on a CNN tensor with a smaller spatial size. Multiple scale and warping are computationally expensive, each requiring many iterations. By learning the flow parameters, we can eliminate the need for these additional steps. Our method is applied on lower resolution CNN feature maps, instead of the RGB input, and is trained in an end-to-end fashion. This not only benefits its speed, but also allows the model to learn a motion representation optimized for activity recognition.

We note that the brightness consistency assumption can similarly be applied to CNN feature maps. Instead of capturing pixel brightness, we capture feature value consistency. This same assumption holds because CNNs are designed to be spatially invariant; i.e., they produce roughly the same feature value for the same object as it moves.

Given the input F_1, F_2, a single channel from sequential CNN feature maps (or input image), we compute the gradient by convolving the input feature maps with the Sobel filter:

$$\nabla F_{2x} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * F_2, \;\; \nabla F_{2y} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * F_2 \tag{8}$$

We set $\boldsymbol{u} = 0, \boldsymbol{p} = 0$ initially, each having width and height matching the input, then we can compute $\rho_c = F_2 - F_1$. Next, we run the iterative optimization for a fixed number of iterations, following Eqs. 2-5. To compute the divergence, we zero-pad $\boldsymbol{p}$ on the first column (x-direction) or row (y-direction) then convolve it with weights, w_x, w_y to compute Eq. 6:

$$\text{divergence}(\boldsymbol{p}) = \boldsymbol{p}_x * w_x + \boldsymbol{p}_y * w_y \tag{9}$$

where initially $w_x = [-1 \quad 1]$ and $w_y = \begin{bmatrix}-1\\1\end{bmatrix}$. Note that these parameters are differentiable and can be learned with backpropagation. We compute $\nabla \boldsymbol{u}$ as

$$\nabla \boldsymbol{u}_x = \begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix} * \boldsymbol{u}_x, \;\; \nabla \boldsymbol{u}_y = \begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix} * \boldsymbol{u}_y \tag{10}$$

Algorithm 1 shows the process of our representation flow layer. Our flow layer with multiple iterations could also be interpreted as having a sequence of convolutional layers with each layer behavior dependent on its previous layer. Note that our method is fully differentiable and allows for the learning of all parameters, including (τ, λ, θ) and the divergence weights (w_x, w_y).

Algorithm 1 Method for the representation flow layer

function REPRESENTATIONFLOW(F_1, F_2)
  $\boldsymbol{u} = 0, \boldsymbol{p} = 0$
  Compute image/feature map gradients (Eq. 8)
  $\rho_c = F_2 - F_1$
  for n iterations **do**
    $\rho = \rho_c + \nabla_x F_2 \cdot \boldsymbol{u}_x + \nabla_y F_2 \cdot \boldsymbol{u}_y$
    $\boldsymbol{v} = \begin{cases}\boldsymbol{u} + \lambda\theta\nabla F_2 & \rho < -\lambda\theta|\nabla F_2|^2\\ \boldsymbol{u} - \lambda\theta\nabla F_2 & \rho > \lambda\theta|\nabla F_2|^2\\ \boldsymbol{u} - \rho\frac{\nabla F_2}{|F_2|^2} & \text{otherwise}\end{cases}$
    $\boldsymbol{u} = \boldsymbol{v} + \theta \cdot \text{divergence}(\boldsymbol{p})$
    $\boldsymbol{p} = \frac{\boldsymbol{p} + \frac{\tau}{\theta}\nabla \boldsymbol{u}}{1 + \frac{\tau}{\theta}|\nabla \boldsymbol{u}|}$
  end for
  return $\boldsymbol{u}$
end function

Computing Flow-of-Flow Standard optical flow algorithms compute the flow for two sequential images. An optical flow image contains information about the direction and magnitude of the motion. Applying the flow algorithm directly on two flow images means that we are tracking pixels/locations showing similar motion in two consecutive frames. In practice, this typically leads to a worse performance due to inconsistent optical flow results and non-rigid motion. On the other hand, our representation flow layer is trained for the data, and is able to suppress such inconsistency and better abstract/represent motion by having multiple regular convolutional layers between the flow layers. Fig. 4 illustrates such design, which we confirm its benefits in the experiment section. By stacking multiple representation flow layers, our model is able to capture longer temporal intervals and consider locations with motion consistency.

Representation Flow within a CNN CNN feature maps may have hundreds or thousands of channels and our representation flow layer computes the flow for each channel, which can take significant time and memory. To address this, we apply a convolutional layer to reduce the number of channels from C to C' before the flow layer (note that C' is still significantly more than traditional optical flow algorithms, which are only applied to a single channel, greyscale images). For numerical stability, we normalize this feature map to be in $[0, 255]$, matching standard image values. We found that the CNN features were quite small on average (< 0.5) and the TVL-1 algorithm default hyperparameters are designed for standard images values in $[0, 255]$, thus we found this normalization step important. Using the normalized feature, we compute the flow and stack the x and y flows, resulting in $2C'$ channels. Finally, we apply another convolutional layer to convert from $2C'$ channels to C channels. This is passed to the remaining CNN layers for the prediction. We average predictions from many frames to classify each video, as shown in Fig. 1.

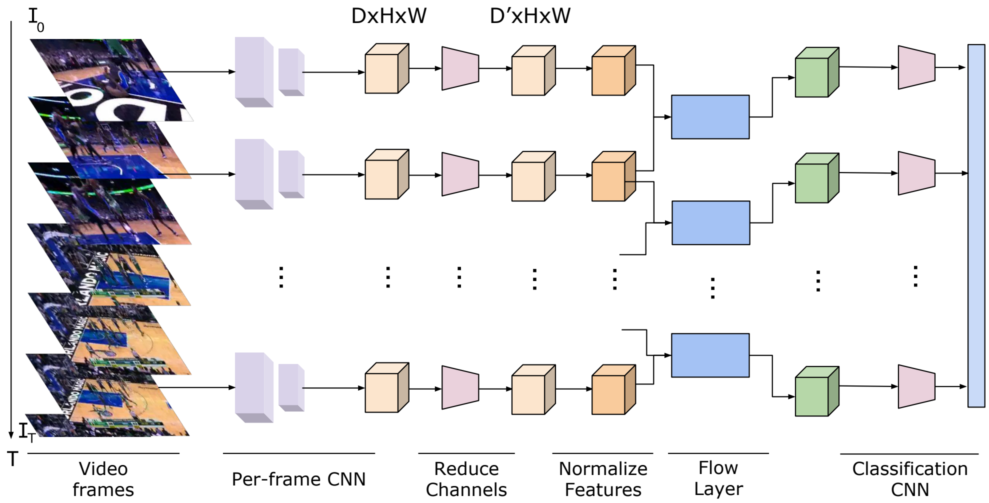

Figure 1: Illustration of a video-CNN with our representation flow layer. The CNN computes intermediate feature maps, and sequential feature maps are used as input to the flow layer. The outputs of the flow layer are used for prediction.

3.3 Activity Recognition Model

We place the representation flow layer inside a standard activity recognition model taking a $T \times C \times W \times H$ tensor as input to a CNN. Here, C is 3 as our model uses direct RGB frames as an input. T is the number of frames the model processes, and W and H are the spatial dims. The CNN outputs a prediction per-timestep and these are temporally averaged to produce a probability for each class. The model is trained to minimize cross-entropy:

$$L(v, c) = -\sum_{i}^{K} (c == i) \log(CNN(v)_i) \tag{11}$$

where v is the video, CNN is the classification CNN and c represents which of the K classes v belongs. That is, the parameters in our flow layers are trained together with the other layers, so that it maximizes the final classification accuracy.

4 Experiments

Implementation details We implemented our representation flow layer in PyTorch and will release our code and models upon publication. As training CNNs on videos is computationally expensive, we used a subset of the Kinetics dataset (Kay et al., 2017) with 100k videos from 150 classes: Tiny-Kinetics. This allowed testing many models more quickly, while still having sufficient data to train large CNNs. For most experiments, we used ResNet-34 (He et al., 2016) with input of size $16 \times 112 \times 112$ (i.e., 16 frames with spatial size of 112). As training video CNNs is computationally expensive, we used this smaller input, which reduces performance, but allowed us to use larger batch sizes and run many experiments more quickly. Our final models are trained on standard 224×224 images. See appendix A for specific training details.

Where to compute flow? To determine where in the network to compute the flow, we compare applying our flow layer on the RGB input, after the first conv. layer, and after the each of the 5 residual blocks. The results are shown in Table 1. We find that computing the flow on the input provides poor performance, similar to the performance of the flow-only networks, but there is a significant jump after even 1 layer, suggesting that computing the flow of a feature is beneficial, capturing both the appearance and motion information. However, after 4 layers, the performance begins to decline as the spatial information is too abstracted/compressed (due to pooling and large spatial receptive field size), and sequential features become very similar, containing less motion information. Note that our HMDB performance in this table is quite low compared to state-of-the-art

Table 1: Computing the optical flow representation after various number of CNN layers. Results are video classification accuracy on our Tiny-Kinetics and LowRes-HMDB51 datasets using 100 iterations to compute the flow representation.

	Tiny-Kinetics	LowRes-HMDB
RGB CNN	55.2	35.5
Flow CNN	35.4	37.5
Two-Stream CNN	57.6	41.5
Flow Layer on RGB Input	37.4	40.5
After Block 1	52.4	42.6
After Block 2	57.4	44.5
After Block 3	59.4	45.4
After Block 4	52.1	43.5
After Block 5	50.3	42.2

Table 2: Comparison of learning different parameters. The flow was computed after Block 3 using 100 iterations.

	Tiny-Kinetics	LowRes-HMDB
None (all fixed)	59.4	45.4
Sobel kernels	58.5	43.5
Divergence (w_x, w_y)	60.2	46.4
τ, λ, θ	59.9	46.2
All	59.2	46.2
Divergence + τ, λ, θ	60.7	46.8

methods due to being trained from scratch using few frames and low spatial resolution (112×112). For the following experiments, unless otherwise noted, we apply the layer after the 3rd residual block.

What to learn? As our method is fully differentiable, we can learn any of the parameters, such as the kernels used to compute image gradients, the kernels for the divergence computation and even τ, λ, θ. In Table 2, we compare the effects of learning different parameters. We find that learning the Sobel kernel reduces performance, but learning the divergence and τ, λ, θ is beneficial.

How many iterations for flow? To confirm that the iterations are important and determine how many we need, we experiment with various numbers of iterations. We compare the number of iterations needed for both learning (divergence+τ, λ, θ) and not learning parameters. The flow is computed after 3 residual blocks. The results are shown in Table 3. We find that learning provides better performance with fewer iterations (similar to the finding in (Fan et al., 2018)), and that iteratively computing the feature is important. We use 10 or 20 iterations in the remaining experiments as they provide good perforamnce and are fast.

Two-stream fusion? Two-stream CNNs fusing both RGB and optical flow features has been heavily studied (Simonyan & Zisserman, 2014; Feichtenhofer et al., 2016b). Based on these works, we compare various ways of fusing RGB and our flow representation, shown in Fig. 2. We compare no fusion, late fusion (i.e., separate RGB and flow CNNs) and addition/multiplication/concatenation

Table 3: Effect of the number of iterations on our Tiny-Kinetics dataset for learning and not learning.

	Not learned	Learned
1 iteration	46.7	49.5
5 iterations	51.3	55.4
10 iterations	52.4	59.4
20 iterations	53.6	60.7
50 iterations	59.2	60.9
100 iterations	59.4	60.7

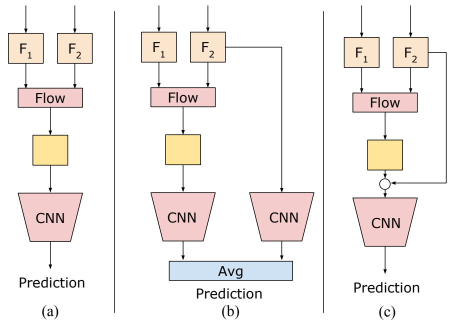

Figure 2: Different approaches to fusing RGB and flow information. **(a)** No fusion **(b)** Late fusion **(c)** The circle represents elementwise addition/multiplication or concatenation.

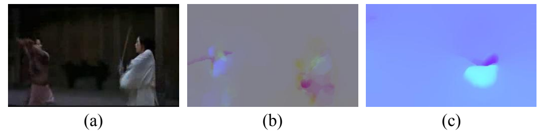

Figure 3: (a) RGB (b) Flow (c) Flow-of-Flow.

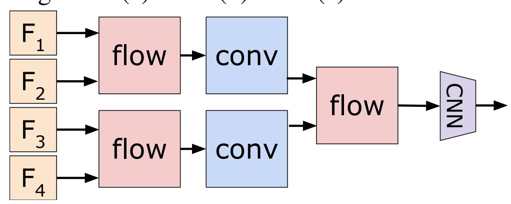

Figure 4: Illustration of how our model computes the FoF.

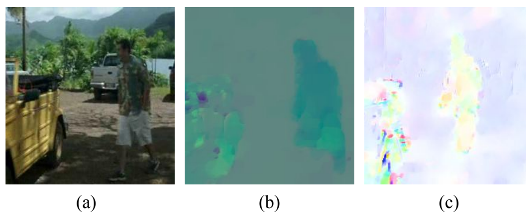

Figure 5: Comparing the results of (b) TVL-1 and (c) our learned flow when applied to RGB images.

fusion. In Table 4, we compare different fusion methods for different locations in the network. We find that fusing RGB information is very important "when computing flow directly from RGB input". However, it is not as beneficial when computing the optical flow of representations as the CNN has already abstracted much appearance information away. We found that concatenation of the RGB and flow features perform poorly compared to the others. We do not use two-stream fusion in any other experiments, as we found that computing the representation flow after the 3rd residual block provides sufficient performance.

Flow-of-flow We can stack our layer multiple times, computing the flow-of-flow (FoF). This has the advantage of combining more temporal information into a single feature. Our results are shown in Table 5. Applying the TVL-1 algorithm twice gives quite poor performance, as optical flow features do not really satisfy the brightness consistency assumption, as they capture magnitude and direction of motion (shown in Fig. 3). Applying our representation flow layer twice performs significantly better than TVL-1 twice, but still worse than our baseline of not doing so. However, we can add a convolutional layer between the first and second flow layer, flow-conv-flow (FcF), (Fig. 4), allowing the model to better learn longer-term flow representations. We find this performs best, as this intermediate layer is able to smooth the flow and produce a better input for the representation flow layer. However, we find adding a third flow layer reduces performance as the motion representation becomes unreliable, due to the large spatial receptive field size.

Table 4: Different fusion methods for flow computed at different locations in the network on our Tiny-Kinetics dataset using 10 iterations and learning flow parameters.

	RGB	1 Block	3 Blocks
None	37.4	52.4	59.4
Late	61.3	60.4	61.5
Add	59.7	57.2	56.5
Multiply	58.3	58.1	57.8
Layer + Multiply	60.1	61.7	61.7
Concat	42.4	48.5	47.6

Table 5: Computing the FoF representation. TVL-1 twice provides poor performance, using two flow layers with a conv. in between provides the best performance. Experiments used 10 iterations and learning flow parameters.

	Tiny-Kinetics
TVL-1 twice	12.2
Single Flow Layer	59.4
Flow-of-Flow	47.2
Flow-Conv-Flow (FcF)	62.3
Flow-Conv-Flow-Conv-Flow	56.5

Table 6: Flow using 3D ResNet-18.

	Tiny-Kinetics
RGB 3D ResNet-18	54.6
TVL-1 3D ResNet-18	37.6
Two-Stream 3D ResNet	57.5
RGB-Only OFF (Sun et al., 2018)	54.8
Input (RGB)	38.5
After Block 1	58.4
After Block 3	59.7

Table 7: Flow using (2+1)D ResNet-18.

	Tiny-Kinetics
RGB (2+1)D ResNet-18	53.4
TVL-1 (2+1)D ResNet-18	36.3
Two-Stream (2+1)D ResNet	55.6
RGB-Only OFF (Sun et al., 2018)	53.7
Input (RGB)	39.2
After Block 1	57.3
After Block 3	60.7

Flow of 3D CNN Feature Since 3D convolutions capture some temporal information, we test computing our flow representation on features from a 3D CNN. As 3D CNNs are expensive to train, we follow the method of Carreira & Zisserman (2017) to inflate a ResNet-18 pretrained on ImageNet to a 3D CNN for videos. We also compare to the (2+1)D method of spatial conv. followed by temporal conv from (Xie et al., 2017), which produces a similar feature combining spatial and temporal information. We find our flow layer increases performance even with 3D and (2+1)D CNNs already capturing some temporal information: Tables 6 and 7. These experiments used 10 iterations and learning the flow parameters. In this, FcF was not used.

We also compared to the OFF (Sun et al., 2018) using (2+1)D and 3D CNNs. We observe that this method does not result in meaningful performance increases using CNNs that capture temporal information, while our approach does.

Comparison to other motion representations We compare to existing CNN-based motion representations methods to confirm that our iterative method to compute a representation is important. For these experiments, when available, we used code provided by the authors and otherwise implemented the methods ourselves. To better compare to existing works, we used $(16\times)$ 224×224 images. MFNet (Lee et al., 2018) captures motion by spatially shifting CNN feature maps, then summing the results, TVNet (Fan et al., 2018) applies a convolutional optical flow method to RGB inputs, and ActionFlowNet (Ng et al., 2018) trains a CNN to jointly predict optical flow and activity classes. We also compare to OFF (Sun et al., 2018) using only RGB inputs. Note that the HMDB performance in (Sun et al., 2018) is only reported using RGB, RGB-diff, and optical flow inputs, here we compare to RGB-only inputs. Our method, which applies the iterative optical flow method on CNN feature maps, performs the best.

Table 8: Comparisons to other CNN-based motion representations, using 10 iterations and learning flow parameters. This is without FcF and two-stream fusion.

	Tiny-Kinetics	HMDB
ActionFlownet (Ng et al., 2018)	51.8	56.2
MFNet (Lee et al., 2018)	52.5	56.8
TVNet (Fan et al., 2018)	39.4	57.5
RGB-OFF (Sun et al., 2018)	55.6	56.9
Ours	61.1	65.4

Table 9: Comparison to the state-of-the-art action classifications. 'HMDB(+Kin)' means that the model was pre-trained on Kinetics before training/testing with HMDB. Missing results are due to those papers not reporting that setting.

	Kinetics	HMDB	HMDB(+Kin)	Run-time (ms)
2D CNNs				
RGB	61.3	53.4	-	225 $\pm$15
Flow	48.2	57.3	-	8039 $\pm$140
Two-stream	64.5	62.4	-	8546 $\pm$147
TVNet (+RGB) (Fan et al., 2018)	-	71.0	-	785 $\pm$21
OFF (RGB Only) (Sun et al., 2018)	-	57.1	-	365 $\pm$26
OFF (RGB + Flow + RGB Diff) (Sun et al., 2018)	-	74.2	-	9520 $\pm$156
Ours (2D CNN + Rep. Flow)	68.5	73.5	76.4	524 $\pm$24
Ours (2D CNN + FcF)	69.4	74.4	77.3	576 $\pm$22
(2+1)D CNNs				
RGB R(2+1)D (Tran et al., 2018)	74.3	-	74.5	471 $\pm$18
Two-Stream R(2+1)D (Tran et al., 2018)	75.4	-	78.7	8623 $\pm$152
Ours ((2+1)D CNN + Rep. Flow)	75.5	-	77.1	622 $\pm$23
Ours ((2+1)D CNN + FcF)	76.1	-	78.2	654 $\pm$21
3D CNNs				
RGB S3D (Xie et al., 2017)	74.7	-	75.9	525 $\pm$22
Two-Stream S3D (Xie et al., 2017)	77.2	-	-	8886 $\pm$162
I3D (RGB) (Carreira & Zisserman, 2017)	71.1	49.8	74.3	594 $\pm$23
I3D (Flow)	63.4	61.9	77.3	8845 $\pm$148
I3D (Two-Stream)	74.2	66.4	80.7	9354 $\pm$154

Computation time We compare our representation flow to state-of-the-art two-stream approaches in terms of run-time and number of parameters. All timings were measured using a single Pascal Titan X GPU, for a batch of videos with size $32 \times 224 \times 224$. The flow/two-stream CNNs include the time to run the TVL-1 algorithm (OpenCV GPU version) to compute the optical flow. All CNNs were based on the ResNet-34 architecture. As also shown in Table 9, our method is significantly faster than two-stream models relying on TVL-1 or other optical flow methods, while performing similarly or better. The number of parameters our model has is half of its two-stream competitors (e.g., 21M vs. 42M in the case of our 2D CNN).

Comparison to state-of-the-arts We also compared our action recognition accuracies with the state-of-the-arts on Kinetics and HMDB. For this, we train our models using $32 \times 224 \times 224$ inputs with the full kinetics dataset, using 8 V100s. We used the 2D ResNet-50 as the architecture. Based on our experiments, we applied our representation flow layer after the 3rd residual block, learned the hyperparameters and divergence kernels, and used 20 iterations. We also compare our flow-of-flow model. Following Szegedy et al. (2016), the evaluation is performed using a running average of the parameters over time. Our results, shown in Table 9, confirm that this approach outperforms existing models using only RGB as inputs and is competitive against expensive two-stream networks. Our model performs the best among those not using optical flow inputs (i.e., among the models only taking $\sim$600ms per video). The models requiring optical flow were more than 10 times slower.

5 Conclusion

We introduced a learnable representation flow layer inspired by optical flow algorithms. We experimentally compared various forms of our layer to confirm that the iterative optimization and learnable parameters are important. Our model outperformed existing methods in both speed and accuracy on standard datasets. We also introduced the concept of 'flow of flow' to compute longer-term motion representations and showed it benefited performance.

References

Hakan Bilen, Basura Fernando, Efstratios Gavves, Andrea Vedaldi, and Stephen Gould. Dynamic image networks for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern*

Recognition (CVPR), 2016.

Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2004.

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 (CVPR)*, 2017.

Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

Lijie Fan, Wenbing Huang, Stefano Ermon Chuang Gan, Boqing Gong, and Junzhou Huang. End-to-end learning of motion representation for video understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Christoph Feichtenhofer, Axel Pinz, and Richard Wildes. Spatiotemporal residual networks for video action recognition. In *Advances in Neural Information Processing Systems (NIPS)*, 2016a.

Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1933–1941, 2016b.

Denis Fortun, Patrick Bouthemy, and Charles Kervrann. Optical flow modeling and computation: a survey. *Computer Vision and Image Understanding*, 134:1–21, 2015.

Ruohan Gao, Bo Xiong, and Kristen Grauman. Im2flow: Motion hallucination from static images for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

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 (CVPR)*, 2016.

Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1725–1732, 2014.

Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2011.

Myunggi Lee, Seung Eui Lee, Sung Joon Son, Gyutae Park, and Nojun Kwak. Motion feature network: Fixed motion filter for action recognition. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2018.

Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4694–4702. IEEE, 2015.

Joe Yue-Hei Ng, Jonghyun Choi, Jan Neumann, and Larry S Davis. Actionflownet: Learning motion representation for action recognition. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018.

AJ Piergiovanni, Chenyou Fan, and Michael S Ryoo. Learning latent sub-events in activity videos using temporal attention filters. In *Proceedings of the American Association for Artificial Intelligence (AAAI)*, 2017.

Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 568–576, 2014.

Shuyang Sun, Zhanghui Kuang, Lu Sheng, Wanli Ouyang, and Wei Zhang. Optical flow guided feature: A fast and robust motion representation for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

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 (CVPR)*, 2018.

Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3169–3176. IEEE, 2011.

Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning for video understanding. *arXiv preprint arXiv:1712.04851*, 2017.

Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint Pattern Recognition Symposium*, pp. 214–223. Springer, 2007.

A Training and Implementation Details

Implementation Details When applying the representation flow layer within a CNN, we first applied a 1x1 convolutional layer to reduce the number of channels from C to 32. CNN feature maps often have hundreds of channels, but computing the representation flow for hundreds of channels is computationally expensive. We found 32 channels to be a good trade-off between performance and speed. The flow layer produces output with 64 channels, x and y flows for the 32 input channels, which are concatenated together. We apply a 3x3 convolutional layer to this representation to produce C output channels. This allows us to apply the rest of the standard CNN to the representation flow feature.

Two-stream networks stack 10 optical flow frames to capture temporal information (Simonyan & Zisserman, 2014). However, we found that stacking representation flows did not perform well. Instead, we computed the flow for sequential images and averaged the predictions from a sequence of 16 frames. We found this outperformed stacking flow representations.

Training Details We trained the network using stochastic gradient descent with momentum set to 0.9. For Kinetics and Tiny-Kinetics, the initial learning rate was 0.1 and decayed by a factor of 10 every 50 epochs. The model was trained for 200 epochs. The 2D CNNs were trained using a batch size of 32 on 4 Titan X GPUs. The 3D and (2+1)D CNNs were trained with a batch size of 24 using 8 V100 GPUs. When fine-tuning on HMDB, the learning rate started at 0.005 and decayed by a factor of 10 every 20 epochs. The network was fine-tuned for 50 epochs. When learning the optical flow parameters, the learning rate for the parameters (i.e., λ, τ, θ, divergence kernels and Sobel filters) was set of $0.01 \cdot \text{lr}$, otherwise the model produced poor predictions. This is likely due to the accumulation of gradients from the many iterations of the algorithm. For Kinetics and Tiny-Kinetics, we used dropout at 0.5 and for HMDB it was set to 0.8.

Testing Details For the results reported in Table 9, we classified actions by applying our model to 25 different random croppings of each video. As found in many previous works, this helps increasing the performance slightly. In all the other experiments (i.e., Tables 1-8), random cropping was not used. Also notice that only the results in Table 9 uses our full model with $32 \times 224 \times 224$ input resolution. The other experiments uses spatially and/or temporally smaller models.