Supplementary Material

AMTnet: Action-Micro-Tube regression by end-to-end trainable deep architecture

Anonymous ICCV submission

Paper ID 1935

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 - main paper) for ROI feature pooling. Our micro-tube linking algorithm (ML) (§ Section 5) is implemented in MATLAB. In all our experiments, at training time, we pick top 2000 RPN 3D proposals using NMS (non-maximum suppression). At test time we select the top 1000 RPN proposals. However, a lower number of RPN proposals, e.g. top 300, does not effect the detection performance, and increase the test time detection speed significantly. In this supplementary report (§ Section 3.2), we show that extracting less number of proposals (at test time) does not effect the detection performance. Shaoqing et al. [8] observed the same with Faster-RCNN.

We build action-tubes using the micro-tubes predicted by our proposed model (§ Section 3.5 - main paper). For UCF-101, we report detection results (video-mAP) using two different action-tube generation algorithms. Firstly, we link the predicted micro-tubes to construct action tubes using our micro-tube linking algorithm (§ Section 5 - main paper). In Table 6 (main paper) we denote this as Ours-ML. Secondly, we construct action-tubes using the tube generation 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. Further, we cross-validate the class-specific $\alpha_c$ as in [9] and generate action-tubes using these cross-validated $\alpha_c$ 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 3D proposals from a single training batch (i.e. a pair of video frames) belong to only one action category (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 the video frame.

1.3. Data augmentation

We augment the training sets by flipping each video frame horizontally with a probability of 0.5, thus doubling the number of training video frames.

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.

1.5. Training iteration

Our model requires at least 2 training epochs, as in the first training epoch we freeze the weights of the convolution networks and update only the weights of the rest of the
network. We start fine-tuning the layers of the two parallel CNNs after completion of 1st epoch. For J-HMDB-21 and UCF-101, we use 195k and 840k training iterations respectively. We stop the training after 195k iterations for J-HMDB-21, and 840k iterations for UCF-101. Training times are 36 and 96 GPU hours respectively. For J-HMDB-21 and UCF-101, the computing time required to train our model are 36 and 96 hours respectively.

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

$$y_{i,j,d}^{\text{sum}} = \mathbf{x}_{i,j,d}^t + \mathbf{x}_{i,j,d}^{t+\Delta} \quad (1)$$

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:

$$y_{i,j,d}^{\text{mean}} = (\mathbf{x}_{i,j,d}^t + \mathbf{x}_{i,j,d}^{t+\Delta})/2 \quad (2)$$

3. Experiments

3.1. Effect of different fusion method 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 using training batches generated using scheme-II and scheme-32 (§ 6.1 - main paper). We train two models, one using mean (Model-11+32 (mean-ML)) in Table 1) and another using sum (Model-11+32 (sum-ML)) fusion. Action-tubes are constructed using our micro-tube linking algorithm and thus denoted by ML. 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].

3.2. Effect of the number of RPN 3D proposals on video-mAP

To investigate the effect of detection performance with less number of region proposals, we generate video-mAPs for two set of detections. One detection set is generated by selecting top 1000 RPN proposals (Model-11+32 (sum-ML-1000)) at test time using NMS, another set (Model-11+32 (sum-ML-300)) with top 300 proposals. The video-mAPs at different IoU thresholds are reported in Table 2. From these figures, it is quite apparent that reduced number of RPN proposals does not effect the detection performance.

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

<table>
<thead>
<tr>
<th>IoU threshold $\delta$</th>
<th>Mean Fusion</th>
<th>Sum Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>55.79</td>
<td>57.91</td>
</tr>
<tr>
<td>0.2</td>
<td>57.16</td>
<td>57.79</td>
</tr>
<tr>
<td>0.3</td>
<td>57.14</td>
<td>57.76</td>
</tr>
<tr>
<td>0.4</td>
<td>57.00</td>
<td>57.68</td>
</tr>
<tr>
<td>0.5</td>
<td>56.13</td>
<td>56.79</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>IoU threshold $\delta$</th>
<th>1000 RPN Proposals</th>
<th>300 RPN Proposals</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>55.79</td>
<td>57.91</td>
</tr>
<tr>
<td>0.2</td>
<td>57.16</td>
<td>57.79</td>
</tr>
<tr>
<td>0.3</td>
<td>57.14</td>
<td>57.76</td>
</tr>
<tr>
<td>0.4</td>
<td>57.00</td>
<td>57.68</td>
</tr>
<tr>
<td>0.5</td>
<td>56.13</td>
<td>56.79</td>
</tr>
</tbody>
</table>

3.3. Loss function hyper-parameters

We have four hyper-parameters $\lambda_{cls}^e, \lambda_{loc}^e, \lambda_{cls}^m$ and $\lambda_{loc}^m$ in our multi-task loss function (§ Eq. (3) 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 4 different combinations of these hyper-parameters. The training set is generated following the sampling scheme-II (§ 6.1 - main paper) for J-HMDB-21 split-1 trainset. The video-mAPs of these 4 trained models are presented in Table 3.

We can observe that when the weights for the mid classification ($\lambda_{cls}^m$) and regression ($\lambda_{loc}^m$) 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 $\lambda_{cls}^e = 1.0, \lambda_{loc}^e = 1.0, \lambda_{cls}^m = 0.5$ and $\lambda_{loc}^m = 0.5$. In 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].

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

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>$\lambda_{cls}^e$</th>
<th>$\lambda_{loc}^e$</th>
<th>$\lambda_{cls}^m$</th>
<th>$\lambda_{loc}^m$</th>
<th>IoU threshold $\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>55.03</td>
<td>55.03</td>
<td>54.63</td>
<td>53.17</td>
<td>50.33</td>
</tr>
<tr>
<td>1.0</td>
<td>56.62</td>
<td>56.62</td>
<td>55.47</td>
<td>54.47</td>
<td>50.51</td>
</tr>
<tr>
<td>0.5</td>
<td>56.3</td>
<td>56.3</td>
<td>55.91</td>
<td>54.76</td>
<td>52.30</td>
</tr>
<tr>
<td>0.25</td>
<td>57.13</td>
<td>56.97</td>
<td>55.82</td>
<td>53.81</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>56.86</td>
<td>56.85</td>
<td>56.57</td>
<td>55.89</td>
<td>52.78</td>
</tr>
</tbody>
</table>
3.4. Computing time requirements

**Training time**. Saha et al. [9] reported in their supplementary material [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 split-1 train set. In our case, we need to train a single model once which requires 96 hours for UCF-101 and 36 hours for J-HMDB-21 to train. The training and test time calculation is done considering a single training/testing script runs on a single NVIDIA Titan X GPU. The computing time required during training for different detection methods are presented in Table 4. Our model requires 2 days less training time as compared to [4, 11] while training our model on UCF-101 trainset.

**Test time**. Saha et al. [9] present a fair comparison of video-level test-time detection speed [10] of their approach [9] with [4, 11]. Similarly, 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 4. 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>
<thead>
<tr>
<th>Methods</th>
<th>Average time (Sec./video)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>113.52</td>
</tr>
<tr>
<td>[11]</td>
<td>52.23</td>
</tr>
<tr>
<td>[9]</td>
<td>10.89</td>
</tr>
<tr>
<td>ours</td>
<td>8.5</td>
</tr>
</tbody>
</table>

3.5. 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%.

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 (e) (§ Figures 1) which have total duration 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.

4. Typographical errors in the main paper

There are a few typographical errors in the main paper which are corrected here. We mistakenly mentioned in the main paper (§ line no. 444-445) that at test time we pick top 2000 3D proposals. At test time we select top 1000 proposals.

In line no. 752-755, we mentioned “relative mAP improvement”, which should be “relative AP improvement”. At line no. 754, the correct figures are 16.9%, 10.8% and 1.5% instead of the incorrect figures 60%, 69% and 73%.
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

matlab-like environment for machine learning. In BigLearn,
NIPS Workshop, number EPFL-CONF-192376, 2011. 1

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

tional Conference on Computer Vision, pages 1440–1448,
2015. 1

Conf. on Computer Vision and Pattern Recognition, 2015. 3

jcjohnson/densecap. 1

for action detection. In ECCV 2016 - European Conference
on Computer Vision, Amsterdam, Netherlands, Oct. 2016. 3

qasemoquab/stnbhwd. 1

[8] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: To-
wards real-time object detection with region proposal net-
works. In Advances in Neural Information Processing Sys-
tems, pages 91–99, 2015. 1

zolin. Deep learning for detecting multiple space-time action
1, 3

[10] S. Saha, G. Singh, M. Sapienza, P. H. S. Torr, and F. Cuz-
zolin. Deep learning for detecting multiple space-time action

track for spatio-temporal action localization. In IEEE Int.
Conf. on Computer Vision and Pattern Recognition, June
2015. 3