1. Model Architectures

We illustrate the model architectures that are used in the ConST-CL framework.

1.1. Base network \( f(\cdot) \)

Table 1 describes the base model architecture that is proposed for reconciling global and local training signals.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Network</th>
<th>Input from</th>
<th>Output size ( T \times S^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw clip</td>
<td>-</td>
<td>-</td>
<td>( 32 \times 224^2 )</td>
</tr>
<tr>
<td>data</td>
<td>stride 2, 1(^2)</td>
<td>raw clip</td>
<td>( 16 \times 224^2 )</td>
</tr>
<tr>
<td>res1</td>
<td>( 5 \times 7^2, 64 ) stride ( 2, 2^2 )</td>
<td>data</td>
<td>( 8 \times 112^2 )</td>
</tr>
<tr>
<td>pool1</td>
<td>( 1 \times 3^2 ) max stride ( 1, 2^2 )</td>
<td>res1</td>
<td>( 8 \times 56^2 )</td>
</tr>
<tr>
<td>res2</td>
<td>( 1 \times 1^2, 64 ), ( 1 \times 3^2, 64 ), ( 1 \times 2^2, 256 ) \times 3</td>
<td>pool1</td>
<td>( 8 \times 28^2 )</td>
</tr>
<tr>
<td>res3</td>
<td>( 1 \times 1^2, 128 ), ( 1 \times 3^2, 128 ), ( 1 \times 2^2, 512 ) \times 4</td>
<td>res2</td>
<td>( 8 \times 14^2 )</td>
</tr>
<tr>
<td>res4</td>
<td>( 3 \times 1^2, 256 ), ( 1 \times 3^2, 256 ), ( 1 \times 2^2, 1024 ) \times 6</td>
<td>res3</td>
<td>( 8 \times 7^2 )</td>
</tr>
<tr>
<td>res5(r)</td>
<td>( 3 \times 1^2, 512 ), ( 1 \times 3^2, 512 ), ( 1 \times 2^2, 2048 ) \times 3</td>
<td>res4</td>
<td>( 8 \times 7^2 )</td>
</tr>
<tr>
<td>res5(g)</td>
<td>( 3 \times 1^2, 512 ), ( 1 \times 3^2, 512 ), ( 1 \times 2^2, 2048 ) \times 3</td>
<td>res4</td>
<td>( 8 \times 7^2 )</td>
</tr>
</tbody>
</table>

Table 1. **Base network \( f(\cdot) \): a ResNet3D-50 (R3D-50) based encoder.**

1.2. ConST-CL head \( g(\cdot, \cdot) \)

Table 2 describes the projection head we use for achieving the instance prediction task.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Input, Dimension</th>
<th>Network</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear project</td>
<td>( h, N \times 2048 )</td>
<td>( n_{nodes}=128 )</td>
<td>Query</td>
</tr>
<tr>
<td>Linear project</td>
<td>( F'_h, M \times 2048 )</td>
<td>( n_{nodes}=128 )</td>
<td>Key</td>
</tr>
<tr>
<td>Linear project</td>
<td>( F'_z, M \times 2048 )</td>
<td>( n_{nodes}=128 )</td>
<td>Value</td>
</tr>
<tr>
<td>MHSA</td>
<td>Query, ( N \times 128 )</td>
<td>hidden_size=128</td>
<td>n_heads=3</td>
</tr>
<tr>
<td>MHSA</td>
<td>Key, ( M \times 128 )</td>
<td>Value, ( M \times 128 )</td>
<td>n_layers=3</td>
</tr>
<tr>
<td>Linear project</td>
<td>Hidden, ( N \times 128 )</td>
<td>( n_{nodes}=2048 )</td>
<td>z</td>
</tr>
</tbody>
</table>

Table 2. **ConST-CL head \( g(\cdot, \cdot) \): a transformer-based decoder.** The inputs are the region features \( h \) and the context features \( F'_z \) and the outputs are the transformed features \( z \). \( N \) and \( M \) are the number of tokens of \( h \) and \( F'_z \) respectively.

strained to have aspect ratio within \([0.5, 2]\) and size within \([0.1, 0.5]\) of the image size.

**Boxes from low-level image cues.** We use the SLIC [1] algorithm to generate 16 superpixels on each frame. Following [10], we alternatively use the graph-based image segmentation method [6] to generate 16 image segments for each frame. We use two scales to generate segments, the scale \( s \) and minimum cluster size \( c \), and \( s = c \in \{500, 1000\} \) in practice. After the segments generation, we convert each segment into its minimal bounding box and only keep those with width/height between \([0.05, 0.7]\) of the image width/height.

**Boxes from detectors.** We also use off-the-shelf modern detectors to generate object-centric bounding boxes for weakly supervised learning. A CenterNet-based [20] person detector is employed to generate bounding boxes on persons only. As an alternative, we use a generic object detector, which is based on Cascade RCNN [3].

2. Region Generation

In this section, we detail three options to generate region priors that we study for training ConST-CL.

**Random boxes.** For all of our related experiments, we randomly generate 8 boxes on each frame. The boxes are con-

3. Downstream tasks

3.1. Action Recognition

On all video action recognition datasets, we use the video clip of 32 frames with temporal stride 2 as input. During training, the temporally consistent random data augmentation [15] of cropping, resizing and flipping are applied and the resolution is set to \(224 \times 224\). During evaluation, we densely sample 10 clips with resolution \(256 \times 256\) from each video and apply a 3-crop evaluation follow-
Linear Evaluation. On action recognition datasets, we train a linear classifier with fixed backbone weights using the SGD optimizer with momentum of 0.9. On Kinetics400 [12], the linear classifier is trained for 100 epochs with learning rate of 32 and batch size of 1024. On UCF101 [16] and HMDB51 [13], the linear classifier is trained for 50 epochs with learning rate of 0.84 and batch size of 128. No dropout and weight decay are applied.

Fine-tuning. On UCF101 [16] and HMDB51 [13], we use the pre-trained models to initialize the network and fine-tune all layers for 50 epochs. We use batch size of 128, weight decay of $1 \times 10^{-5}$ and dropout rate of 0.5 during fine-tuning. The learning rate is set to 0.72 and 0.84 for UCF101 and HMDB51 respectively.

3.2. Spatio-temporal Action Localization

We use the same action transformer head as in [7, 14] to our R3D-50 backbone and follow the setting in [14]. The model is fine-tuned with batch size 256 for 50k steps, which is around 36 epochs on AVA-Kinetics [14]. The input has 32 frames with resolution 400 and temporal stride 2. We use the SGD optimizer with momentum 0.9 during the fine-tuning. On AVA-Kinetics, the learning rate is 1e-2 and the weight decay is 1e-7. On AVA [9], the learning rate is set to 3e-2 and the weight decay is 1e-4. During evaluation, we use the same set of detected boxes in [14] for AVA-Kinetics and in [5] for AVA v2.2 for a fair comparison.

3.3. Object Tracking

To evaluate on OTB2015 [17] dataset, we follow the same practice as in [8, 18, 19] to adopt the SiameseFC [2] as the tracker. Specifically, we modify the spatial stride and dilation rate to be (1, 2) and (1, 4) in the first layer of the res4 and res5 blocks. These modifications allow us to increase the feature map resolution without impacting on the pre-trained model. We fine-tune the tracker on GOT-10K [11] dataset using the SGD optimizer with momentum of 0.9. We use batch size of 256, learning rate of 0.1 and weight decay of 1e-4 and the tracker is fine-tuned for 20 epochs.

4. Visualization

4.1. Attention

We visualize the learned attention map during the training in Figure 1. For visualization purpose only, we use the boxes from the object detector to pool the region features in the source views to generate the attention maps. The model is trained with the randomly generated boxes as described in the paper. In Figure 1, we visualize the center frames in the source and the target views and the source frames are superimposed with one box for visualization. The zoomed-in thumbnails are presented in the second column. Given the context (features from the target views), we use these thumbnails’ region feature as the query to generate the attention maps shown in the fourth column. It is interesting to observe that the model learns to attend to not only the corresponding instance in the target view, but also to some other semantically meaningful objects the instance potentially interacts with.

4.2. Visual Object Tracking

We provide some qualitative results on visual object tracking on OTB2015 [17] in Figure 2. The results show that our tracker could robustly track objects under different scenarios.
Figure 1. **Visualization of the attention during the training.** We use boxes from the object detector to pool region features from the source view as the query in order to generate the attention maps given the context (features from the target view). Interestingly, the model learns to attend to not only the corresponding instance in the target frame, but also to some other semantically meaningful objects the instance potentially interacts with.
Figure 2. Qualitative results for visual object tracking on OTB2015 [17]. Best view in color.
References