Compressed Video Action Recognition

Chao-Yuan Wu[1,5*] cywu@cs.utexas.edu
Manzil Zaheer[2,5*] manzil@cmu.edu
Hexiang Hu[3,5*] hexiangh@usc.edu
R. Manmatha[4] manmatha@a9.com
Alexander J. Smola[5] smola@amazon.com
Philipp Krähenbühl[1] philkr@cs.utexas.edu

[1]The University of Texas at Austin, [2]Carnegie Mellon University,
[3]University of Southern California, [4]A9, [5]Amazon

Abstract

Training robust deep video representations has proven to be much more challenging than learning deep image representations. This is in part due to the enormous size of raw video streams and the high temporal redundancy; the true and interesting signal is often drowned in too much irrelevant data. Motivated by that the superfluous information can be reduced by up to two orders of magnitude by video compression (using H.264, HEVC, etc.), we propose to train a deep network directly on the compressed video.

This representation has a higher information density, and we found the training to be easier. In addition, the signals in a compressed video provide free, albeit noisy, motion information. We propose novel techniques to use them effectively. Our approach is about 4.6 times faster than Res3D and 2.7 times faster than ResNet-152. On the task of action recognition, our approach outperforms all the other methods on the UCF-101, HMDB-51, and Charades dataset.

1. Introduction

Video commands the lion's share of internet traffic at 70% and rising [24]. Most cell phone cameras now capture high resolution videos in addition to images. Many real-world data sources are video based, ranging from inventory systems at warehouses to self-driving cars or autonomous drones. Video is also arguably the next frontier in computer vision, as it captures a wealth of information still images cannot convey. Videos carry more emotion [32], allow us to predict the future to a certain extent [23], provide temporal context and give us better spatial awareness [26]. Unfortunately, very little of this information is currently exploited.

State-of-the-art deep learning models for video analysis are quite basic. Most of them naïvely use convolutional neural networks (CNNs) designed for images to parse a video *frame by frame*. They often demonstrate results no better than hand-crafted techniques [16, 42]. So, why did deep learning not yet make as transformative of an impact on video tasks, such as action recognition, as it did on images?

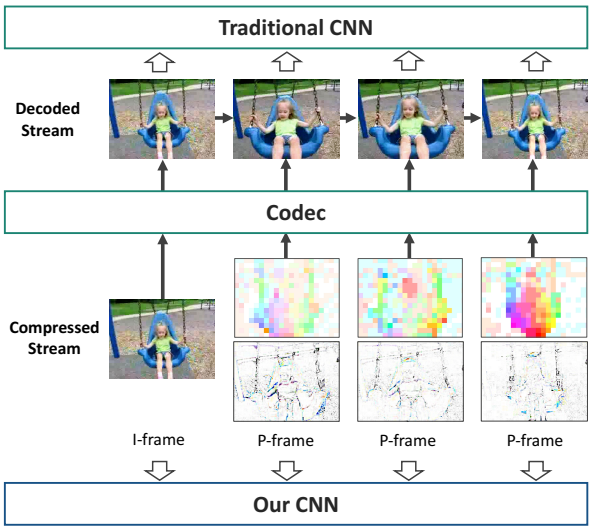


Figure 1: Traditional architectures first decode the video and then feed it into a network. We propose to use the compressed video directly.

We argue that the reason is two-fold. First, videos have a very low information density, as 1h of 720p video can be compressed from 222GB raw to 1GB. In other words, videos are filled with boring and repeating patterns, drowning the 'true' and interesting signal. The redundancy makes it harder for CNNs to extract meaningful information, and makes the training much slower. Second, with only RGB images, learning temporal structure is difficult. A vast body of literature attempts to process videos as RGB image sequences, either with 2D CNNs, 3D CNNs, or recurrent neural networks (RNNs), but has yielded limited success [16, 40]. Using precomputed optical flow almost always boosts the performance [2].

*Part of this work performed while interning at Amazon.

To address these issues, we exploit the compressed representation developed for storage and transmission of videos rather than operating on the RGB frames (Figure 1). These compression techniques (like MPEG-4, H.264 etc.) leverage that successive frames are usually similar. They retain only a few frames completely and reconstruct other frames based on offsets, called motion vectors and residual error, from the complete images. Our model consists of multiple CNNs that directly operate on the motion vectors, residuals, in addition to a small number of complete images.

Why is this better? First, video compression removes up to two orders of magnitude of superfluous information, making interesting signals prominent. Second, the motion vectors in video compression provide us the *motion* information that lone RGB images do not have. Furthermore, the motion signals already exclude spatial variations, e.g. two people performing the same action in different clothings or in different lighting conditions exhibit the same *motion* signals. This improves generalization, and the lowered variance further simplifies training. Third, with compressed video, we account for correlation in video frames, i.e. spatial view plus some small changes over time, instead of i.i.d. images. Constraining data in this structure helps us tackling the curse of dimensionality. Last but not least, our method is also much more efficient as we only look at the true signals instead of repeatedly processing near-duplicates. Efficiency is also gained by avoiding to decompress the video, because video is usually stored or transmitted in the compressed version, and access to the motion vectors and residuals are free.

On action recognition datasets UCF-101 [34], HMDB-51 [18], and Charades [32], our approach significantly outperforms all other methods that train on traditional RGB images. Our approach is simple and fast, without using RNNs, complicated fusion or 3D convolutions. It is 4.6 times faster than state-of-the-art 3D CNN model Res3D [40], and 2.7 times faster than ResNet-152 [12]. When combined with scores from a standard temporal stream network, our model outperforms state-of-the-art methods on all these datasets.

2. Background

In this section we provide a brief overview about video action recognition and video compression.

2.1. Action Recognition

Traditionally, for video action recognition, the community utilized hand-crafted features, such as Histogram of Oriented Gradients (HOG) [3] or Histogram of Optical Flow (HOF) [19], both sparsely [19] and densely [43] sampled. While early methods consider independent interest points across frames, smarter aggregation based on dense trajectories have been used [25, 41, 42]. Some of these traditional methods are competitive even today, like iDT which corrects for camera motion [42].

In the past few years, deep learning has brought significant improvements to video understanding [6, 16]. However, the improvements mainly stem from improvements in deep image representations. Modeling of temporal structure is still relatively simple — most algorithms subsample a few frames and perform average pooling to make final predictions [33, 44]. RNNs [6, 48], temporal CNNs [21], or other feature aggregation techniques [11, 44] on top of CNN features have also been explored. However, while introducing new computation overhead, these methods do not necessarily outperform simple average pooling [44]. Some works explore 3D CNNs to model the temporal structure [39, 40]. Nonetheless, it results in an explosion of parameters and computation time and only marginally improves the performance [40].

More importantly, evidence suggests that these methods are not sufficient to capture all temporal structures — using of pre-computed optical flow almost always boosts the performance [2, 9, 33, 44]. This emphasizes the importance of using the right input representation and the inadequacy of RGB frames. Finally, note that all of these methods require raw video frame-by-frame and cannot exploit the fact that video is stored in some compressed format.

2.2. Video Compression

The need for efficient video storage and transmission has led to highly efficient video compression algorithms, such as MPEG-4, H.264, and HEVC, some of which date back to 1990s [20]. Most video compression algorithms leverage the fact that successive frames are usually very similar. We can efficiently store one frame by reusing contents from another frame and only store the difference.

Most modern codecs split a video into *I-frames* (intra-coded frames), *P-frames* (predictive frames) and zero or more *B-frames* (bi-directional frames). I-frames are regular images and compressed as such. P-frames reference the previous frames and encode only the 'change'. A part of the change – termed motion vectors – is represented as the movements of block of pixels from the source frame to the target frame at time t, which we denote by $\mathcal{T}^{(t)}$. Even after this compensation for block movement, there can be difference between the original image and the predicted image at time t. We denote this residual difference by $\Delta^{(t)}$. Putting it together, a P-frame at time t only comprises of motion vectors $\mathcal{T}^{(t)}$ and a residual $\Delta^{(t)}$. This gives the recurrence relation for reconstructing P-frames as

$$I_i^{(t)} = I_{i-\mathcal{T}_i^{(t)}}^{(t-1)} + \Delta_i^{(t)}, \tag{1}$$

for all pixel i, where $I^{(t)}$ denotes the RGB image at time t. The motion vectors and the residuals are then passed through discrete cosine transform (DCT) and entropy-encoded.

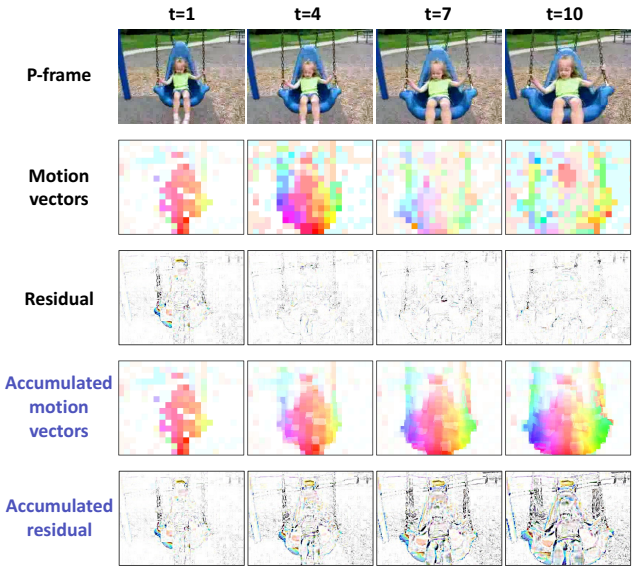


Figure 2: Original motion vectors and residuals describe only the change between two frames. Usually the signal to noise ratio is very low and hard to model. The accumulated motion vectors and residuals consider longer term difference and show clearer patterns. Assume I-frame is at $t = 0$. Motion vectors are plotted in HSV space, where the H channel encodes the direction of motion, and the S channel shows the amplitude. For residuals we plot the absolute values in RGB space. Best viewed in color.

A B-frame may be viewed as a special P-frame, where motion vectors are computed bi-directionally and may reference a future frame as long as there are no circles in referencing. Both B- and P- frames capture only what changes in the video, and are easier to compress owing to smaller dynamic range [28]. See Figure 2 for a visualization of the motion estimates and the residuals. Modeling arbitrary decoding order is beyond the scope of this paper. We focus on videos encoded using only backward references, namely I- and P- frames.

Features from Compressed Data. Some prior works have utilized signals from compressed video for detection or recognition, but only as a non-deep feature [15, 36, 38, 47]. To the best of our knowledge, this is the first work that considers training deep networks on compressed videos. MV-CNN apply distillation to transfer knowledge from an optical flow network to a motion vector network [50]. However, unlike our approach, it does not consider the general setting of representation learning on a compressed video; it still needs the entire decompressed video as RGB stream, and it requires optical flow as an additional supervision.

Equipped with this background, next we will explore how to utilize the compressed representation, devoid of redundant information, for action recognition.

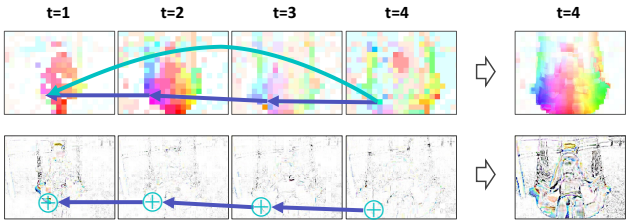


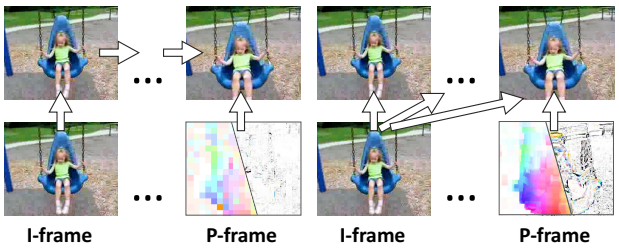
Figure 3: We trace all motion vectors back to the reference I-frame and accumulate the residual. Now each P-frame depends only on the I-frame but not other P-frames.

3. Modeling Compressed Representations

Our goal is to design a computer vision system for action recognition that operates directly on the stored compressed video. The compression is solely designed to optimize the size of the encoding, thus the resulting representation has very different statistical and structural properties than the images in a raw video. It is not clear if the successful deep learning techniques can be adapted to compressed representations in a straightforward manner. So we ask how to feed a compressed video into a computer vision system, specifically a deep network?

Feeding I-frames into a deep network is straightforward since they are just images. How about P-frames? From Figure 2 we can see that motion vectors, though noisy, roughly resemble optical flows. As modeling optical flows with CNNs has been proven effective, it is tempting to do the same for motion vectors. The third row of Figure 2 visualizes the residuals. We can see that they roughly give us a motion boundary in addition to a change of appearance, such as the change of lighting conditions. Again, CNNs are well-suited for such patterns. The outputs of corresponding CNNs from the image, motion vectors, and residual will have different properties. To combine them, we tried various fusion strategies, including mean pooling, maximum pooling, concatenation, convolution pooling, and bilinear pooling, on both middle layers and the final layer, but with limited success.

Digging deeper, one can argue that the motion vectors and residuals alone do not contain the full information of a P-frame — a P-frame depends on the reference frame, which again might be a P-frame. This chain continues all the way back to a preceding I-frame. Treating each P-frame as an independent observation clearly violates this dependency. A simple strategy to address this is to reuse features from the reference frame, and only *update* the features given the new information. This recurrent definition screams for RNNs to aggregate features along the chain. However, preliminary experiments suggest the elaborate modeling effort in vain (see supplementary material for details). The difficulty arises from the long chain of dependency of the P-frames. To mitigate this issue, we devise a novel yet simple back-tracing technique that decouples individual P-frames.



(a) Original dependency. (b) New dependency.

Figure 4: We decouple the dependencies between P-frames so that they can be processed in parallel.

Decoupled Model. To break the dependency between consecutive P-frames, we trace all motion vectors back to the reference I-frame and accumulate the residual on the way. In this way, each P-frame depends only on the I-frame but not other P-frames.

Figure 3 illustrates the back-tracing technique. Given a pixel at location i in frame t, let $\mu_{\mathcal{T}^{(t)}}(i) := i - \mathcal{T}_i^{(t)}$ be the referenced location in the previous frame. The location traced back to frame $k < t$ is given by

$$\mathcal{J}_i^{(t,k)} := \mu_{\mathcal{T}^{(k+1)}} \circ \cdots \circ \mu_{\mathcal{T}^{(t)}}(i). \tag{2}$$

Then the accumulated motion vectors $\mathcal{D}^{(t)} \in \mathbb{R}^{H\times W\times 2}$ and the accumulated residuals $\mathcal{R}^{(t)} \in \mathbb{R}^{H\times W\times 3}$ at frame t are

$$\begin{aligned}\mathcal{D}_i^{(t)} &:= i - \mathcal{J}_i^{(t,k)}, \text{ and}\\ \mathcal{R}_i^{(t)} &:= \Delta_{\mathcal{J}_i^{(t,k+1)}}^{(k+1)} + \cdots + \Delta_{\mathcal{J}_i^{(t,t-1)}}^{(t-1)} + \Delta_i^{(t)},\end{aligned}$$

respectively. This can be efficiently calculated in linear time through a simple feed forward algorithm, accumulating motion and residuals as we decode the video. Each P-frame now has a different dependency

$$I_i^{(t)} = I_{i-\mathcal{D}_i^{(t)}}^{(0)} + \mathcal{R}_i^{(t)}, \quad t = 1, 2, \ldots,$$

as shown in Figure 4b. Here P-frames depend only on the I-frame and can be processed in parallel.

A nice side effect of the back-tracing is robustness. The accumulated signals contain longer-term information, which is more robust to noise or camera motion. Figure 2 shows the accumulated motion vectors and residuals respectively. They exhibit clearer and smoother patterns than the original ones.

Proposed Network. Figure 5 shows the graphical illustration of the proposed model. The input of our model is an I-frame, followed by T P-frames, i.e. $\left(I^0, \mathcal{D}^{(1)}, \mathcal{R}^{(1)}, \ldots, \mathcal{D}^{(T)}, \mathcal{R}^{(T)}\right)$. For notational simplicity we set $t = 0$ for the I-frame. Each input source is modeled

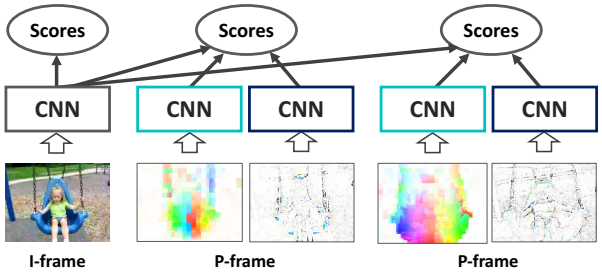


Figure 5: Decoupled model. All networks can be trained independently. Models are shared across P-frames.

by a CNN, i.e.

$$\begin{aligned}x_{\text{RGB}}^{(0)} &:= \phi_{\text{RGB}}\left(I^{(0)}\right)\\ x_{\text{motion}}^{(t)} &:= \phi_{\text{motion}}\left(\mathcal{D}^{(t)}\right)\\ x_{\text{residual}}^{(t)} &:= \phi_{\text{residual}}\left(\mathcal{R}^{(t)}\right)\end{aligned}$$

While I-frame features $x_{\text{RGB}}^{(0)}$ are used as is, P-frame features $x_{\text{motion}}^{(t)}$ and $x_{\text{residual}}^{(t)}$ need to incorporate the information from $x_{\text{RGB}}^{(0)}$. There are several reasonable candidates for such a fusion, e.g. maximum, multiplicative or convolutional pooling. We also experiment with transforming RGB features according to the motion vector. Interestingly, we found a simple summing of scores to work best (see supplementary material for details). This gives us a model that is easy to train and flexible for inference.

Implementation. Note that most of the information is stored in I-frames, and we only need to learn the *update* for P-frames. We thus focus most of the computation on I-frames, and use a much smaller and simpler model to capture the updates in P-frames. This yields significant saving in terms of computation, since in modern codecs most frames are P-frames.

Specifically, we use ResNet-152 (pre-activation) to model I-frames, and ResNet-18 (pre-activation) to model the motion vectors and residuals [13]. This offers a good trade-off between speed and accuracy. For video-level tasks, we use Temporal Segments [44] to capture long term dependency, i.e. feature at each step is the average of features across $k = 3$ segments during training.

4. Experiments

We now validate for action recognition that (i) compressed video is a better representation (Section 4.1), leading to (ii) good accuracy (Section 4.3) and (iii) high speed (Section 4.2). However, note that the principle of the proposed method can be applied effortlessly to other tasks like video classification [1], object detection [29], or action localization [32]. We pick action recognition due to its wide range of applications and strong baselines.

Datasets and Protocol. We evaluate our method Compressed Video Action Recognition (CoViAR) on three action recognition datasets, UCF-101 [34], HMDB-51 [18], and Charades [32]. UCF-101 and HMDB-51 contain short ($<$ 10-second) trimmed videos, each of which is annotated with one action label. Charades contains longer ($\sim$ 30-second) untrimmed videos. Each video is annotated with one or more action labels and their intervals (start time, end time). UCF-101 contains 13,320 videos from 101 action categories. HMDB-51 contains 6,766 videos from 51 action categories. Each dataset has 3 (training, testing)-splits. We report the average performance of the 3 testing splits unless otherwise stated. The Charades dataset contains 9,848 videos split into 7,985 training and 1,863 test videos. It contains 157 action classes.

During testing we uniformly sample 25 frames, each with flips plus 5 crops, and then average the scores for final prediction. On UCF-101 and HMDB-51 we use temporal segments, and perform the averaging before softmax following TSN [44]. On Charades we use mean average precision (mAP) and weighted average precision (wAP) to evaluate the performance, following previous work [31].

Training Details. Following TSN [44], we resize UCF-101 and HMDB-51 videos to 340×256. As Charades contains both portrait and landscape videos, we resize them to 256×256. Our models are pre-trained on the ILSVRC 2012-CLS dataset [4], and fine-tuned using Adam [17] with a batch size of 40. Learning rate starts from 0.001 for UCF-101/HMDB-51 and 0.03 for Charades. It is divided by 10 when the accuracy plateaus. Pre-trained layers use a $100\times$ smaller learning rate. We apply color jittering and random cropping to 224×224 for data augmentation following TSN [44]. Where available, we select the hyper-parameters on splits other than the tested one. We use MPEG-4 encoded videos, which have on average 11 P-frames for every I-frame. Optical flow models use TV-L1 flows [49].

4.1. Ablation Study

Here we study the benefits of using compressed representations over RGB images. We focus on UCF-101 and HMDB-51, as they are two of the most well-studied action recognition datasets. Table 1 presents a detailed analysis. On both datasets, training on compressed videos significantly outperforms training on RGB frames. In particular, it provides 5.8% and 2.7% absolute improvement on HMDB-51 and UCF-101 respectively.

Quite surprisingly, while residuals contribute to a very small amount of data, it alone achieves good accuracy. Motion vectors alone perform not as well, as they do not contain spatial details. However, they offer information orthogonal to what still images provide. When added to other streams, it significantly boosts the performance. Note that we use only I-frames as full images, which is a small subset of all frames, yet CoViAR achieves good performance.

	I	M	R	I+M	I+R	I+M+R (gain)
UCF-101						
Split 1	88.4	63.9	79.9	<u>90.4</u>	90.0	**90.8** (+2.4)
Split 2	87.4	64.6	80.8	<u>89.9</u>	89.6	**90.5** (+3.1)
Split 3	87.3	66.6	82.1	<u>89.6</u>	89.4	**90.0** (+2.7)
Average	87.7	65.0	80.9	<u>89.9</u>	89.7	**90.4** (+2.7)
HMDB-51						
Split 1	54.1	37.8	44.6	<u>60.3</u>	55.9	**60.4** (+6.3)
Split 2	51.9	38.7	43.1	<u>57.9</u>	54.2	**58.2** (+6.3)
Split 3	54.1	39.7	44.4	<u>58.5</u>	55.6	**58.7** (+4.6)
Average	53.3	38.8	44.1	<u>58.9</u>	55.2	**59.1** (+5.8)

Table 1: Action recognition accuracy on UFC-101 [34] and HMDB-51 [18]. Here we compare training with different sources of information. "+" denotes score fusion of models. I: I-frame RGB image. M: motion vectors. R: residuals. The bold numbers indicate the best and the underlined numbers indicate the second best performance.

	M	R	I+M	I+R	I+M+R
Original	58.3	79.0	90.0	89.8	90.4
Accumulated	**63.9**	**79.9**	**90.4**	**90.0**	**90.8**

Table 2: Action recognition accuracy on UFC-101 [34] (Split 1). The two rows show the performance of the models trained using the original motion vectors/residuals and the models using the accumulated ones respectively. I: I-frame RGB image. M: motion vectors. R: residuals.

Accumulated Motion Vectors and Residuals. Our back-tracing technique not only simplifies the dependency but also results in clearer patterns to model. This improves the performance, as shown in Table 2. On the first split of UCF-101, our accumulation technique provides 5.6% improvement on the motion vector stream network and on the full model, 0.4% improvement (4.2% error reduction). Performance of the residual stream also improves by 0.9% (4.3% error reduction).

Visualizations. In Figure 7, we qualitatively study the RGB and compressed representations of two videos of the same action in t-SNE [22] space. We can see that in RGB space the two videos are clearly separated, and in motion vector and residual space they overlap. This suggests that a RGB-image based model needs to learn the two patterns separately, while a compressed-video based model sees a shared representation for videos of the same action, making training and generalization easier.

In addition, note that the two ways of the RGB trajectories overlap, showing that RGB images cannot distinguish between the up-moving and down-moving motion. On the other hand, compressed signals preserve motion. The trajectories thus form circles instead of going back and forth on the same path.

	GFLOPs	Accuracy (%) UCF-101	HMDB-51
ResNet-50 [8]	**3.8**	82.3	48.9
ResNet-152 [8]	11.3	83.4	46.7
C3D [39]	38.5	82.3	51.6
Res3D [40]	19.3	85.8	54.9
CoViAR	4.2	**90.4**	**59.1**

Table 3: Network computation complexity and accuracy of each method. Our method is 4.6x more efficient than state-of-the-art 3D CNN, while being much more accurate.

	Preprocess	CNN (sequential)	CNN (concurrent)
Two-stream			
BN-Inception	75.0	1.6	0.9
ResNet-152	75.0	7.5	4.0
CoViAR	**2.87/0.46**	**1.3**	**0.3**

Table 4: Speed (ms) per frame. CoViAR is fast in both preprocessing and CNN computation. Its preprocessing speed is presented for both single-thread / multi-thread settings.

4.2. Speed and Efficiency

Our method is efficient because the computation on the I-frame is shared across multiple frames, and the computation on P-frames is cheaper. Table 3 compares the CNN computational cost of our method with state-of-the-art 2D and 3D CNN architectures. Since for our model the P- and I-frame computational costs are different, we report the average GFLOPs over all frames. As shown in the table, CoViAR is 2.7 times faster than ResNet-152 [12] and is 4.6 times more than Res3D [40], while being significantly more accurate.

A more detailed speed analysis is presented in Table 4. The preprocessing time of the two-stream methods, i.e. optical flow computation, is measured on a Tesla P100 GPU with an implementation of the TV-L1 flow algorithm from OpenCV. Our preprocessing, i.e. the calculation of the accumulated motion vectors and residuals, is measured on Intel E5-2698 v4 CPUs. CNN time is measured on the same P100 GPU. We can see that the optical flow computation is the bottleneck for two-stream networks, even with low-resolution 256×340 videos. Our preprocessing is much faster despite our CPU-only implementation.

For CNN time, we consider both settings where (i) we can forward multiple CNNs at the same time, and (ii) we do it sequentially. For both settings, our method is significantly faster than traditional methods. Overall, our method can be up to 100 times faster than traditional methods with multi-thread preprocessing, running at 1,300 frames per second. Figure 6 summarizes the results. CoViAR achieves the best efficiency and good accuracy, while requiring a far lesser amount of data.

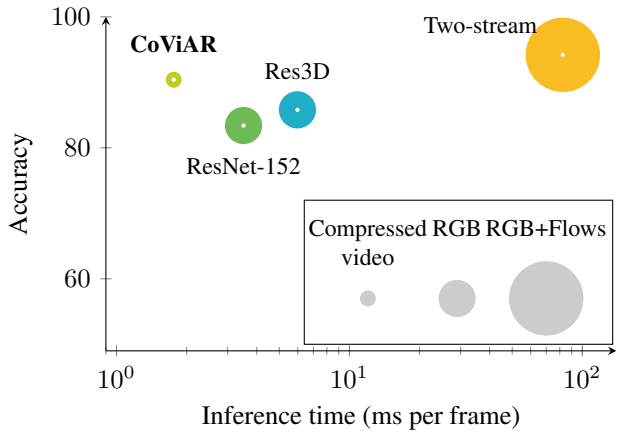


Figure 6: Speed and accuracy on UCF-101 [34], compared to a two-stream network (TSN) [33, 44], Res3D [40], and ResNet-152 [12] trained on RGB frames. Node size denotes the input data size. Training on compressed videos is both accurate and efficient.

	UCF-101			HMDB-51		
	CoViAR	Flow	CoViAR +flow	CoViAR	Flow	CoViAR +flow
Split 1	90.8	87.7	**94.0**	60.4	61.8	**71.5**
Split 2	90.5	90.2	**95.4**	58.2	63.7	**69.4**
Split 3	90.0	89.1	**95.2**	58.7	64.2	**69.7**
Average	90.4	89.0	**94.9**	59.1	63.2	**70.2**

Table 5: Action recognition accuracy on UFC-101 [34] and HMDB-51 [18]. Combining our model with a temporal-stream network achieves state-of-the-art performance.

4.3. Accuracy

We now compare the accuracy of CoViAR with state-of-the-art models in Table 6. For fair comparison, here we focus on models using the same pre-training dataset, ILSVRC 2012-CLS [4]. While pre-training using Kinetics yields better performance [2], since it is larger and more similar to the datasets used in this paper, those results are not directly comparable.

From the upper part of the table, we can see that our model significantly outperforms traditional RGB-image based methods. C3D [39], Res3D [40], P3D ResNet [27], and I3D [2] consider 3D convolution to learn temporal structures. Karpathy *et al.* [16] and TLE [5] consider more complicated fusions and pooling. MV-CNN [50] apply distillation to transfer knowledge from an optical-flow-based model. Our method uses much faster 2D CNNs plus simple late fusion without additional supervision, and still significantly outperforms these methods.

[†]Despite our best efforts, we were not able to reproduce the performance reported in the original paper. Here we report the performance based on our implementation. For fair comparison, we use the same data augmentation and architecture as ours. Training follows the 2-stage pro-

	UCF-101	HMDB-51
Without optical flow		
Karpathy *et al.* [16]	65.4	-
ResNet-50 [12] (from ST-Mult [8])	82.3	48.9
ResNet-152 [12] (from ST-Mult [8])	83.4	46.7
C3D [39]	82.3	51.6
Res3D [40]	85.8	54.9
TSN (RGB-only) [44]*	85.7	-
TLE (RGB-only) [5]†	87.9	54.2
I3D (RGB-only) [2]*	84.5	49.8
MV-CNN [50]	86.4	-
P3D ResNet [27]	88.6	-
Attentional Pooling [10]	-	52.2
CoViAR	**90.4**	**59.1**
With optical flow		
iDT+FV [42]	-	57.2
Two-Stream [33]	88.0	59.4
Two-Stream fusion [9]	92.5	65.4
LRCN [6]	82.7	
Composite LSTM Model [35]	84.3	44.0
ActionVLAD [11]	92.7	66.9
ST-ResNet [7]	93.4	66.4
ST-Mult [8]	94.2	68.9
I3D [2]*	93.4	66.4
TLE [5]†	93.8	68.8
L^2STM [37]	93.6	66.2
ShuttleNet [30]	94.4	66.6
STPN [45]	**94.6**	68.9
TSN [44]	94.2	69.4
CoViAR + optical flow	**94.9**	**70.2**

Table 6: Accuracy on UCF-101 [34] and HMDB-51 [18]. The upper lists real-time methods that do not require optical flow; the lower part lists methods using optical flow. Our method outperforms all baselines in both settings. Asterisk indicates results evaluated only on split 1 of the datasets (purely for reference).

Two-stream Network. Most state-of-the-art models use the two-stream framework, i.e. one stream trained on RGB frames and the other on optical flows. It is natural to ask: What if we replace the RGB stream by our compressed stream? Here we train a temporal-stream network using 7 segments with BN-Inception [14], and combine it with our model by late fusion. Despite its simplicity, this achieves very good performance as shown in Table 5.

The lower part of Table 6 compares our method with state-of-the-art models using optical flow. CoViAR outperforms all of them. LRCN [6], Composite LSTM Model [35], and L^2STM [37] use RNNs to model temporal dynamics. ActionVLAD [11] and TLE [5] apply more complicated feature aggregation. iDT+FT [42] is based on hand-engineered features. Again, our method simply trains 2D CNNs separately without any complicated fusion or RNN and still outperforms these models.

cedure described in the original paper. We reached out to the authors, but they were unable to share their implementation.

	mAP (%)	wAP (%)
Without optical flow		
ActionVLAD [11] (RGB only)	17.6	25.1
Sigurdsson *et al.* [31] (RGB only)	18.3	-
CoViAR	**21.9**	**29.4**
With optical flow		
Two-stream [33] (from [32])	14.3	-
Two-stream [33] + iDT [42] (from [32])	18.6	-
ActionVLAD [11] (RGB only) + iDT	21.0	29.9
Sigurdsson *et al.* [31]	22.4	-
CoViAR + optical flow	**24.1**	**32.3**

Table 7: Accuracy on Charades [32]. Without using additional annotations as Sigurdsson *et al.* [31], our method achieves the best performance.

Finally we evaluate our method on the Charades dataset (Table 7). As Charades consists of annotations at frame-level, we train our network to predict the labels of each frame. At test time we average the scores of the sampled frames as the final prediction. Our method again outperforms other models trained on RGB images. Note that Sigurdsson *et al.* use additional annotations including objects, scenes, and intentions to train a conditional random field (CRF) model [31]. Our model requires only action labels. When using optical flow, CoViAR outperforms all other state-of-the-art methods. The effectiveness on Charades demonstrates the effectiveness of CoViAR for both video-level and frame-level predictions.

5. Conclusion

In this paper, we propose to train deep networks directly on compressed videos. This is motivated by the *practical* observation that either video compression is essentially free on all modern cameras, due to hardware-accelerated video codecs or that the video is directly *available* in its compressed form. In other words, decompressing the video is actually an inconvenience.

We demonstrate that, quite surprisingly, this is not a drawback but rather a virtue. In particular, video compression reduces irrelevant information from the data, thus rendering it more robust. After all, compression is not meant to affect the content that humans consider pertinent. Secondly, the increased relevance and reduced dimensionality makes computation much more effective (we are able to use much simpler networks for motion vectors and residuals). Finally, the accuracy of the model actually *improves* when using compressed data, yielding new state of the art.

In short, our method is both faster and more accurate, while being simpler to implement than previous works.

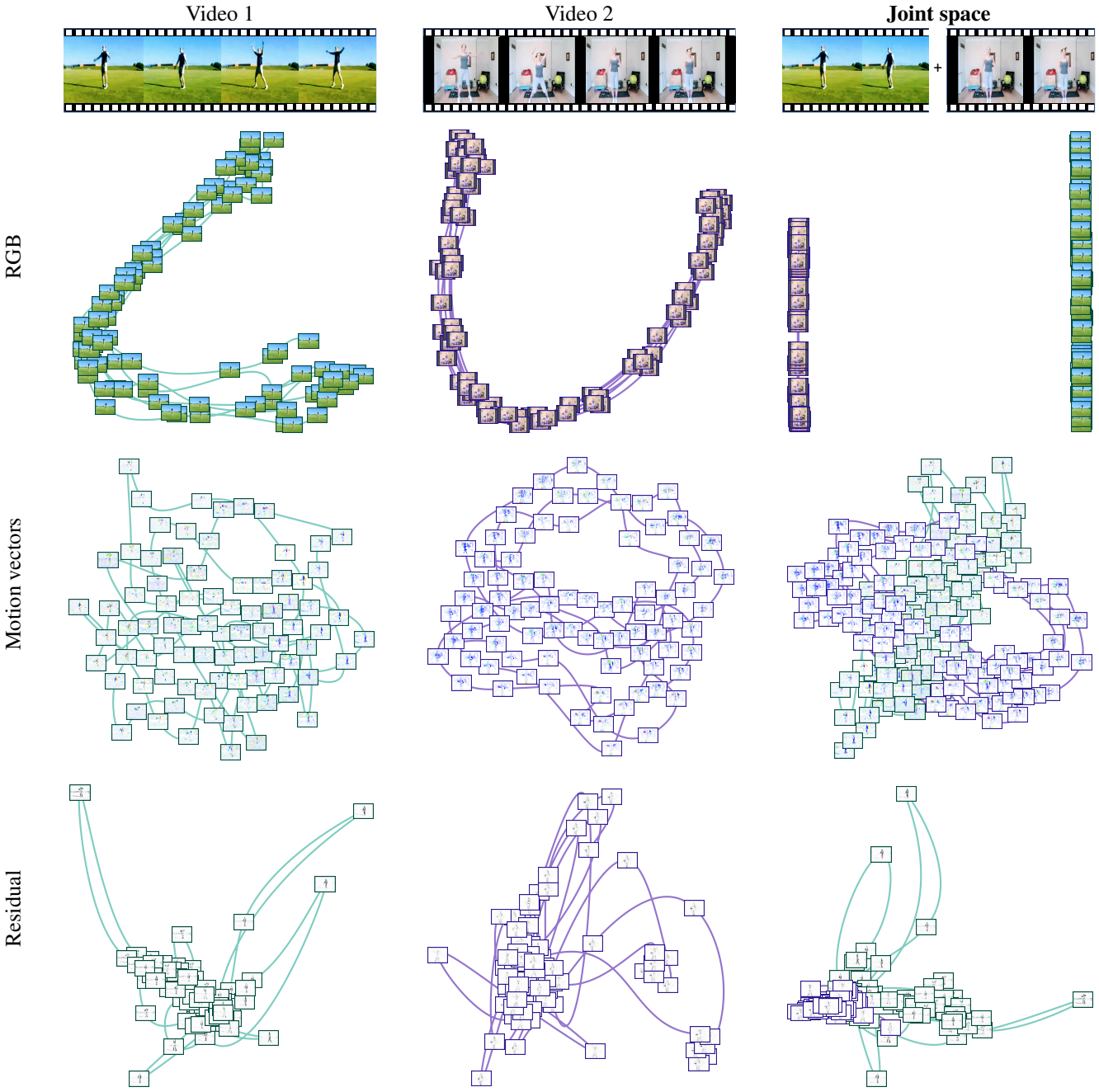


Figure 7: Two videos of "Jumping Jack" from UCF-101 in their RGB, motion vector, and residual representations plotted in t-SNE [22] space. The curves show video trajectories. While in the RGB space the two videos are clearly separated, in the motion vector and residual space they overlap. This suggests that with compressed signals, videos of the same action can share statistical strength better. Also note that the RGB images contain no motion information, and thus the two ways of the trajectories overlap. This is in contrast to the *circular* patterns in the trajectories of motion vectors. Best viewed on screen.

Acknowledgment

We would like to thank Ashish Bora for helpful discussions. This work was supported in part by Berkeley DeepDrive and an equipment grant from Nvidia.

References

[1] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan. YouTube-8M: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*, 2016. 4

[2] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, 2017. 1, 2, 6, 7

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. 2

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 5, 6

[5] A. Diba, V. Sharma, and L. Van Gool. Deep temporal linear encoding networks. *CVPR*, 2017. 6, 7

[6] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015. 2, 7

[7] C. Feichtenhofer, A. Pinz, and R. Wildes. Spatiotemporal residual networks for video action recognition. In *NIPS*, 2016. 7

[8] C. Feichtenhofer, A. Pinz, and R. P. Wildes. Spatiotemporal multiplier networks for video action recognition. In *CVPR*, 2017. 6, 7, 11

[9] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *CVPR*, 2016. 2, 7

[10] R. Girdhar and D. Ramanan. Attentional pooling for action recognition. In *NIPS*, 2017. 7

[11] R. Girdhar, D. Ramanan, A. Gupta, J. Sivic, and B. Russell. Actionvlad: Learning spatio-temporal aggregation for action classification. In *CVPR*, 2017. 2, 7

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2, 6, 7

[13] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 4

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 7

[15] V. Kantorov and I. Laptev. Efficient feature extraction, encoding and classification for action recognition. In *CVPR*, 2014. 3

[16] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014. 1, 2, 6, 7

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[18] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *ICCV*, 2011. 2, 5, 6, 7

[19] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *CVPR*, 2008. 2

[20] D. Le Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 1991. 2

[21] C.-Y. Ma, M.-H. Chen, Z. Kira, and G. AlRegib. TS-LSTM and temporal-inception: Exploiting spatiotemporal dynamics for activity recognition. *arXiv preprint arXiv:1703.10667*, 2017. 2

[22] L. v. d. Maaten and G. Hinton. Visualizing data using t-SNE. *JMLR*, 2008. 5, 8

[23] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *ICLR*, 2016. 1

[24] C. V. networking Index. Forecast and methodology, 2016-2021, white paper. *San Jose, CA, USA*, 2016. 1

[25] X. Peng, C. Zou, Y. Qiao, and Q. Peng. Action recognition with stacked fisher vectors. In *ECCV*, 2014. 2

[26] M. Pollefeys, D. Nistér, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, et al. Detailed real-time urban 3d reconstruction from video. *IJCV*, 2008. 1

[27] Z. Qiu, T. Yao, and T. Mei. Learning spatio-temporal representation with pseudo-3d residual networks. In *ICCV*, 2017. 6, 7

[28] I. E. Richardson. *Video codec design: developing image and video compression systems*. John Wiley & Sons, 2002. 3

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *IJCV*, 2015. 4

[30] Y. Shi, Y. Tian, Y. Wang, W. Zeng, and T. Huang. Learning long-term dependencies for action recognition with a biologically-inspired deep network. In *ICCV*, 2017. 7

[31] G. A. Sigurdsson, S. Divvala, A. Farhadi, and A. Gupta. Asynchronous temporal fields for action recognition. In *CVPR*, 2017. 5, 7

[32] G. A. Sigurdsson, G. Varol, X. Wang, A. Farhadi, I. Laptev, and A. Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *ECCV*, 2016. 1, 2, 4, 5, 7

[33] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014. 2, 6, 7

[34] K. Soomro, A. Roshan Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. In *CRCV-TR-12-01*, 2012. 2, 5, 6, 7

[35] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *ICML*, 2015. 7

[36] O. Sukmarg and K. R. Rao. Fast object detection and segmentation in MPEG compressed domain. In *TENCON*, 2000. 3

[37] L. Sun, K. Jia, K. Chen, D.-Y. Yeung, B. E. Shi, and S. Savarese. Lattice long short-term memory for human action recognition. In *ICCV*, 2017. 7

[38] B. U. Töreyin, A. E. Cetin, A. Aksay, and M. B. Akhan. Moving object detection in wavelet compressed video. *Signal Processing: Image Communication*, 2005. 3

[39] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015. 2, 6, 7

[40] D. Tran, J. Ray, Z. Shou, S.-F. Chang, and M. Paluri. ConvNet architecture search for spatiotemporal feature learning. *arXiv preprint arXiv:1708.05038*, 2017. 1, 2, 6, 7, 11

[41] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Dense trajectories and motion boundary descriptors for action recognition. *IJCV*, 2013. 2

[42] H. Wang and C. Schmid. Action recognition with improved trajectories. In *ICCV*, 2013. 1, 2, 7

[43] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In *BMVC*, 2009. 2

[44] 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 *ECCV*, 2016. 2, 4, 5, 6, 7, 11

[45] Y. Wang, M. Long, J. Wang, and P. S. Yu. Spatiotemporal pyramid network for video action recognition. In *CVPR*, 2017. 7

[46] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NIPS*, 2015. 11

[47] B.-L. Yeo and B. Liu. Rapid scene analysis on compressed video. *IEEE Transactions on circuits and systems for video technology*, 1995. 3

[48] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015. 2

[49] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l 1 optical flow. *Pattern Recognition*, 2007. 5

[50] B. Zhang, L. Wang, Z. Wang, Y. Qiao, and H. Wang. Real-time action recognition with enhanced motion vector CNNs. In *CVPR*, 2016. 3, 6, 7

Appendix A. RNN-Based Models

Given the recurrent definition of P-frames, one can use a RNN to model a compressed video. In preliminary experiments, we experiment with a variant using Conv-LSTMs [46].

The architecture is identical to CoViAR except that i) it uses the original $\mathcal{T}$ and Δ instead of the accumulated $\mathcal{D}$ and $\mathcal{R}$, because here we want to the original dependency, and ii) it uses a Conv-LSTM to aggregate the CNN features instead of average pooling. Formally, let $x^{(t)}_{\text{fusion}} := \max\left(x^{(t)}_{\text{motion}}, x^{(t)}_{\text{residual}}\right)$ denote the max-pooled P-frame feature at time t. The Conv-LSTM takes the input sequence

$$\left(x^{(0)}_{\text{RGB}}, x^{(1)}_{\text{fusion}}, x^{(2)}_{\text{fusion}}, \dots\right).$$

Here the number of channels of $x^{(0)}_{\text{RGB}}$ is reduced from 2048 to 512 by an 1×1 convolution so that its dimensionality matches $x^{(t)}_{\text{fusion}}$. We use 512-dimensional hidden states and 3×3 kernels for the Conv-LSTM. Due to memory constraint, we subsample one every two P-frames to reduce the sequence length.

Table 8 presents the results. Even though the Conv-LSTM model outperforms traditional RGB-based methods, the decoupled CoViAR achieves the best performance. We also try adding the input of Conv-LSTM to its output as a skip connection, but it leads to worse performance (Conv-LSTM-Skip).

RGB-only	Conv-LSTM	Conv-LSTM-Skip	CoViAR
88.4	89.1	87.8	**90.8**

Table 8: Accuracy on UCF-101 split 1. CoViAR decouples the long dependency and outperforms RNN-based models.

Appendix B. Feature Fusion

We experiment with different ways of combining P-frame features, $x^{(t)}_{\text{motion}}$, $x^{(t)}_{\text{residual}}$, and I-frame features $x^{(0)}_{\text{RGB}}$. In particular, we evaluate maximum, mean, and multiplicative fusion, concatenation of feature maps, and late fusion (summing softmax scores). For maximum, mean, and multiplicative fusion, we perform 1×1 convolution on I-frame feature maps before fusion, so that their dimensionality matches P-frame features.

Table 9 summarizes the results; we found late fusion works the best for CoViAR. Note that late fusion allows training of a decoupled model, while the rest requires training multiple CNNs jointly. The ease of training of late fusion may also contribute to its superior performance.

Max	Mean	Mult	Concat	Late
87.9	88.1	87.8	89.7	**90.8**

Table 9: Accuracy on UCF-101 split 1 with different feature fusion methods.

Appendix C. CoViAR without Temporal Segments

For further analysis, we also evaluate CoViAR without using temporal segments [44] (Table 10). It still significantly outperforms models using RGB images only, including ResNet-152 (83.4% in ST-Mult [8]; 84.7% with out implementation) and Res3D [40] (85.8%).

I	M	R	I+M	I+R	I+M+R
84.7	63.4	76.6	87.9	87.2	**88.9**

Table 10: Accuracy of CoViAR without temporal segments on UFC-101 split 1.

Appendix D. Confusion Matrix

Figure 8 and Figure 9 show the confusion matrices of CoViAR and the model using only RGB images respectively, on UCF-101. Figure 10 shows the difference between their predictions. We can see that CoViAR corrects many mistakes made by the RGB-based model (off-diagonal purple blocks in Figure 10). For example, while the RGB-based model gets confused about the similar actions of *Cricket Bowling* and *Cricket Shot*, our model better distinguishes between them.

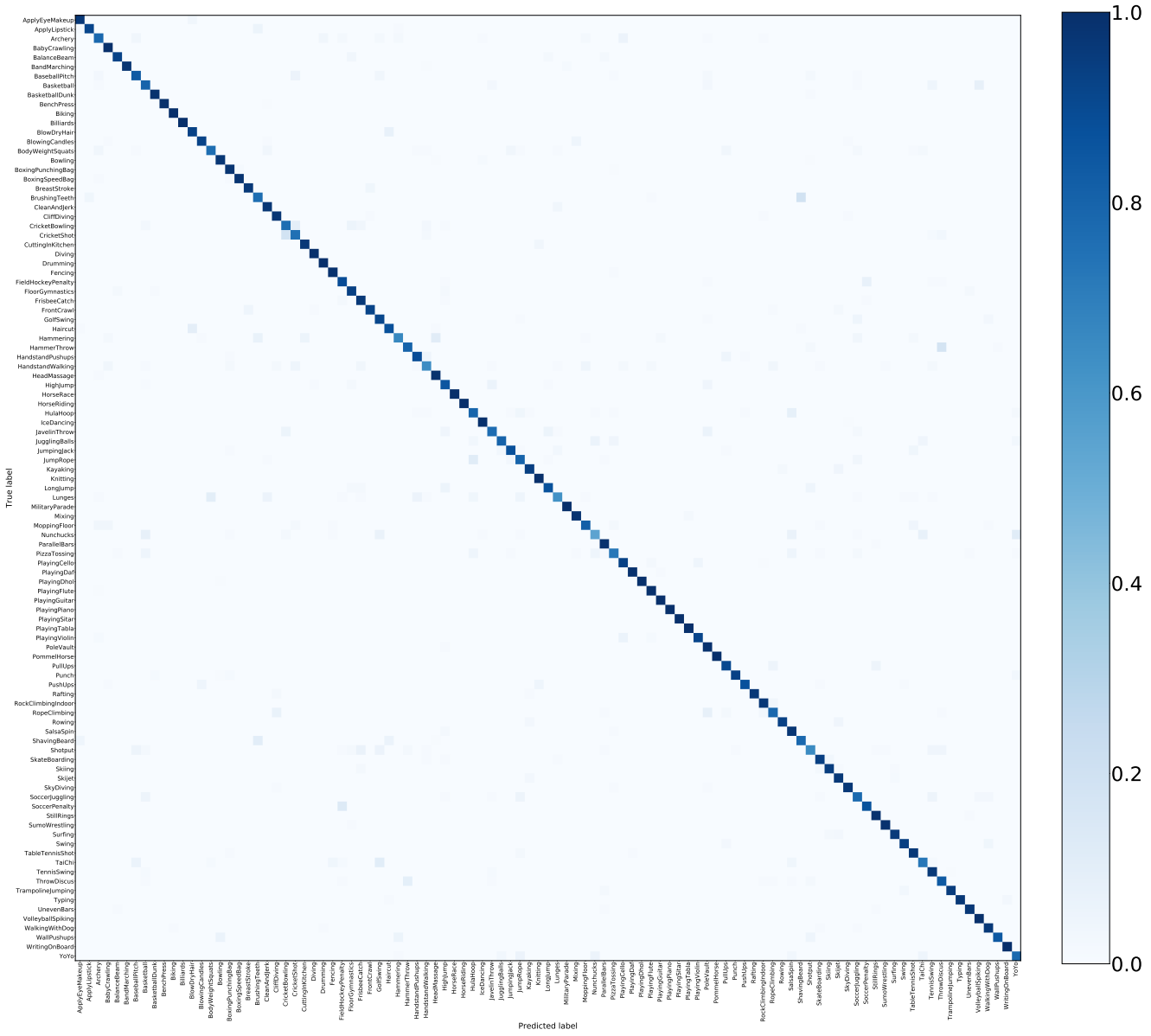


Figure 8: Confusion matrix of CoViAR on UCF-101.

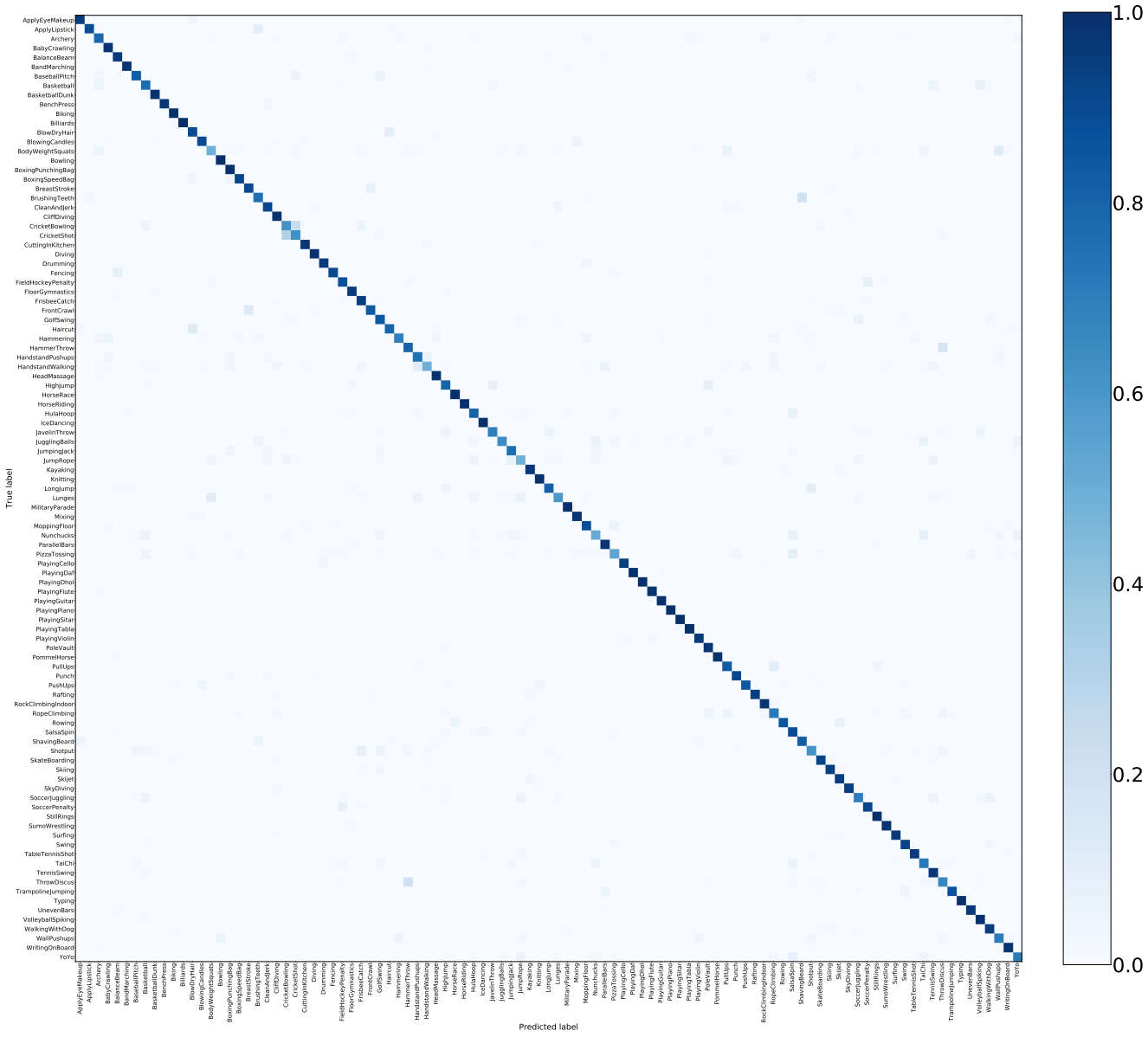


Figure 9: Confusion matrix of the model using RGB images on UCF-101.

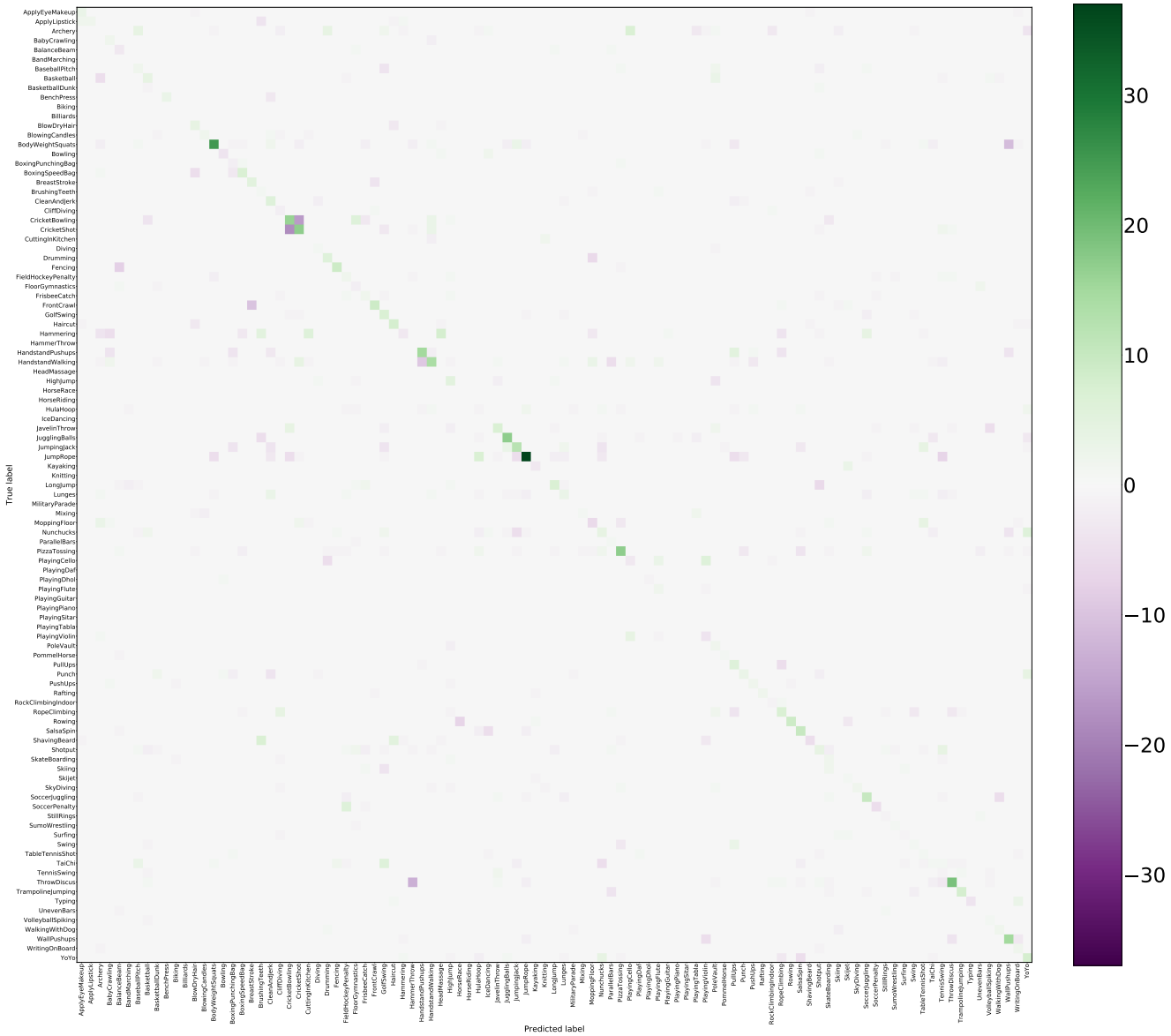


Figure 10: Difference between CoViAR's predictions and the RGB-based model's predictions. For diagonal entries, positive values (in green) is better (increase of correct predictions). For off-diagonal entries, negative values (purple) is better (reduction of wrong predictions).