Supplementary Material
AMTnet: Action-Micro-Tube Regression by End-to-end Trainable Deep Architecture

Suman Saha Gurkirt Singh Fabio Cuzzolin
Oxford Brookes University, Oxford, United Kingdom
{suman.saha-2014, gurkirt.singh-2015, fabio.cuzzolin}@brookes.ac.uk

1. Implementation details

We implement our method using Torch 7 [1]. To develop our codebase, we take coding reference from the publicly available repository [5]. We use the coding implementation of bilinear interpolation [7] (§ Section **3.4** in the main paper) for ROI feature pooling. Our micro-tube linking algorithm (ML) (§ Section **5** in the main paper) is implemented in MATLAB.

In all our experiments, at training time we pick top 2000 RPN generated 3D proposals using NMS (non-maximum suppression). At test time we select top 1000 3D proposals. However, a lower number of proposals, e.g. top 300 proposals does not effect the detection performance, and increase the test time detection speed significantly. In Section 3.2, we show that extracting less number of 3D proposals (at test time) does not effect the detection performance. Shaoqing *et al*. [8] observed the same with Faster-RCNN.

For UCF-101, we report test time detection results (video-mAP) using two different action-tube generation algorithms. Firstly, we link the micro-tubes predicted by the proposed model (at test time) using our **m**icro-tube **l**inking (ML) algorithm (§ Section **5** in the main paper). we denote this as *"Ours-ML"* in Table **6** (main paper). Secondly, we construct final action-tubes from the predicted micro-tubes using the **2 p**ass **d**ynamic **p**rogramming (2PDP) algorithm proposed by [9]. We denote this as *"Ours-2PDP"* in Table **6** (main paper). The results in Table **1**, **3**, **4** and **5** are generated using our new micro-tube linking algorithm (*"Ours-ML"*). Further, we cross-validate the class-specific α_c as in Section 3.4 of Saha *et al*.'s paper [9], and generate action-tubes using these cross-validated α_c values. We denote the respective results using an asterisk (**'*'**) symbol in Table **6** (main paper).

1.1. Mini-batch sampling

In a similar fashion [3], we construct our gradient descent mini-batches by first sampling N pairs of successive video frames, and then sampling R 3D proposals for each pair. In practice, we set $N = 1$ and $R = 256$ in all our experiments. We had one concern over this way of sampling training examples because, all the positive 3D proposals from a single training batch (i.e. a pair of video frames) belong to only one action category [1] (that is, they are correlated), which may cause slow training convergence. However, we experience a fast training convergence and good detection results with the above sampling strategy.

1.2. Data preprocessing

The dimension of each video frame in both J-HMDB-21 and UCF-101 is $[320 \times 240]$. We scale up each frame to dimension $[800 \times 600]$ as in [8]. Then we swap the RGB channels to BGR and subtract the VGG image mean $\{103.939, 116.779, 123.68\}$ from each BGR pixel value.

1.3. Data augmentation

We augment the training sets by flipping each video frame horizontally with a probability of 0.5.

1.4. Training batch

Our training data loader script constructs a training batch which consists of: a) a tensor of size $[2 \times D \times H \times W]$ containing the raw RGB pixel data for a pair of video frames, where $D = 3$ refers to the 3 channel RGB data, $H = 600$ is the image height and $W = 800$ is the image width; b) a tensor of size $[2 \times T \times 6]$ which contains the ground-truth micro-tube annotation in the following format: [*fno tid* x_c y_c w h], where T is the number of micro-tubes, *fno* is the frame number of the video frame, *tid* is an unique identification number assigned to each individual action tube within a video, $\{x_c, y_c\}$ is the center and w and h are the width and height of the ground-truth bounding box; c) a $[1 \times T]$ tensor storing the action class label for each micro-tube. The J-HMDB-21 (Model-11+32) train set has 58k training batches, and UCF-101 train set consists of 340k training batches.

[1] Each video clip of UCF-101 and J-HMDB-21 is associated with a single class label. Therefore, a pair of video frames belongs to a single action class.

1.5. Training iteration

Our model requires at least 2 training epochs because, in the first training epoch we freeze the weights of all the convolutional layers and only update the weights of the rest of the network. We start updating the weights of the convolutional layers (alongside other layers) in the second epoch. We stop the training after 195k and 840k iterations for J-HMDB-21 and UCF-101 respectively. The training times required for J-HMDB-21 and UCF-101 are 36 and 96 GPU hours respectively using a single GPU. The training time can be further reduced by using two or more GPUs in parallel.

2. Fusion methods

A fusion function $f : \mathbf{x}^t, \mathbf{x}^{t+\Delta}, \rightarrow y$ fuses two convolution feature maps $\mathbf{x}^t, \mathbf{x}^{t+\Delta} \in \mathbb{R}^{H' \times W' \times D}$ to produce an output map $y \in \mathbb{R}^{H' \times W' \times D}$, where W', H' and D are the width, height and number of channels of the respective feature maps [2]. In this work we experiment with the following two fusion methods.

Sum fusion. Sum fusion $y^{sum} = f^{sum}(\mathbf{x}^t, \mathbf{x}^{t+\Delta})$ computes the sum of the two feature maps at the same spatial locations, (i, j) and feature channels d:

$$y^{sum}_{i,j,d} = \mathbf{x}^t_{i,j,d} + \mathbf{x}^{t+\Delta}_{i,j,d} \tag{1}$$

where $1 \leq i \leq H', 1 \leq j \leq W', 1 \leq d \leq D$ and $\mathbf{x}^t, \mathbf{x}^{t+\Delta}, y \in \mathbb{R}^{H' \times W' \times D}$.

Mean fusion. Mean fusion is same as sum fusion, only the difference is, instead of computing the element-wise sum, here we compute the element-wise mean:

$$y^{mean}_{i,j,d} = (\mathbf{x}^t_{i,j,d} + \mathbf{x}^{t+\Delta}_{i,j,d})/2 \tag{2}$$

3. Experiments and discussions

3.1. Effect of different fusion methods on video-mAP

In Table 1 we report video-mAPs obtained using mean and sum fusion methods for J-HMDB-21 dataset. We train our model on the combined trainset (*set-11+32*) (§ Section **6.1** and **6.2** in the main paper). We train two models, one using mean and another using sum fusion and denote these two models in Table 1 as *Model-11+32 (mean-ML)* and *Model-11+32 (sum-ML)* respectively. Action-tubes are constructed using our micro-tube linking (ML) algorithm. We can observe that at higher IoU threshold $\delta = 0.5$, the sum fusion performs better and improve the mAP by almost 1%. As a future work, we would like to explore different spatial and temporal feature map fusion functions [2].

Table 1. *Effect of element-wise mean and sum fusion methods on video-mAP for J-HMDB-21 dataset (averaged over 3 splits).*

IoU threshold δ	0.1	0.2	0.3	0.4	0.5
Model-11+32 (mean-ML)	57.16	57.14	57.00	56.13	54.51
Model-11+32 (sum-ML)	**57.79**	**57.76**	**57.68**	**56.79**	**55.31**

3.2. Effect of the number of predicted 3D proposals

To investigate the effect of the number of predicted 3D proposals on detection performance, we generate video-mAPs using two different sets of detections on J-HMDB-21 dataset. One detection set is generated by selecting top 1000 3D proposals and another set is by selecting top 300 3D proposals at test time using NMS. Once the two sets of detections are extracted, predicted micro-tubes are then linked up in time to generate final action tubes. Subsequently, video-mAPs are computed for each set of action tubes. The corresponding video-mAPs for each detection set at different IoU thresholds are reported in Table 2. We denote these two detection sets in Table 2 as *Detection-1000* and *Detection-300*. It is quite apparent that reduced number of RPN proposals does not effect the detection performance.

Table 2. *Effect of the number of predicted 3D proposals on video-mAP for J-HMDB-21 dataset (averaged over 3 splits).*

IoU threshold δ	0.1	0.2	0.3	0.4	0.5
Detection-1000	57.79	57.76	57.68	56.79	**55.31**
Detection-300	**57.91**	**57.89**	**57.84**	**56.87**	55.26

3.3. Loss function hyper-parameters

We have four hyper-parameters λ^e_{cls}, λ^e_{loc}, λ^m_{cls} and λ^m_{loc}, in our multi-task loss function (see Equation **(3)** in the main-paper) which weigh the relative importance of the four loss terms. To investigate the effect of these hyper-parameters on video-mAP, we train our model with different combinations of these four hyper-parameters on J-HMDB-21 split-1. The trainset is generated as per *scheme-11* (see Section **(6.1)** in the main paper). The video-mAPs of these trained models are presented in Table 3. We can observe that when the weigths for the *mid classifcation* (λ^m_{cls}) and *regression* (λ^m_{loc}) layers' loss terms are too low (e.g. 0.1 & 0.05), the model has the worst detection performance. When all weights are set to 1, then the model exhibits good detection performance. However, we get the best video-mAPs with $\lambda^e_{cls} = 1.0$, $\lambda^e_{loc} = 1.0$, $\lambda^m_{cls} = 0.5$ and $\lambda^m_{loc} = 0.5$. In all our experiments we set all 4 weights to 1. As a future work, we will explore the setting $[1.0, 1.0, 0.5, 0.5]$.

3.4. Ablation study

An ablation study of the proposed model is presented in Section 3.5. Besides, as a part of the ablation study, per

Table 3. *Effect of different combinations of hyper-parameters on video-mAP for J-HMDB-21 split-1 train set.*

Hyper-parameters				IoU threshold δ				
λ^e_{cls}	λ^e_{loc}	λ^m_{cls}	λ^m_{loc}	0.1	0.2	0.3	0.4	0.5
1.0	1.0	0.1	0.05	55.03	55.03	54.63	53.17	50.33
1.0	1.0	0.1	0.1	55.62	55.62	55.47	54.47	50.51
1.0	1.0	0.5	0.25	56.3	56.3	55.91	54.76	52.30
1.0	1.0	0.5	0.5	**57.3**	**57.13**	**56.79**	**55.82**	**53.81**
1.0	1.0	1.0	1.0	56.86	56.85	56.57	55.89	52.78

Model-11-2PDP

class frame- and video-APs of J-HMDB-21 dataset are reported in Table **2**, and per class video-APs of UCF-101 are presented in Table **3** in the main paper.

3.5. Discussion

Why there is no confusion matrix present for different action classes?
The paper is about action detection, where evaluation is by class-wise average precision(AP) rather than classification accuracy, a confusion matrix cannot be used.

Is the proposed model limited to learn from pairs of consecutive frames?
Our model is not limited to learn from pairs of consecutive frames, but can learn from pairs at any arbitrary interval Δ (see Figure **2(a)** in the main paper).

"Two consecutive frames will produce identical or almost identical outputs with no displacement information" – is that true?
To confute this point we conducted an **ablation study** of our model which is discussed below. For consecutive frames, we trained our model on J-HMDB-21 (split-01) dataset by passing training pairs composed of identical frames, e.g. passing the video frame pair $(65, 65)$ instead of $(65, 66)$. As you can see in Table 4, video-mAP drops significantly by 8.13% (at IoU threshold $\delta = 0.5$) which implies that the two streams do not output identical representations.
To double-check, we also extracted the two VGG-16 conv feature maps (see Figure **(b)** in the main paper) for each test frame pair $((f_t,\ f_{t+1}))$ of J-HMDB-21 and UCF-101 datasets. For each pair of conv feature maps, we first flattened them into feature vectors, and then computed the normalised $L2$ distance between them. For identical frames we found that the $L2$ distance is 0 for both J-HMDB-21 and UCF-101 datasets. Whereas, for consecutive frames it is quite high, in case of J-HMDB-21 the mean $L2$ distance is 0.67; for UCF-101 the mean $L2$ distance is 0.77 which again implies that the two streams generate significantly different feature encoding even for pairs consist of consecutive video frames.

Table 4. *An ablation study on J-HMDB-21 (split-01). Video-mAP is computed at IoU threshold $\delta = 0.5$.*

Model	video-mAP (%)
Model-01	48.9
Model-02	52.7
Model-03	**57.1**

Model-01: Training pairs with identical frames
Model-02: Training pairs with consecutive frames (model-11)
Model-03: Training pairs with mixture of consecutive and successive frames (model-11+32)

3.6. Training and test time requirements

Training time. Saha *et al*. reported [10] that the state-of-the-art [4, 11] action detection methods require at least 6+ days to train all the components (including fine-tuning CNNs, CNN feature extraction, one vs rest SVMs) of their detection pipeline for UCF-101 trainset (split-01). In our case, we need to train the model once which requires 96 hours for UCF-101 and 36 hours for J-HMDB-21 to train. The training and test time calculations are done considering a single NVIDIA Titan X GPU. The computing time requirement for different detection methods are presented in Table 5. Our model requires 2 days less training time as compared to [4, 11] on UCF-101 trainset.

Test time. We report video-level detection speed comparison of our method with [4, 11, 9] on J-HMDB-21 dataset. The figures are shown in Table 6. Our method exhibits the fastest test-time detection speed with 8.5 Sec./video as compared to [4, 11, 9]. Peng *et al*. [6] do not report any computing time requirement analysis in their paper, so we could not compare our figures with them.

Table 5. *Training time on UCF-101 dataset.*

Methods	days
[4]	6+
[11]	6+
[9]	**3+**
ours	4

Table 6. *Test time on J-HMDB-21 dataset.*

Methods	Avgerage time (Sec./video)
[4]	113.52
[11]	52.23
[9]	10.89
ours	**8.5**

3.7. Qualitative results

Spatiotemporal action detection results on UCF-101. We show the spatiotemporal action detection qualitative results in Figures 1 and 2. To demonstrate the robustness of

the proposed detector against temporal action detection, we select those action categories which have highly temporally untrimmed videos. We select action classes *VolleyballSpiking*, *BasketballDunk* and *CricketBowling*. For *VolleyballSpiking* class, the average temporal extent of the action in each video is 40%, that means, the remaining 60% of the video doesn't contain any action. Similarly, for *BasketballDunk* and *CricketBowling* classes, we have average durations 41% and 46% respectively.

Video clip **(a)** (§ Figures 1) has duration 107 frames and the action *VolleyballSpiking* takes place only between frames 58 to 107. Note that our method able to successfully detect the temporal extent of the action (alongside spatial locations) which closely matches the ground-truth. We can observe similar quality of detection results for video clip **(b)** and **(c)** (§ Figures 1) which have durations 41 and 94 frames and the temporal extent of action instances are between frames 17 to 41 and frames 75 to 94 respectively for *BasketballDunk* and *CricketBowling*. Video clips **(a)** and **(b)** in Figures 2 show some more spatiotemporal detection results for action classes *BasketballDunk* and *CricketBowling*.

Figures 3 shows sample detection results on UCF-101. Note that in **(1)**, the 2nd "biker" is detected in spite of partial occlusion. Figures 3 **(1)**, **(2)**, **(3)** and **(5)** are examples of multiple action instance detection with complex real world scenarios like 3 fencers (§ **(2)**) and bikers (§ **(3)**). Further, note that the detector is robust against *scale changes* as the 3rd fencer (§ **(2)**) and the biker (§ **(3)**) are detected accurately in spite of their relatively smaller shapes.

Spatiotemporal action detection results on J-HMDB-21. Figure 4 presents the detection results of our model on J-HMDB-21 dataset. In Figure 4 **(1)**, **(2)** and **(3)**, the actions "run" and "sit" are detected accurately in spite of large variations in illumination conditions, which shows that our detector is robust against *illumination changes*. In Figure 4 **(5)**, **(6)** and **(7)**, the actions "jump" and "run" are detected successfully. Note that due to fast motion, these video frames are affected by *motion blur*. Further, in Figure 4 **(9)** to **(12)**, actions "stand" and "sit" are detected with correct action labels. Even for human, it is hard to infer which instance belong to "stand" and "sit" class. This again tells that our classifier is robust against inter-class similarity.

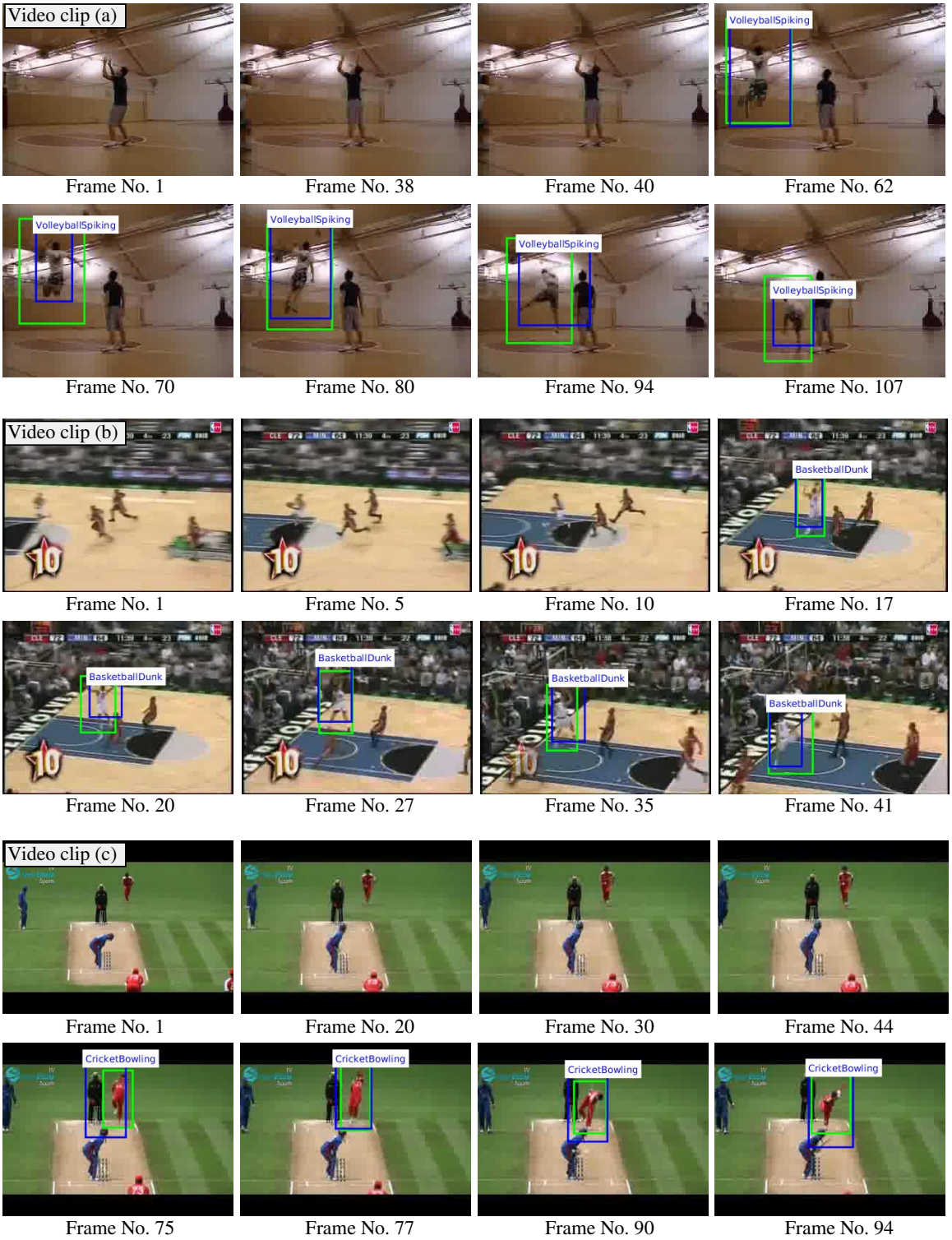

Figure 1. *Spatiotemporal action detection results. Video clips* ***(a)****,* ***(b)*** *and* ***(c)*** *are test videos belong to UCF-101 action classes* VolleyballSpiking*,* BasketballDunk *and* CricketBowling *respectively.*

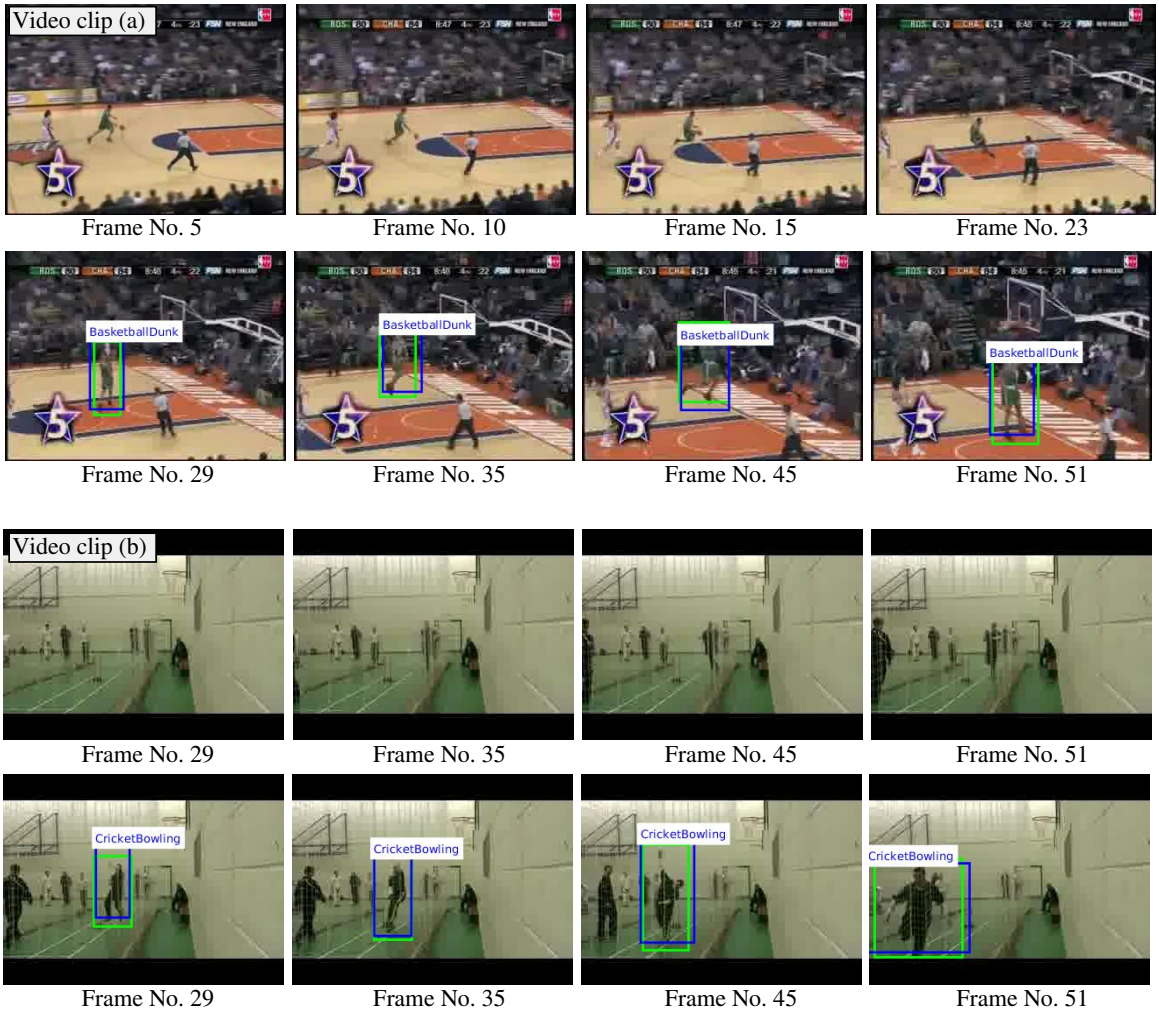

Figure 2. *Spatiotemporal action detection results. Video clips* ***(a)*** *and* ***(b)*** *are test videos belong to UCF-101 action classes* BasketballDunk *and* CricketBowling *respectively.*

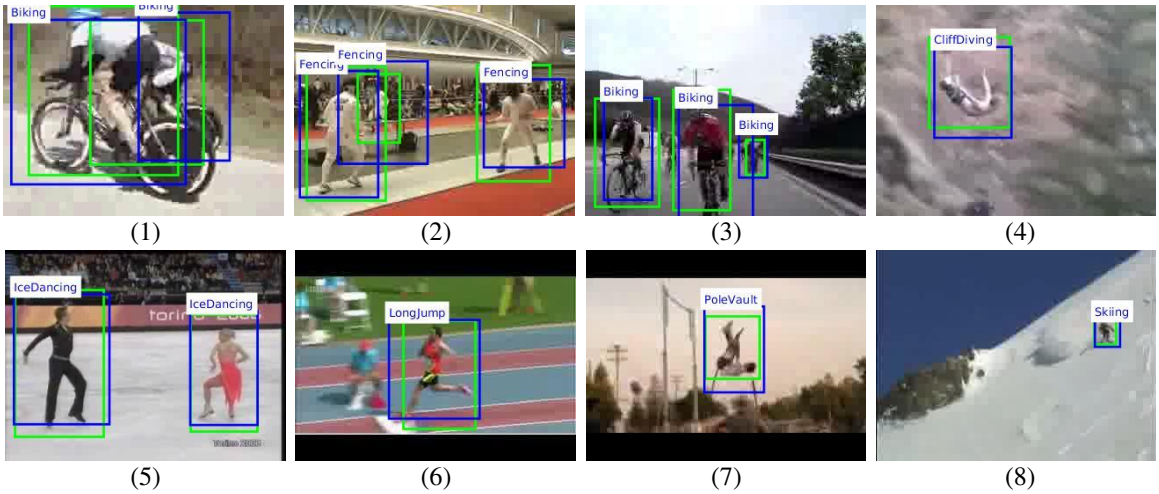

Figure 3. *More sample detection results on UCF-101 test videos.*

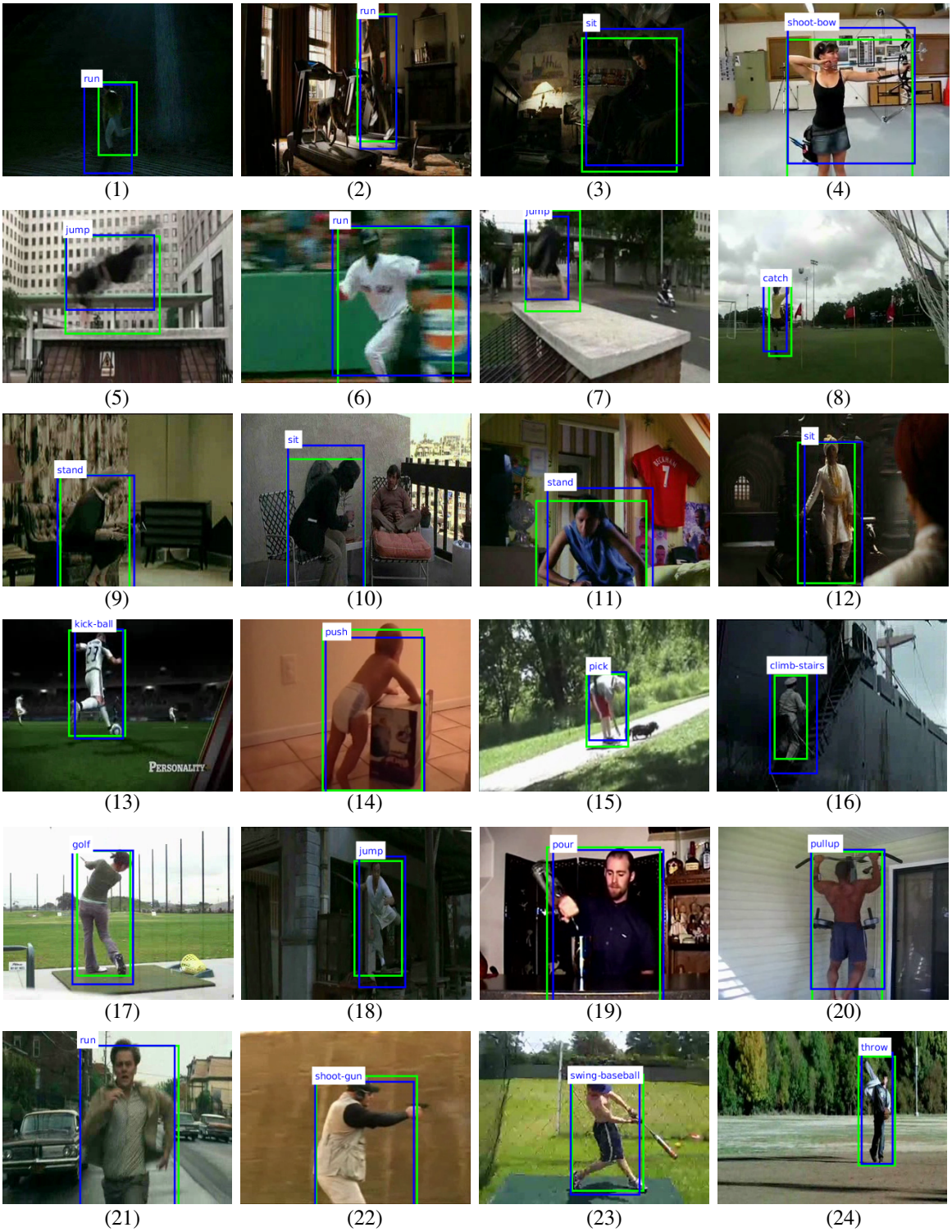

Figure 4. *Spatiotemporal action detection results on J-HMDB-21 test videos.*

References

[1] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011. 1

[2] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016. 2

[3] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015. 1

[4] G. Gkioxari and J. Malik. Finding action tubes. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2015. 3

[5] jcjohnson. densecap, 2016. https://github.com/jcjohnson/densecap. 1

[6] X. Peng and C. Schmid. Multi-region two-stream R-CNN for action detection. In *ECCV 2016 - European Conference on Computer Vision*, Amsterdam, Netherlands, Oct. 2016. 3

[7] qassemoquab. stnbhwd, 2015. https://github.com/qassemoquab/stnbhwd. 1

[8] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015. 1

[9] S. Saha, G. Singh, M. Sapienza, P. H. S. Torr, and F. Cuzzolin. Deep learning for detecting multiple space-time action tubes in videos. In *British Machine Vision Conference*, 2016. 1, 3

[10] S. Saha, G. Singh, M. Sapienza, P. H. S. Torr, and F. Cuzzolin. Deep learning for detecting multiple space-time action tubes in videos, 2016. http://tinyurl.com/map6lde. 3

[11] P. Weinzaepfel, Z. Harchaoui, and C. Schmid. Learning to track for spatio-temporal action localization. In *IEEE Int. Conf. on Computer Vision and Pattern Recognition*, June 2015. 3