Activity Recognition in Videos with Segmented Streams
A report submitted for the course SCNC2103 Advanced Studies 3

Zixian Cai   Stephen Gould
Australian National University
{u5937495, stephen.gould}@anu.edu.au

Abstract

We investigate a Convolutional Neural Networks (CNN) architecture for activity recognition in short video clips. Applications are ubiquitous, ranging from guiding unmanned vehicles to captioning video clips. While the employment of CNN architectures on large image datasets (such as ImageNet) has been successfully demonstrated in many prior works, there is still no clear answer as to how one can use adapt CNNs to video data. Several different architectures have been explored such as C3D and two-stream networks. However, they all use RGB frames of the video clips as is. In this work, we introduce segmented streams, where each stream consists of the original RGB frames segmented by motion types. We find that after training on the UCF101 dataset, we are able to improve over the original two-stream work by fusing our segmented streams.

1. Introduction

With the wide use of camera-equipped devices (such as ubiquitous smart phones), society is collectively generating more videos than can be feasibly processed manually. Therefore, machine aided video processing has been receiving a great amount of attention from both industry and academia. One important aspect of video processing is action recognition, for example, determining whether a person is crossing a street. It has a wide range of applications, such as guiding unmanned vehicles, captioning video clips, etc.

Convolutional neural networks (CNN) have been successfully applied to action classification in images. Numerous different architectures (*e.g.*, AlexNet [14], ResNet [6], etc.) achieve decent performance on a large image dataset (*i.e.*, ImageNet [2]). Furthermore, their performance is improving over the years. Therefore, it is natural to find out whether these architectures works well on videos. However, it has been shown by Karpathy et al. [12] that naïvely applying CNN on videos using stacked frames leads to poor results.

One can see that video clips present both temporal and spatial information. This additional temporal component might explain why performing activity recognition tasks on videos calls for different architectures. The research community has already seen different proposals on how one can improve [19, 5, 11, 3, 9, 22, 23] architectures in this respect. In this work, we propose a new architecture, whose inputs are segmented frames of different types of motion. We implement this architecture[1] using PyTorch and test its performance on the UCF101 dataset [20].

2. Background

2.1. Activity recognition

A generic description of activity recognition is that it aims to identity what classes of actions one or more agents are performing based on certain kinds of of percept. For this work, we use a narrower definition where the only percept we have is the visual information from a sequence of video frames.

In the context of machine learning, activity recognition is a supervised classification task, where we assign a given input (an image or a video) to one of the classes of actions. For this work, we assume that the classes are disjoint. That is, each input correspond to one and only one class.

There have been many approaches arising from the research community on how one can perform activity recognition. We will discuss some of the related work in Section 5.

2.2. Cross-entropy loss

As we have discussed, activity recognition is a classification task, and an appropriate error function needs to be defined to perform training. A widely used error function for classification is the cross-entropy loss, which is a generalization of the negative log-likelihood loss of the ground-truth label according to the learned model.

[1]Available at `https://gitlab.anu.edu.au/u5937495/scnc2103`.

Assuming we have N data points, let the target vector for each data point be $\vec{t_n}$ using the one-hot encoding of for each class $\mathcal{C}_k$. Let the fitted probability of the n-th data point be y_{nk} for each $\mathcal{C}_k$. The cross-entropy error for that data point is defined as follows

$$E_n = -\sum_{k=1}^{K} t_{nk} \ln y_{nk}. \tag{1}$$

Then, the error for the entire dataset is summed over the set of N points.

2.3. Convolutional neural networks and training

A convolutional neural network (CNN) is a specific type of neural network, which is a nonlinear model architecture constructed as the compositions of a sequence of simple linear and nonlinear functions. Recall that for logistic regression, input features undergo some transformations defined by a set of fixed basis functions. In a neural network, the hidden layers (which is parametrized) of the network define the basis functions, and their parameters are adjusted during the training. Instead of making all the layers fully connected, CNNs have convolution and pooling layers, and a neuron in such layers only connects to a small subset of neurons in the previous layer. CNNs are widely used in tasks related to visual information, and we based our model on ResNet [14, 6] which will be detailed in Section 3.

Training a model (including a CNN) usually amounts to finding a set of (approximately) optimal parameters that minimize a defined loss (*e.g.*, the cross-entropy loss for activity recognition). Since a set of parameters consists of real numbers, we need to perform continuous optimization in the real vector space. In this problem space, gradient-based methods are often used. By taking steps in the direction of negative gradient, we update weights to minimize the loss.

For a large dataset, it is often not feasible to fit the entire dataset in the memory to perform gradient calculation. Stochastic gradient descent (SGD) is a widely used method to address the above problem. LeCun et al. [16] demonstrated that SGD is effective for training with large datasets. SGD can also be performed optionally with momentum [21], making the training of deeper neural networks more tractable.

2.4. Optical flow

To describe the apparent motion between consecutive frames from a sequence of images, one can calculate the corresponding optical flow (*e.g.*, Horn and Schunck [7]).

For each pair of consecutive frames, Let $p = (t_p, x_p, y_p)$ be the space-time index for each pixel p and the difference in time between two consecutive frames is given by dt. The optical flow gives a field of displacement vectors aiming to describe the motion.

$$\text{flow}(p) = (dx, dy)$$

The displacement vector should have the following property, where I is the intensity of the image

$$I(t_p, x_p, y_p) = I(t_p + dt, x_p + dx, y_p + dy). \tag{2}$$

We also note that the displacement vector (dx, dy) can be viewed in the polar coordinate system as (r, φ) where r is the magnitude and φ is the angle of the vector. This view will be important when we describe the segmentation of videos in Section 3.

3. Classifying videos by segmented streams

We humans characterize objects all the times. When we see a moving object, we perceive it being moving fast, moving slowly, moving towards us, moving aways from us, etc. Similarly, given a video clip, we can segment it into multiple parts, where each part contains objects with the same type of motion.

For example, we can segment the video into a fast moving part and a slow moving part. Take a video with a person running on a patch of lawn as an example. In the fast part, we can only see a running person, and in the slow part, we can see green grass in the stationary background. We can reason about the two parts individually and then combine the information to tell what is happening in the original video. Typically the fast segment pertains to the activity and the slow segment provides context, *i.e.*, a swimming pool may provide evidence for a swimming related activity even if we do not see a person swimming. Likewise, seeing a person wearing swimmers and perform an over-arm motion indicates that the person is performing freestyle even if we do not see the swimming pool.

In this section, we are going to formalize the above intuition and show how an architecture can be designed around this idea.

3.1. Overview

First, we describe the overall structure (shown in Fig. 1) of our pipeline so that it is easier for readers to follow the remaining sections, where each part of the architecture will be described in more details.

The architecture starts with calculating optical flow for each video. Then, using the optical flow, we construct a binary mask each type of motion, The binary mask is then smoothed spatially and temporally to a soft mask. When applying the soft mask, we then produce a segmented streams (sequences of frames) for each of the original videos. For each stream, there is a CNN which takes *each frame of the stream* as the input. Finally, a component fuses CNNs together, producing a score vector for *each frame of the video*. To classify *a video*, we perform prediction on each frame of the video, and the result will later be aggregated.

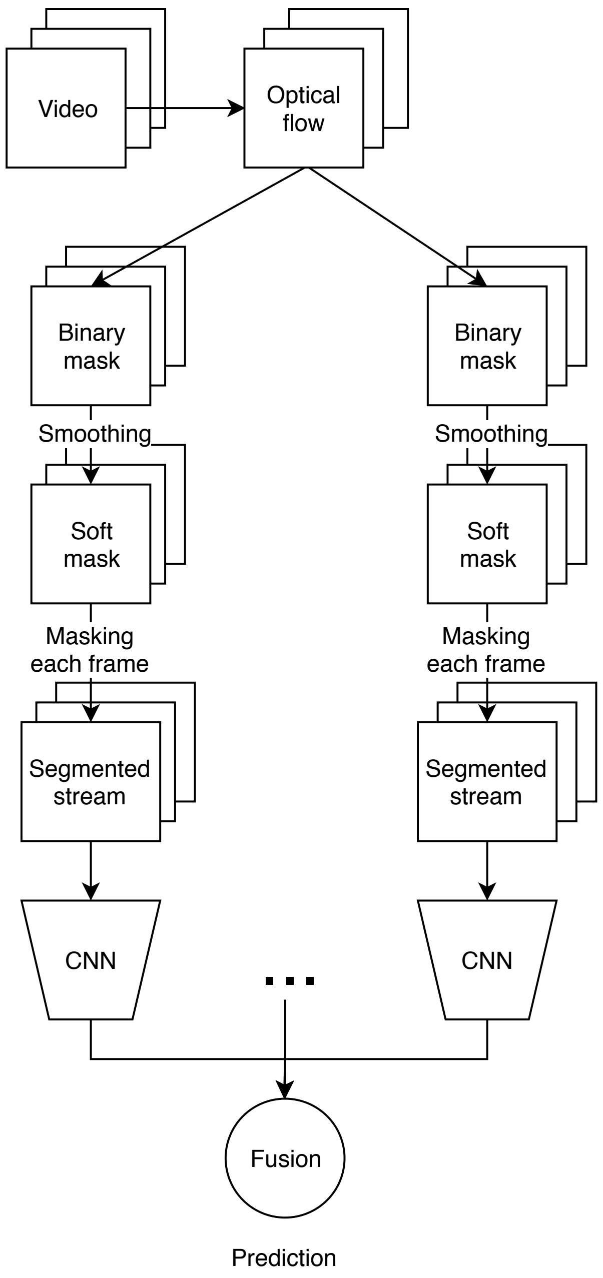
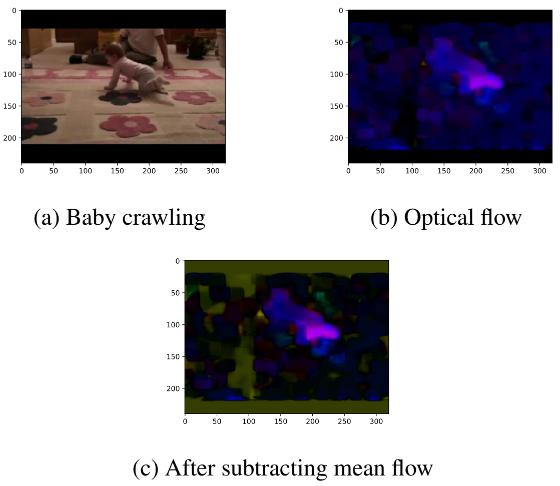


Figure 1: Schematic of the architecture

3.1.1 Optical flow calculation

To assist in determining motion types of each pixel, we first need to estimate the dense optical flow for each pair of consecutive frames. Known dense optical flow estimation algorithms, such as Dual TV-L1 [26, 18] and Farneback's algorithm [4], can be used. Sometimes, however, the direct output from the above output cannot be directly used. When there is camera motion for example, the corresponding displacement caused by camera movement can dominate the optical flow. This increases the difficulty of determining the true movement of the subject of a video.

Mean flow subtraction has been proposed by Simonyan and Zisserman [19] as a method to mitigate the above issue. However, the subtraction of mean flow itself can introduce new errors. Consider the example shown in Fig. 2a, where a baby is crawling and the top and the bottom of the frame is a non-moving black padding. The colour of each pixel of the optical flow indicates the direction of the vector, while the brightness indicates the magnitude of the vector. The mean flow points toward the direction the baby is crawling as shown in Fig. 2b. After subtracting the mean flow, the patch of non-moving background can "appear" to be moving as shown in Fig. 2c. This poses as a problem when we use the optical flow to perform the segmentation. For example, the non-moving background might appear in the fast stream, when it is meant to appear in the slow stream.



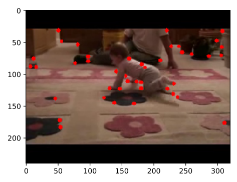
(a) Baby crawling (b) Optical flow

(c) After subtracting mean flow

Figure 2: Problematic mean flow subtraction

To workaround this issue, we propose an alternative way to estimate the dense optical flow. For a pair of consecutive frame, we first choose a set of points of interest to track in the first frame as shown in Fig. 3a. A corner detection algorithm can be used, or one can simply just choose a grid of points. Then, we can choose a sparse optical flow algorithm, which estimate the new location of each point of interest in the second frame. By subtracting the old locations from the corresponding new locations, we have displacement vectors for points we are tracking as shown in Fig. 3b.

Next, a dense optical flow can be constructed using an interpolation algorithm. An edge aware interpolation algorithm usually produces better result, but it is also usually slower as bilateral filtering is involved. We believe simple Gaussian interpolation would suffice. First, we create a flow tensor F of size $2 \times H \times W$ and a normalization matrix M of size $H \times W$. Matrix M is initialize


(a) Points tracked

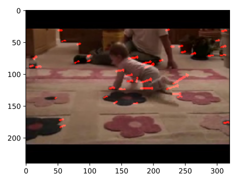
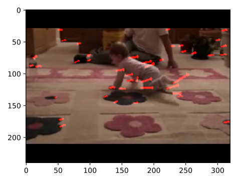

(b) Sparse optical flow

Figure 3: Estimating the sparse optical flow

with a very small value, such as $1e^{-6}$. Then for each of the displacement vector (dx, dy) at location (x_n, y_n) for the sparse optical flow, we perform the following. Let $\mathcal{N}\left((x,y)\middle|(x_n,y_n), \begin{bmatrix} W/2 & 0 \\ 0 & W/2 \end{bmatrix}\right)$ be a Gaussian centered at (x_n, y_n). Then, we evaluate the Gaussian for each pixel location of a frame, obtaining a matrix A of size $H \times W$. We add A to M, and add $\begin{bmatrix} dx\,A & dy\,A \end{bmatrix}$ to F. After iterating through all displacement vectors of the sparse flow, element-wise division of F by M gives an estimation of the dense optical flow as shown in Fig. 4b. In this dense optical flow, the displacement vector for a pixel location is essentially "weighted average" of nearby displacement vectors of the sparse flow.


(a) Sparse optical flow. An arrow represents a displacement vector, where the arrow length is not to scale.

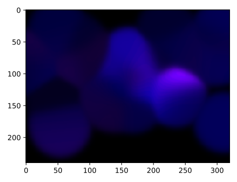

(b) Dense optical flow. Colour represents the direction of movement. The brighter an area is, the faster it moves.

Figure 4: Gaussian interpolation

3.2. Segmentation by optical flow

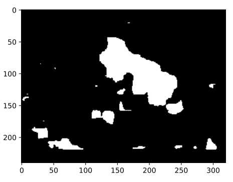
Optical flow can describe the apparent motion of the input video as shown in Section 2.4. The fields of displacement vectors in optical flow express the vertical and horizontal movements of pixels between pairs of consecutive frames. As described before, we can view these vectors in the polar coordinate system, which yields the representation consisting of the magnitude and direction.

The above representation gives the opportunity for segmenting videos, where the users of the architecture can produce a binary mask for any type of motion they are interested in. For a given video and a type of motion, the corresponding binary mask is defined as follows. We use $p = (t_p, x_p, y_p)$ to denote the time-space index for each pixel from the set of all pixels P. The binary mask $\mathrm{BM}_{\text{flow}} : P \to \{0, 1\}$ is an indicative function for a type of motion given the optical flow. Note that for a video with T frames of size H by W, the optical flow is only calculated for the $T-1$ pairs consecutive frames. Therefore, the binary mask $\mathrm{BM}_{\text{flow}}$ is also of size $T-1 \times H \times W$.

Here is an example of constructing the binary mask for fast moving objects. Assuming the polar coordinate view of displacement vectors $\text{flow}(p) = (r, \varphi)$

$$\mathrm{BM}_{\text{flow}}(p) = \begin{cases} 1 & \text{if } r \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Using the above baby crawling example, an example of the fast and slow mask is shown in Fig. 5, where the white pixel indicates that the corresponding pixel in the original video will be included in the segmented stream.


(a) Fast mask

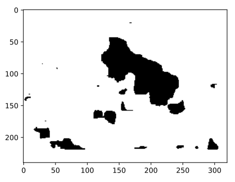
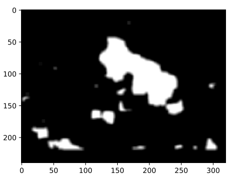

(b) Slow mask

Figure 5: Binary mask

Due to camera motion and noise in the video, we apply spatial and temporal smoothing on the obtained binary mask. For spatial smoothing, we use the normalized box filter on each of the frame of the binary mask. For temporal smoothing, we use the 1D Gaussian filter along the time axis of the binary mask. After smoothing, we get a soft mask where each pixel is assigned value between 0 and 1 as shown in Fig. 6. The soft mask is of the same size as the binary mask.


(a) Fast mask

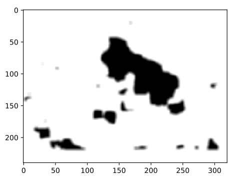

(b) Slow mask

Figure 6: Soft mask

We are now ready to generate a segmented stream for each type of motion, which is a sequence of derived frames from the original frames. Since each of the original frame is of consists of three channels (red, green and blue), the soft mask is applied as an element-wise product repeatedly across all three channels. As you can see in Fig. 7, the fast part highlights the crawling baby while the slow part shows the background as expected.

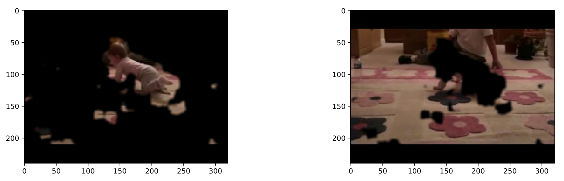

(a) Fast part (b) Slow part

Figure 7: One frame of the fast and slow streams

3.3. Adapt CNN for images

We adapt ResNet [6], which is a CNN designed for tasks on images as described in Section 2.3. It is a common practise to pre-train a CNN on an image dataset (*e.g.* ImageNet) for weight initialization. Image datasets usually contain images of three channels (red, green and blue), which dictates the size of the input tensor to the network.

However, unlike images, our input to the network does not necessarily have three channels. For example, if we are to train the network using the vertical and horizontal components of the optical flow, which means this input to the network has two channels. This poses a problem as to how pre-trained weights can be transformed. We use the cross-modality pre-training proposed by Wang et al. [25]. The average of weights of the RGB channels are calculated, and this average is replicated by the number of desired channels.

3.4. Fusion of multiple streams

For each type of motion, we have a corresponding segmented stream and trained CNN. Each of the CNNs is capable of performing classification on its own. However, we expect to see the improvement of predicative performance if multiple streams are combined. We investigate the following two strategies.

Fusion of class scores One of the fusion methods used in the two-stream work [19] is by averaging the softmax scores. For each CNN, the output softmax scores is a length-K vector where each entry is between zero and one and the entires sum to one. The network predicts the class $\mathcal{C}_k$ being more likely if the corresponding score is higher. The average fusion simply takes the average of the softmax score vector from each CNN. This new vector of scores can then be used in further prediction.

Fusion of feature vectors The above method is simple to implement and there is no extra training involved. However, this primitive method may not be able to the capture and combine the different features learnt by different CNNs from different streams.

We propose a different fusion method that the feature vectors after the last fully connected layer of each CNN are used. We train a multilayer perceptron (MLP) on the concatenated feature vectors. We choose an MLP as it is able to learn non-linear combination of input features. The MLP has two hidden fully connected layer with batch normalization [8] and ReLU as the activation.

End-to-end training is also used, where the CNNs are put directly in front of the MLP to allow fine-tuning.

3.5. Implementation details

Compare with the two-stream architecture [19], we change how the input of each stream is constructed. Therefore, it is natural to base our implementation on that.

3.5.1 Data augmentation

We use several standard data augmentation techniques. In training, a random 224×224 subframe is cropped from an input frame, and then undergoes random horizontal flipping. In testing, an input frame is scaled to be of size 224×224. In both training and testing, the 224×224 frame is then normalized using the mean and standard deviation calculated from the ImageNet dataset.

We also consider optional greyscaling, where only the brightness of the input frame is retained. We follow ITU-R Recommendation BT.601[2], where the brightness (luma) is calculated as follows

$$Y = 0.299R + 0.587G + 0.114B \tag{4}$$

where R, G and B are the intensity of the red, green, and blue channels respectively.

3.5.2 Preprocessing optical flow

Optical flow estimation is a computationally intensive part of our architecture. To save time, we precompute and save the optical flow for each video.

For Dual TV-L1 and Farneback's algorithm, we use the implementation in OpenCV out-of-the-box.

For converting sparse optical flow to dense optical flow, we use the Shi-Tomasi corner detector [10] in OpenCV to find points to track. The corresponding sparse optical flow is calculated with the iterative Lucas-Kanade method [17], which can also be found in OpenCV. Finally, the interpolation is performed as described in Section 3.1.1.

[2]`https://www.itu.int/rec/R-REC-BT.601/en`

3.5.3 Smoothing

As described Section 3.2, the binary masks derived from optical flow undergo temporal and spatial smoothing. We choose the following parameter based on how the resulting segmented streams look visually nice.

For spatial smoothing, we use a 5×5 sized normalized box filter on each frame of the binary mask. Then, for temporal smoothing, we choose to apply a Gaussian filter with $\sigma = 1.5$ along the time axis.

3.5.4 Training

When training, we adjust the parameters to minimize the cross-entropy loss as described in Section 2.2. We optimize the parameters using SGD with 0.9 as the momentum as described in Section 2.3. The learning rate is reduced adaptively when the loss does not show a decreasing trend for a certain number of epochs. Additionally, when training the MLP for fusion, we use weight decay to alleviate overfitting.

For each iteration of an epoch, we first sample some videos. For each video, we then sample three frames from it. Each frame then undergoes data augmentation described in Section 3.5.1. These frames together become a minibatch used for that iteration.

3.5.5 Testing

We follow the same testing protocol stated in [19]. During testing, we first sample some videos. For a given video, we pick frames with equal temporal spacing between them (19 in our case). Similar to what we do for training, we perform data augmentation for each frames, and construct a minibatch.

After processing all the minibatches and therefore all the testing videos, the class scores for each video are then calculated by averaging the class scores of all the frames that are picked.

4. Experiments

In this section, we evaluate the predicative capability of our architecture.

4.1. Dataset

We use UCF101 dateset [20] for training and testing of our architecture. We choose UCF101 because of the following properties.

1. The dataset size does not require more time and computational resource than what we have.

2. Videos in UCF101 are found on the internet, and they are mostly not professionally and purposefully filmed. These videos usually lower fidelity and quality, where camera motion, less-ideal lighting conditions, *etc*. can be found. This can help distinguish architectures that can handle various different conditions from those that cannot.

3. Although UCF101 is smaller than some newer datasets, such as the Kinetics dataset [13], it still contains more action classes than some of other widely used datasets, such as HMDB51 [15].

We acknowledge that UCF101 does have a limited diversity in terms of video clips available. For each action class, there are multiple clips, but they are sampled from a smaller number of original videos, making the actually population of the dataset lacks in variety.

We use the top-1 and top-5 precision as the performance metric. That is, a prediction is correct if the target label is one of action classes that attain one or five highest class scores. In this paper, we use "Prec@1" and "Prec@5" to denote the top-1 and top-5 precision respectively.

The train/test splits published on the UCF101 website[3] is used. We acknowledge that only the first train/test split is used due to our time constraint.

4.2. Baseline architecture

Our baseline architecture is the two-stream architecture [19]. This work achieved competitive performance compared with the state-of-the-art deep architectures and shallow hand-crafted models on UCF101 and HMDB51. We argue that this is suitable baseline for us to compare with, as the overview architecture is the same except how we choose to construct different streams.

We *do not* intend to compete with the current state-of-the-art on UCF101.

1. I3D [1] pre-trains on a much larger dataset (*i.e.*,the Kinetics dataset) and is then used on UCF101.

2. Some architectures combine IDT [24], where trajectory information is incorporated. This requires matching features points between all frames to estimate the trajectories.

3. Temporal Segment Networks[4] [25] divides an input video into multiple "segments" along the time axis. Each "segment" contains a two-stream architecture and all segments are fused using the segmental consensus function.

The above architectures all base on the two-stream architecture. But they change orthogonal parts of the two-stream

[3]`https://www.crcv.ucf.edu/data/UCF101.php`

[4]Their use of "segment" is different from ours.

architecture compared with ours. We argue that our idea can be easily adapted to the above architectures.

We based our implementation on a baseline implementation[5] of the two-stream architecture [19]. To establish a fair comparison, we share the same underlying CNNs (*i.e.*, ResNet-101 [6]), preprocessing techniques, and training parameters with the baseline implementation.

In the two-stream architecture, the spatial stream CNN is trained on individual RGB frames, and the temporal stream CNN is trained on stacked optical flow. The two stream can be fused using averaging. We try to reproduce the work and the obtained the precision shown in Table 1. These numbers are closed to what is reported by the author of the baseline implementation.

Method	Prec@1	Prec@5
Spatial	81.0%	95.6%
Temporal	79.2%	94.9%
Average fusion	**87.8%**	**97.9%**

Table 1: Precision of the two-stream work on UCF101

4.3. Fast and slow streams

As we have stated in Section 3.2, our architecture is flexible that users can decide what types of motion they are interested in. The corresponding binary masks can then be produced and use to segment the original video into multiple streams.

To illustrate how our architecture might be used, we decide to focus on fast and slow streams, and evaluate the performance of the architecture. As stated in the beginning of Section 3, the idea behind using fast and slow streams is that the slow part of a video provides contextual information, and the fast part of a video shows the objects in motion.

4.3.1 Training on a fast and on a slow stream

To segment a video into fast and slow streams, we need a criteria as to under what conditions a pixel is deemed moving fast. For this basic experiment of training on a fast and on a slow stream, we use a threshold of 1.17 (measured in the number pixels) across all videos. That is, if the corresponding displacement vector in the optical flow of a pixel has a magnitude greater than or equal to 1.17, it is moving fast. We pick this value by trying different threshold values, testing on a small sample of videos, and choosing one that gives results visually close to what we are expecting. More methods of doing thresholding will be discussed in Section 4.3.3.

[5]`https://github.com/jeffreyhuang1/two-stream-action-recognition`

Method	Prec@1	Prec@5
Fast	71.3%	90.7%
Slow	77.1%	94.2%
Average fusion	**80.6%**	**96.1%**

Table 2: Precision of fast and slow streams on UCF101

We train the fast stream CNN and the slow stream CNN and obtain the precision shown in Table 2. The performance of the above fast and slow streams are worse than both the spatial and temporal streams in the two-stream work (shown in Table 1). After average fusion of the fast and slow stream, it only performs marginally better than the temporal stream, and is still behind the precision of the spatial stream. This inferior performance can be explained by that the fast and slow streams are effectively just the spatial stream with some information masked out. Therefore, it is unlikely that the fast or slow stream on its own can perform better than the spatial stream. Furthermore, the fast and slow streams might learn features that are not well captured by the simple averaging fusion, making the increase in the precision after fusion not substantial.

4.3.2 Different fusion methods

As discussed in the previous section, the simple average fusion of class scores may not be able to capture the different features learnt by different CNNs of different streams. In this section, we evaluate whether we can benefit from using a non-linear classifier for fusion. Specifically, we train an MLP (denoted "MLP with BN" in the table) as described in Section 3.4 on the concatenated feature vectors from the fast and slow stream CNNs. During the training of the MLP, we notice that the network quick saturated on the training set, achieving nearly perfect prediction. However, the loss on the test set did not decrease much after few epochs. We suspect that the MLP is overfitting on the training set, and therefore we trained another MLP with weight decay set to $1e^{-3}$ (denoted "end-to-end MLP with BN, WD"). We also allow end-to-end training to fine tune the fast stream CNN and the slow stream CNN.

Method	Prec@1	Prec@5
Average fusion	80.6%	96.1%
MLP with BN	81.6%	**96.2%**
End-to-end MLP with BN, WD	**82.5%**	95.8%

Table 3: Precision of fusing fast and slow streams with an MLP on UCF101. BN stands for batch normalization and WD stands for weight decay.

Table 3 summarized the precisions of MLPs we trained. As we can see from the table, using an MLP to fuse the fast stream CNN and the slow stream CNN does improve over the average fusion. Furthermore, allowing end-to-end training and regularizing the network with weight decay, we are able to get better precision out of the MLP fusion method. However, as one can notice, after deploying an MLP, the fusion of fast and slow stream CNNs still performs worse than the simple average fusion of spatial and temporal streams of the two-stream architecture.

In the next few sections, we will discuss more approaches attempting to improve the precision of our architecture.

4.3.3 Different thresholding

As discussed in Section 4.3.1, we use an absolute threshold value *for all videos* to segment them into fast and slow streams. However, this might not always be the best option. For example, the typical moving speeding of objects in different types of actions can be different. Assuming the same zoom level and viewing perspective, a person running moves slower (measured in pixels per frame) than a car driving at speed. If we use a threshold value that is just enough to segment the moving car in the fast stream, the value will not work for the running person. Conversely, a threshold that works for a running person is also likely to put objects in the fast stream, that are moving slowly alongside the car and not the subject of the action.

To address this problem, we adaptively choose the threshold *per video* using the percentile. For a given percentile, we check all displacement vectors in the optical flow, and mark those higher than the given percentile. Then, the corresponding pixels in the original video are the fast moving ones.

Method	Prec@1	Prec@5
Fast (absolute)	71.3%	90.7%
Fast 90th	67.3%	88.4%
Fast 95th	56.4%	80.8%
Slow (absolute)	77.1%	94.2%
Slow 90th	78.7%	94.3%
Slow 95th	80.4%	95.5%
Fast (absolute) + slow 95th	**82.8%**	**96.4%**

Table 4: Precision of fast and slow streams using percentile-based thresholding on UCF101

Due to our time constraint, we only test the 90th and 95th percentile. As we can see in Table 4, switching from the absolute threshold value to the percentile-based, there is a decreasing trend on the fast stream and an increasing trend on the slow stream in the precision values. With a higher percentile value, the fast stream performs worse and the slow stream performs better. This might be due to that increasing the threshold makes more information retaining in the slow stream and less information retaining in the fast stream. We also test all combinations of fast streams and slow streams using average fusion, with the best one being fusing the fast stream with the absolute threshold with the slow stream with the 95th percentile threshold. That is, by switching to a slow stream with better precision (based on 95th percentile), we are able to achieve better precision than the MLP fusion with the fast stream unchanged.

4.3.4 Greyscaling

We also want to find out whether the appearance of the fast moving objects matters. We use greyscaling (see Section 3.5.1) on the fast stream, only keeping the brightness and texture of the fast moving objects.

Method	Prec@1	Prec@5
Fast	71.3%	90.7%
Fast grey	67.8%	87.3%
Fast + slow	80.6%	**96.1%**
Fast grey + slow	**81.1%**	95.8%

Table 5: Precision of fast and slow streams with greyscaling on the fast stream on UCF101

As we can see in Table 5, after performing greyscaling on the fast stream, the precision of the fast stream on its own decreases compared with the RGB one. However, it is interesting that after average fusion with the same slow stream, the greyscale fast stream has a slight edge over the RGB stream. This suggests that the performance of fused stream does not entirely depend on the performance of individual stream. It is perhaps more important for a stream to provide information that others do not have.

4.3.5 Fusion with the two-stream work

In the above sections, we discussed ways to construct fast and slow streams and the precision results we obtained. Recall that the two-stream work has a spatial stream and a temporal stream, and the corresponding CNNs has the same output format as our CNNs. It would be interesting to see whether some combinations of these four streams will lead to improvement over the two-stream work.

Due the large number of combinations we have, it is not feasible to training MLPs for fusion and the simple average fusion is used. Thus, the result below might not be fair for streams learning features that cannot be well fused with averaging. However, we are able to improve over the two-stream work, with the best combination being the spatial stream, the temporal stream, and the slow stream based on 90th percentile thresholding. The combination achieves

89.7% top-1 precision and 98.7% top-5 precision, improving the precision obtained by the two-stream work (87.8% and 97.9%).

Let's compare the combination against the two-stream work further. Figure 8 shows the recall of each action class

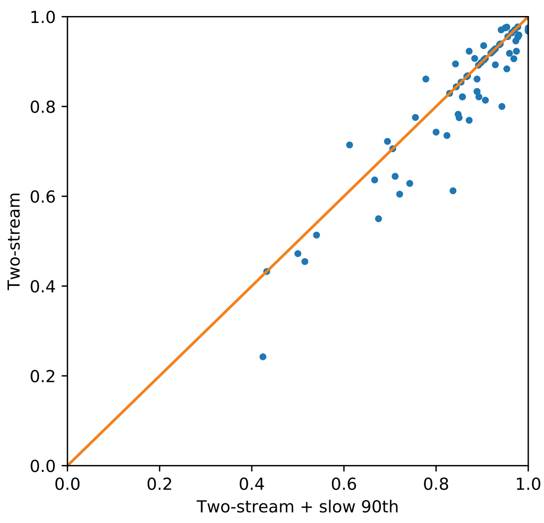
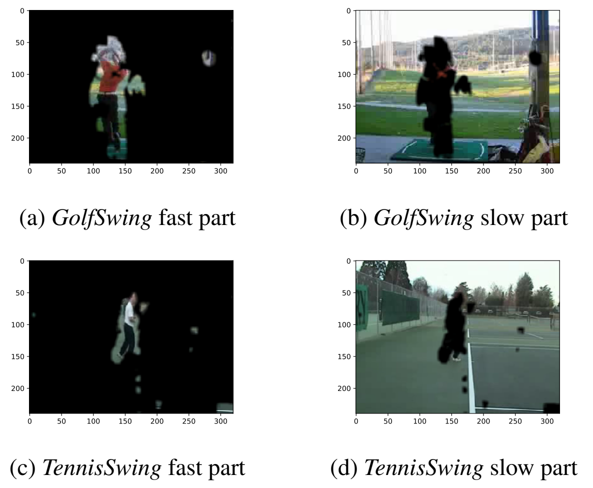


Figure 8: Recall of action classes of the combination vs the two-stream work. Further away from the orange line towards right means the combination is better. Further away from the orange line upwards means the two-stream work is better.

by the two models, where the orange line is where the two models achieve the same recall. As we can see in the plot, the combination improves the recall by a large margin for quite a few action classes, while the negative impact to other action classes is minimal. In fact, the combination improves the recall of seven action classes by more than 10%, while it only makes one action class (*i.e.*, *CricketShot*) more than 10% worse.

An example of action class that we see improvements by a lot is *TennisSwing*. The two-stream work mispredicts many clips of *TennisSwing* to be *GolfSwing*. One can notice that these actions look similar in the optical flow that there is one human player swinging a stick-like object. This similarity is also evident in Fig. 9a and Fig. 9c, where the fast parts of two actions are similar. However, the slow parts of two actions, which is a golf course (shown in Fig. 9b) and a tennis court (shown in Fig. 9d), look different. Therefore, we hypothesize that the inclusion of a slow stream mitigates the problem by putting more emphasize on the background.

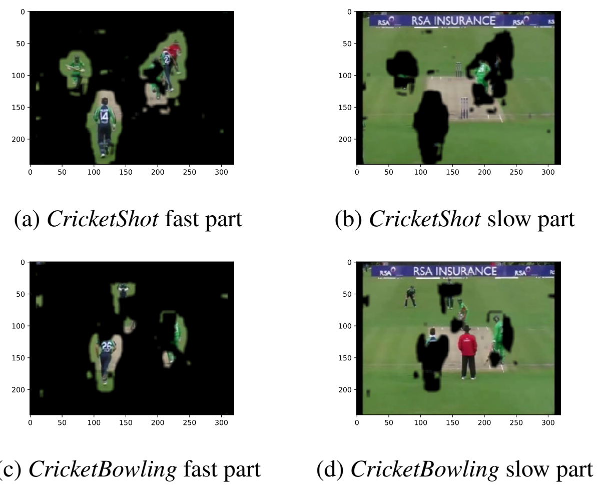
By examining the predictions, we can also see that the combination mispredicts many clips of *CricketShot* to be *CricketBowling*. As you can see from Fig. 10b and Fig. 10d, these two actions have almost identical backgrounds. We hypothesize that this is due to the combination putting too much weight to the background. Recall that the slow stream carries $\frac{1}{3}$ weight as we use an average fusion of three streams. We also hypothesize that adding a fast stream will not help in case, as both fast parts (shown in Fig. 10a and Fig. 10c) contain batsmen and bowlers. This reveals a drawback of only relying on streams of motions of different speeds, as there are cases we need the direction of the motion as well. We might be able to mitigate the problem by including streams of motions with different directions.



(a) *GolfSwing* fast part (b) *GolfSwing* slow part

(c) *TennisSwing* fast part (d) *TennisSwing* slow part

Figure 9: *GolfSwing* vs *TennisSwing*



(a) *CricketShot* fast part (b) *CricketShot* slow part

(c) *CricketBowling* fast part (d) *CricketBowling* slow part

Figure 10: *CricketShot* vs *CricketBowling*

We argue that overall this result is promising as constructing a segmented stream (in this case a slow stream) does provide extra information that cannot be found in either the spatial or the temporal stream. As many current state-of-the-art activity recognition architectures (*e.g.*, I3D [1], Temporal Segment Network [25]) base on the two-stream work, they have the concepts of using a stream of RGB frames and a stream of optical flow. We can adapt our idea in those architectures by adding extra segmented streams and hopefully will see improvements.

5. Related work

Convolutional neural networks (CNN) have been successfully applied on images. Within the research community, different architectures have been proposed to answer the question that how CNNs can be applied on videos. In this section, we discuss related work in the field of activity recognition in videos using CNNs.

5.1. LSTM on top of CNN

Compared with images, videos not only have spatial information (*i.e.*, individual frames) but also temporal information (*i.e.*, relationship between frames). A simple feed-forward structure might not be able to capture the temporal component of a video. Recurrent structures, such as an LSTM, can be added to improve the predictive performance as shown in [11, 3]. An LSTM is a more explicit way to capture temporal relationships, while in our architecture, the temporal information is implicitly presented in segmented streams.

5.2. 3D CNN

It seems like a very natural way to use 3D CNN [9, 22] on videos. By taking stacked frames as input, it should theoretically be able to learn the spatial-temporal features of the input data. However, the kernels of these spatial-temporal filters are bigger than those found in a normal 2D CNN, introducing more parameters to learn. This can be a problem for training if the dataset is not big enough.

5.3. Two-stream networks and variants

The two-stream architecture proposed by Simonyan and Zisserman [19] has been the basis of many successful architectures for activity recognition on videos. The architecture explicitly presents the spatial and temporal information of videos in spatial and temporal streams, taking a RGB frame and stacked optical flow respectively. It was able to archive high accuracy on datasets such as UCF101 and HMDB51. Compared with 3D CNN, it does not require as much training data and is more efficient to train.

Our architecture resembles the two-stream work in the overall form. However, instead of having an explicit temporal stream, we implicitly present temporal information in segmented streams, for example, fast and slow streams.

One extension [5] to the two-stream work is using 3D pooling and fusion by 3D convolution. It improves over the two-stream work, and was shown to have competitive performance against the then state-of-the-art.

Another extension to the two-stream work is the Temporal Segment Networks [25]. They essentially divide each video into a given number of segments along the time axis. Note that their use of "segments" carries a different meaning from our use. For each segment, the two-stream structure is then used. Segments are finally combined to give class scores for the entire video.

In the case of I3D [1], 2D CNNs for images are inflated into 3D ones, and weights can be bootstrapped from pre-trained 2D CNNs on ImageNet.

The above extensions all improve over the two-streaming work by changing the backbone of the architecture.

6. Conclusion

In this paper, we presented a multi-stream architecture for activity recognition in videos. For each video clip, we segment it into multiple streams based on different motion types, which are then used as inputs to CNNs. Compared with the two-stream work [19], our approach use the temporal information of videos implicitly rather than explicitly. In the experiment of using fast and slow streams, we experiment with various techniques such as using an MLP for fusion, different thresholding and greyscaling. We found that by fusing the two-stream work with a variant of slow stream, we can improve over the two-stream work, seeing about 2% improvement in the precision on the UCF101 dataset [20]. This alternative way of constructing inputs to CNNs can also be adapted to many state-of-the-art two-stream based architectures.

As for future work, there are a few small tweaks we can make. First, since our pipeline is quite general, we can explore the possibility of using other types of motion, including left/right moving, converging and diverging. As discussed in the *CricketShot* vs *CricketBowling* example in Section 4.3.5, the direction of actions might help with differentiating one action from another. Next, we can exploit the available data parallelism. Notice that in our pipeline, we have one CNN for each stream, which naturally lends itself to multi-GPU training. Finally, as noted in Section 5, there are extensions of the two-stream work, changing the underlying network structure. These changes are orthogonal to changing the input of each stream. Therefore, we can incorporate these extensions, and see whether further improvement can be obtained.

Moreover, there are some bigger changes we can make, which are shown below.

Compensating camera motion As we have discussed, many videos in the UCF101 dataset were filmed in an unconstrained environment. Many of them were likely filmed using handheld devices, and thus contains camera motion. Camera motion poses a problem when segmenting videos. That is, an object can appear to be moving due to the estimated optical flow has high magnitude resulting from the camera motion.

In Section 3.1.1, we have proposed that the dense optical flow can also be estimated by performing Gaussian interpolation on the sparse optical flow, which can be used to address the camera motion problem. Recall that when estimating the sparse optical flow, we only track points from the corner detector. Therefore, subtracting the mean flow on the sparse optical flow is less likely to make the non-moving background appear to be moving. Due to our time constraint, we only implemented this method. Future work can test this method on UCF101 and see whether better predictive precision can be achieved.

Also in Section 3.1.1, we have discussed why simply subtracting the mean flow does not work due to making the non-moving background appear to be moving. However, one can potentially using a texture detector and skip the mean flow subtraction on non-textural parts of a frame. The intuition is that a background appearing to be non-moving under camera motion is texture-less. We did not have enough time to test this idea.

There are also some commercial software that stabilizes videos, such as hyperlapse[6]. They can be used to preprocess and stabilize all videos before carrying out the optical flow calculation. We did not have enough time to test them.

Training with non-examples In our implementation, we use an MLP and perform end-to-end training to fuse CNNs for segmented streams. Since each CNN is trained separately, it might suffer from the problem that when presented with a frame, these CNNs might not agree with each other. One potential way to mitigate the problem is to add another "background" class. We can then give the MLP inputs that are from different videos or inputs at different times from the same video with a label "background". This will penalize the CNNs for giving agreeing predictions even when the input doe not make sense.

References

[1] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the Kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, July 2017. doi: 10.1109/CVPR.2017.502.

[2] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009. doi: 10.1109/CVPR.2009.5206848.

[3] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko. Long-term recurrent convolutional networks for visual recognition and description. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2625–2634, June 2015. doi: 10.1109/CVPR.2015.7298878.

[4] G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In J. Bigun and T. Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45103-7.

[5] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1933–1941, June 2016. doi: 10.1109/CVPR.2016.213.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. doi: 10.1109/CVPR.2016.90.

[7] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185 – 203, 1981. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(81)90024-2. URL `http://www.sciencedirect.com/science/article/pii/0004370281900242`.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/ioffe15.html`.

[9] S. Ji, W. Xu, M. Yang, and K. Yu. 3D convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, Jan 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.59.

[10] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, June 1994. doi: 10.1109/CVPR.1994.323794.

[11] Joe Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4694–4702, June 2015. doi: 10.1109/CVPR.2015.7299101.

[12] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Com-*

[6]`https://www.microsoft.com/en-us/research/product/computational-photography-applications/microsoft-hyperlapse-pro/`

puter Vision and Pattern Recognition, pages 1725–1732, June 2014. doi: 10.1109/CVPR.2014.223.

[13] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman, and A. Zisserman. The Kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017. URL `http://arxiv.org/abs/1705.06950`.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL `http://dl.acm.org/citation.cfm?id=2999134.2999257`.

[15] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: A large video database for human motion recognition. In *2011 International Conference on Computer Vision*, pages 2556–2563, Nov 2011. doi: 10.1109/ICCV.2011.6126543.

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, Dec. 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541. URL `http://dx.doi.org/10.1162/neco.1989.1.4.541`.

[17] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1623264.1623280`.

[18] J. Sánchez Pérez, E. Meinhardt-Llopis, and G. Facciolo. TV-L1 optical flow estimation. *Image Processing On Line*, 3: 137–150, 2013. doi: 10.5201/ipol.2013.26.

[19] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pages 568–576, Cambridge, MA, USA, 2014. MIT Press. URL `http://dl.acm.org/citation.cfm?id=2968826.2968890`.

[20] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012. URL `http://arxiv.org/abs/1212.0402`.

[21] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `http://proceedings.mlr.press/v28/sutskever13.html`.

[22] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *Computer Vision – ECCV 2010*, pages 140–153, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15567-3.

[23] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3D convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4489–4497, Dec 2015. doi: 10.1109/ICCV.2015.510.

[24] H. Wang and C. Schmid. Action recognition with improved trajectories. In *2013 IEEE International Conference on Computer Vision*, pages 3551–3558, Dec 2013. doi: 10.1109/ICCV.2013.441.

[25] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 20–36, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46484-8.

[26] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime TV-L1 optical flow. In F. A. Hamprecht, C. Schnörr, and B. Jähne, editors, *Pattern Recognition*, pages 214–223, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-74936-3.