CoCon: Cooperative-Contrastive Learning
Supplementary Material

A. Additional Details

A.1. Model Overview

We build our framework borrowing the learning framework present in [10] which learns video representations through spatio-temporal contrastive losses. It should be noted that even though we use this particular self-supervised backbone in our experiments, our approach is not restricted by the choice of the underlying self-supervised task.

A video V is a sequence of T frames (not necessarily RGB images) with resolution $H \times W$ and C channels, $\{\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_T\}$, where $\mathbf{i}_t \in \mathbb{R}^{H \times W \times C}$. Assume $T = N * K$, where N is the number of blocks and K denotes the number of frames per block. We partition a video clip V into N disjoint blocks $V = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, where $\mathbf{x}_j \in \mathbb{R}^{K \times H \times W \times C}$ and a non-linear encoder $f(.)$ transforms each input block x_j into its latent representation $z_j = f(x_j)$.

An aggregation function, $g(.)$ takes a sequence $\{z_1, z_2, \ldots, z_j\}$ as input and generates a context representation $c_j = g(z_1, z_2, \ldots, z_j)$. In our setup, $z_j \in \mathbb{R}^{H' \times W' \times D}$ and $c_j \in \mathbb{R}^D$. D represents the embedding size and H', W' represent down-sampled resolutions as different regions in z_j represent features for different spatial locations. We define $z'_j = Pool(z_j)$ where $z'_j \in \mathbb{R}^D$ and $c = F(V)$ where $F(.) = g(f(.))$. In our experiments, $H' = 4, W' = 4, D = 256$.

To learn effective representations, we create a prediction task involving predicting z of future blocks similar to [10]. In the ideal scenario, the task should force our model to capture all the necessary contextual semantics in c_t and all frame level semantics in z_t. We define $\phi(.)$ which takes as input c_t and predicts the latent state of the future frames. The formulation is given in Eq. (4).

$$\begin{aligned} \widetilde{z}_{t+1} &= \phi(c_t), \\ \widetilde{z}_{t+1} &= \phi(g(z_1, z_2, \ldots, z_t)), \\ \widetilde{z}_{t+2} &= \phi(g(z_1, z_2, \ldots, z_t, \widetilde{z}_{t+1})), \end{aligned} \tag{4}$$

where $\phi(.)$ takes c_t as input and predicts the latent state of the future frames. We then utilize the predicted $\widetilde{z}_{t+1}$ to compute $\widetilde{c}_{t+1}$. We can repeat this for as many steps as we want, in our experiments we restrict ourselves to predict till 3 steps in to the future.

Note that we use the predicted $\widetilde{z}_{t+1}$ while predicting $\widetilde{z}_{t+2}$ to force the model to capture long range semantics. We can repeat this for a varying number of steps, although the difficulty increases tremendously as the number of steps increases as seen in [10]. In our experiments, we predict the next three blocks using the first five blocks.

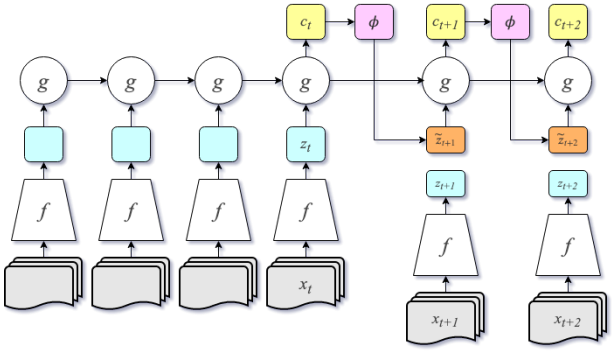


Figure 6: A diagram of the learning framework utilized. We look at features in a sequential manner while simultaneously trying to predict representations for future states.

A.2. Datasets

Kinetics400 contains 400 human action classes, with at least 400 real-world video clips for each action. Each clip lasts around 10s and is taken from a different YouTube video. The actions are human focused and cover a broad range of classes including human-object and human-human interactions. The large diversity and variance in the dataset make it an extremely challenging dataset.

HMDB51 dataset contains around 6800 real-world video clips from 51 action classes. These action classes cover a wide range of actions - facial actions, facial action with object manipulations, general body movement, and general body movements with human interactions. This dataset is challenging as it contains many poor quality video with significant camera motions and also the number of samples are not enough to effectively train a deep network. We report classification accuracy for 51 classes across 3 splits provided by the authors.

UCF101 dataset contains 13320 videos from 101 action classes that are divided into 5 categories - human-object interaction, body-movement only, human-human interaction, playing musical instruments and sports. Action classification in this datasets is challenging owing to variations in pose, camera motion, viewpoint and spatio-temporal extent of an action.

A.3. Views

We simultaneously learn encoders for RGB and Optical Flow while training on Kinetics-400. Instead of using the commonly used TVL1-Flow, we rely on Farneback flow

which usually results in lower performance for action classification, however is much faster to compute. For UCF101 and HMDB51, we simultaneously learn encoders for RGB, TVL1 Optical Flow, Pose Heatmaps and Semantic Maps.

We give a brief overview of the views utilized and their generation.

- **RGB Images, RGB** - We directly use sequences of RGB frames present in videos
- **Optical Flow, Flow** - We use the popular TVL1 flow for UCF101 and HMDB51 and Farneback Flow (FF) for Kinetics400. FF is known to perform worse than TVL1-Flow on visual recognition tasks, however, it is quicker to compute. This view mismatch views leads to lesser gains when using Kinetics pre-trained flow weights for UCF101 and HMDB51.
- **Pose Keypoint Heatmaps, PoseHMs** - We use an off-the-shelf keypoint detector [45] and extract confidence heatmaps for each keypoint. Note that we perform no pre/post-processing on the results and directly use this as input to our model. The input modality is inherently very noisy, however, we still observe improved performance.
- **Human Segmentation Masks, SegMasks** - Similar to the above, we use an off-the-shelf semantic segmentation network [45] and extract confidence scores for human segmentation. Similar to pose keypoint heatmaps, this input modality is inherently very noisy.

Fig. 7 shows examples of different views. Note the prevalence of noise in a few samples, specially SegMasks. There are multiple other instances where PoseHMs are noisy as we're unable to even localize the actor accurately.

A.4. Implementation Details

We choose to use a 3D-ResNet similar to [13] as the encoder $f(.)$. Following [10] we only expand the convolutional kernels present in the last two residual blocks to be 3D ones. We used 3D-ResNet18 for our experiments, denoted as ResNet18. We use a weak aggregation function $g(.)$ in order to learn a strong encoder $f(.)$. Specifically, we use a one-layer Convolutional Gated Recurrent Unit (ConvGRU) with kernel size (1, 1) as $g(.)$. The weights are shared amongst all spatial positions in the feature map. This design allows the aggregation function to propagate features in the temporal axis. A dropout [38] with $p = 0.1$ is used when computing the hidden state at each time step. A shallow two-layer perceptron is used as the predictive function $\phi(.)$. Recall $z'_j = Pool(z_j) where z'_j \in R_D$. We utilize stacked max pool layers as $Pool(.)$.

To construct blocks to pass to the network, we uniformly choose one out of every 3 frames. We then group these into 8 blocks containing 5 frames each. Since the videos we use are usually 30fps, each block roughly covers 0.5 seconds worth of content. The predictive task we design involves predicting the last three blocks using the first five. Therefore, we effectively predict the next 1.5 seconds based on the first 2.5 seconds.

We perform random cropping, random horizontal flipping, random greying, and color jittering to perform data augmentation in the case of images. For optical flow, we only perform random cropping on the image. As discussed earlier, Keypoint Heatmaps and Segmentation Confidence Masks are modelled as images, therefore we perform random cropping and horizontal flipping in their case. Note that random cropping and flipping is applied for the entire block in a consistent way. Random greying and color jittering are applied in a frame-wise manner to prevent the network from learning low-level features such as optical flow. Therefore, each video block may contain both colored and grey-scale image with different contrast.

All individual view-specific models are trained independently using only $\mathcal{L}_{cpc}$. After which we proceed to train all view-specific models simultaneously using $\mathcal{L}_{cocon}$. All models are trained end-to-end using Adam [21] optimizer with an initial learning rate 10^{-3} and weight decay 10^{-5}. Learning rate is decayed to 10^{-4} when validation loss plateaus. A batch size of 16 samples per GPU is used, and our experiments use 4 GPUs. We train models on UCF101 for 100 epochs using $\mathcal{L}_{cpc}$, after which they are collectively trained together for 60 epochs using $\mathcal{L}_{cocon}$. We repeat the same for Kinetics400 with reduced epochs. We train models on Kinetics400 for 80 epochs using $\mathcal{L}_{cpc}$ and further for 40 epochs using $\mathcal{L}_{cocon}$.

The learned representations are evaluated by their performance on the downstream task of action classification. We follow the evaluation practice from recent works and use the weights learned through our self-supervised framework as initialization for supervised learning. The whole setup is then fine-tuned end-to-end using class label supervision. We finally report the fine-tuned accuracies on UCF101 and HMDB51. We use the learned composite function $F(.)$ to generate context representations for video blocks. The context feature is further passed through a spatial pooling layer followed by a fully-connected layer and a multi-way softmax for action classification. We use dropout with $p = 0.7$ for classification. The models are fine-tuned for 100 epochs with learning rate decreasing at different steps. During inference, video clips from the validation set are densely sampled from an input video and cut into blocks with half-length overlapping. The softmax probabilities are averaged to give the final classification result.

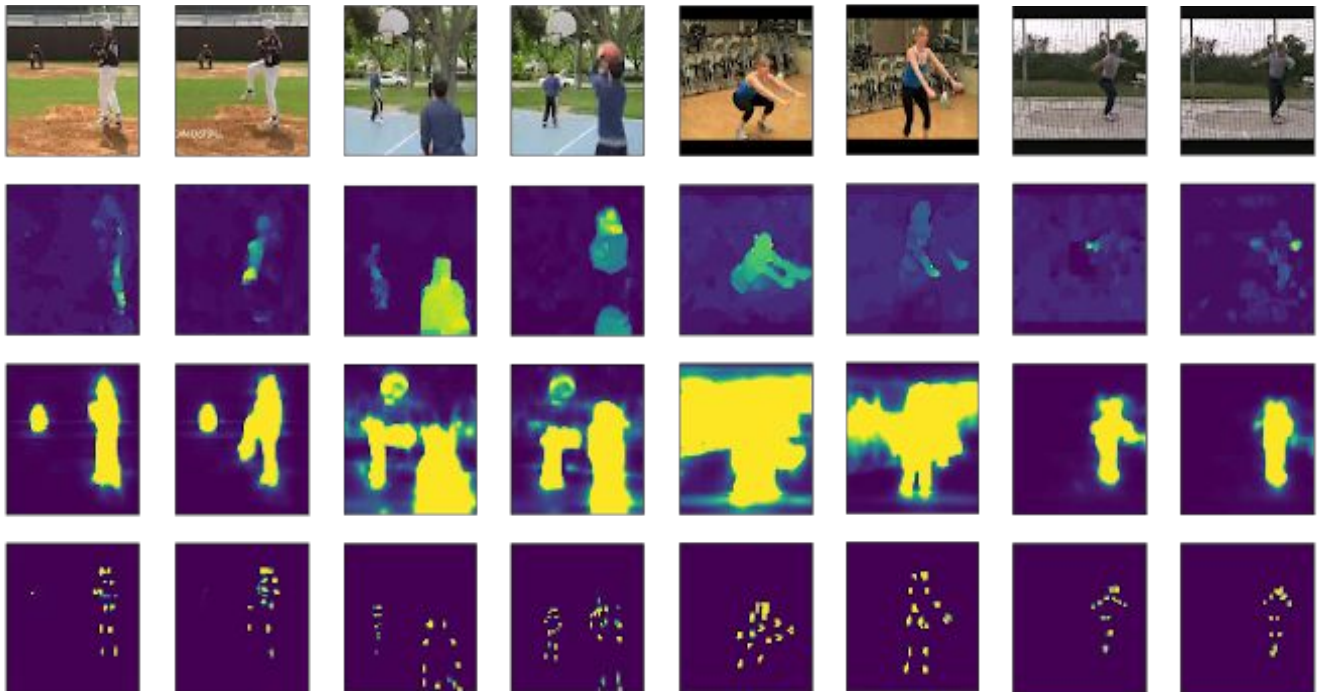


Figure 7: Examples for each view. From top to bottom - RGB, Flow, SegMasks and Poses.

B. Additional Results

We motivate CoCon arguing about the benefits of preserving similarities across view-specific feature spaces. We observe respecting structure across views results in emergence of higher-order semantics without additional supervision e.g. sensible class relationships and good feature representations. We go over different results in the following sections.

B.1. t-SNE Visualization

We explore t-SNE visualizations of our learned representations on the 1^{st} test split of UCF101 extracted using $F(.)$. Our model is trained on the corresponding train split to ensure we're testing out of sample quality. For clarity, only 21 action classes are displayed. We loosely order the action classes according to their relationships. Classes having similar colors are semantically similar. Results are displayed in Fig 8. Even though we operate in a self-supervised setting, our approach is able to uncover deeper semantic features allowing us to uncover inter-class relationships. We can see a much more concise and consistent clustering in CoCon compared to CPC. We also observe the distinct improvement in the compactness of the clusters as we increase the number of views.

B.2. Inter-Class Relationships

In order to study the manifold consistency across different views, we look at relationships between classes by inferring their similarities through the learned features. We compare cosine similarities across video clips from different classes. We then compute the most similar five classes for each action. We repeat the process for all views and look at the consistency of the results. Ideally, semantically similar classes should be consistent across all views, assuming the views reasonably capture the essence of the task we're interested in.

We observe that CoCon leads to much higher consistency across different views. Specifically, we see 41 classes which have at least four out of five top-classes consistent in all views; as opposed to 10 classes in CPC. Similar patterns are seen when we consider other thresholds. In order to confirm that the nearest classes are actually sensible, we mention the most-similar classes for a few action classes.

We can see that the nearest actions generated are semantically related to the original actions. In the cases of PlayingCello, we encounter a cluster of categories involving playing instruments. Similarly for BasketBall, we can see emergence of sports-based relationships even though there is no visual commonality between categories. We also see a few seemingly unrelated classes as well, e.g., BoxingPunchingBag and YoYo; SalsaSpin and WalkingWithDog.

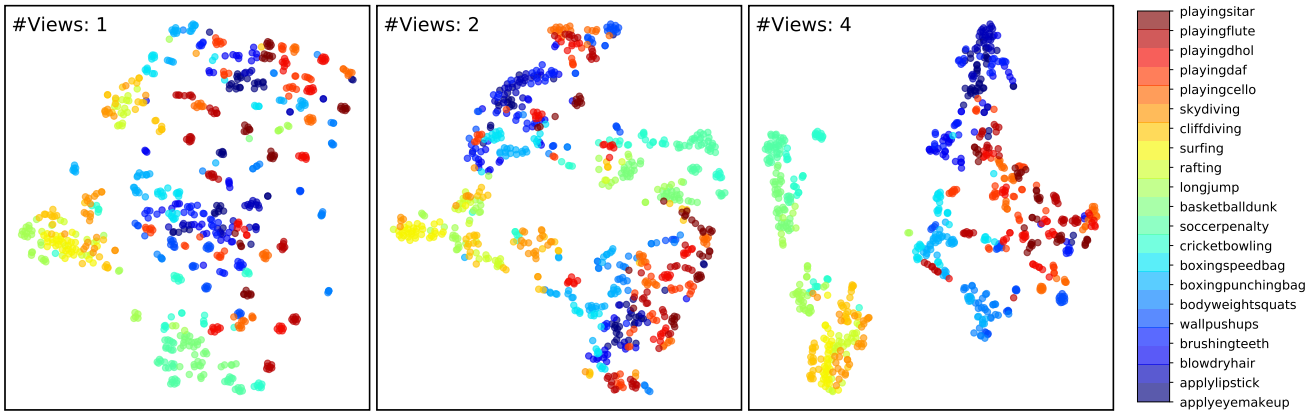


Figure 8: Emergence of relationships between different actions using CoCon with varying number of views. Note that CoCon becomes the same as CPC when $\#views = 1$

Action Class	Nearest Classes	
	CoCon	CPC
skiing	surfing, skijet	surfing
playingcello	playingsitar, playingtabla, playingdhol	N/A
jumpingjack	jumprope, pullups, bodyweightsquats, cleanandjerk	N/A
basketball	baseballpitch, cricketshot, fieldhockey, cricketbowling	N/A
hammerthrow	baseballpitch, throwdiscus, shotput	N/A
wallpushups	writingonboard, bodyweightsquats	N/A
brushingteeth	applylipstick, applyeyemakeup, shavingbeard, haircut	applylipstick

Table 7: Closest semantic classes provided by different models. CPC has very few consistent nearest classes across views. While views trained using CoCon show consistent results across views, leading to sensible inter-class relationships

A deeper inspection into the samples is required to comment whether this truly makes sense. It is worth noting that as these nearest action classes are mostly consistent across different views, our approach cannot cheat to generate them i.e. it cannot look at 'background crowd' or 'green field' and infer that the video clip is related to sports. Since views such as Optical-Flow, SegMasks and KeypointHeatmap do not have such information and are much low-dimensional.

C. Action Alignment

An interesting side-effect of improved representations for actions is the possibility of performing loose action alignment. Even though we only use self-supervision, CoCon embeddings are able to capture relevant semantics through our multi-view approach allowing loose alignment between videos. To compute this soft alignment, we divide each video into 18 blocks and compute block-level features z'. We then utilize relative cosine similarities to infer associations between the videos. We smoothen the heatmap in order to make it visually appealing. Figure 5 shows alignment between different videos. Figure 9 highlights a few examples when we perform alignment between same videos. Notice the periodicity implicitly present in these actions captured through the heatmap.

C.1. Cosine similarity

This section highlights the ability of representation generated through CoCon to capture meaningful semantics going beyond low-level features. We look at cosine similarity distributions of video representation from UCF101. We extract one context representation for each video and pool it into a vector. We then compute the cosine similarity for each pair of video features across the unseen UCF101 test set. The cosine distance is summarized by a histogram, where the 'blue' histogram represents the score distribution for positives i.e. videos belonging to the same class; and the 'orange' one shows the distribution for negatives i.e. videos from different classes.

C.2. Nearest Neighbors

We utilize CoCon to perform video retrieval for different query videos. Note that CoCon is able to look past purely visual background features and focus on actions even though it only used RGB inputs. For example, we see that we are able to retrieve close neighbors for BenchPress, even

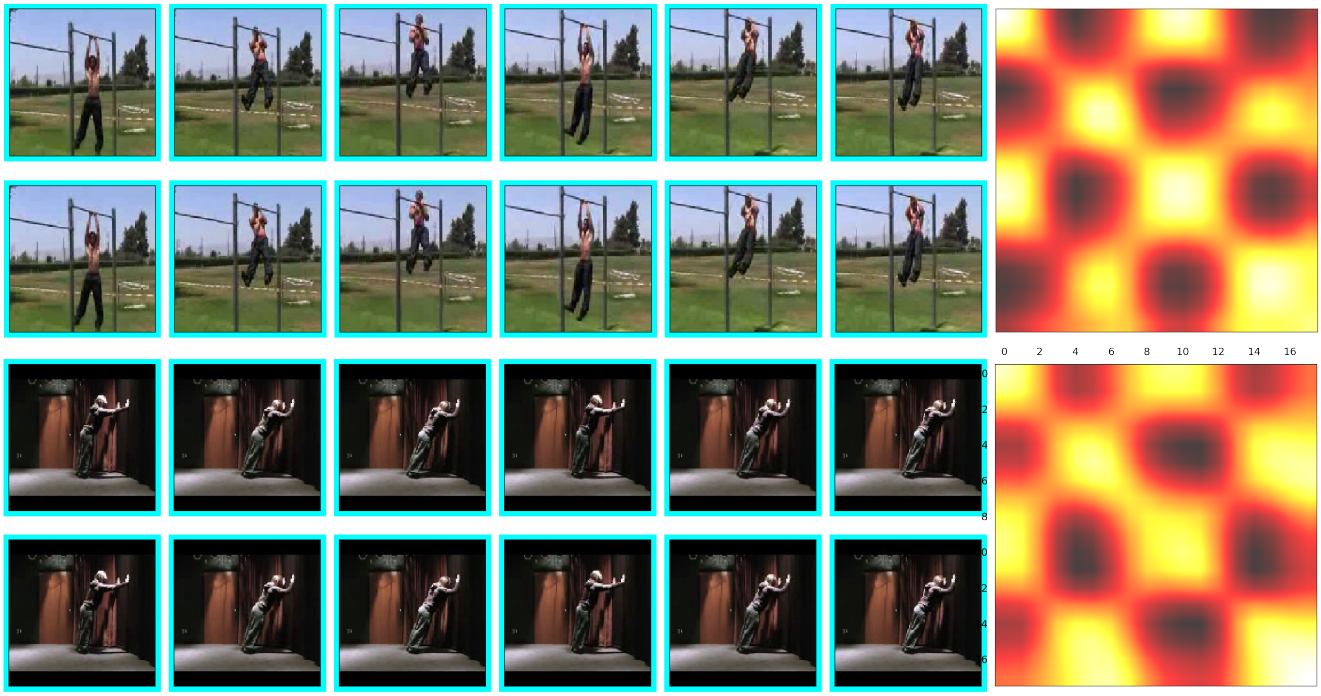
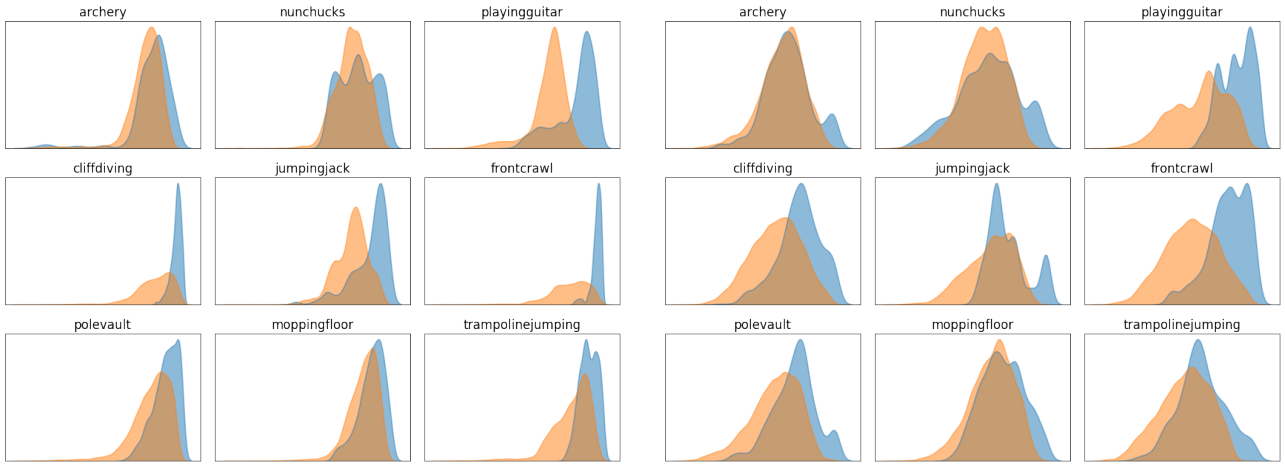


Figure 9: Soft Alignment of actions between the same video instances. The heat-map represents the relative similarities between blocks at various timesteps. Notice periodic patterns in the actions.



(a) Generated by CoCon

(b) Generated by CPC

Figure 10: Distributions of cosine-similarity scores between representations of videos from the same (blue) and other classes (red).

though it is very visually different with varying poses. For the IceDancing sample, even though it incorrectly considers onbe video where the person is running, we can still see similarities between the underlying actions in the videos. Similar results can be seen in other examples as well. This hints towards the fact that CoCon representation are able to capture action semantics even while using RGB views.

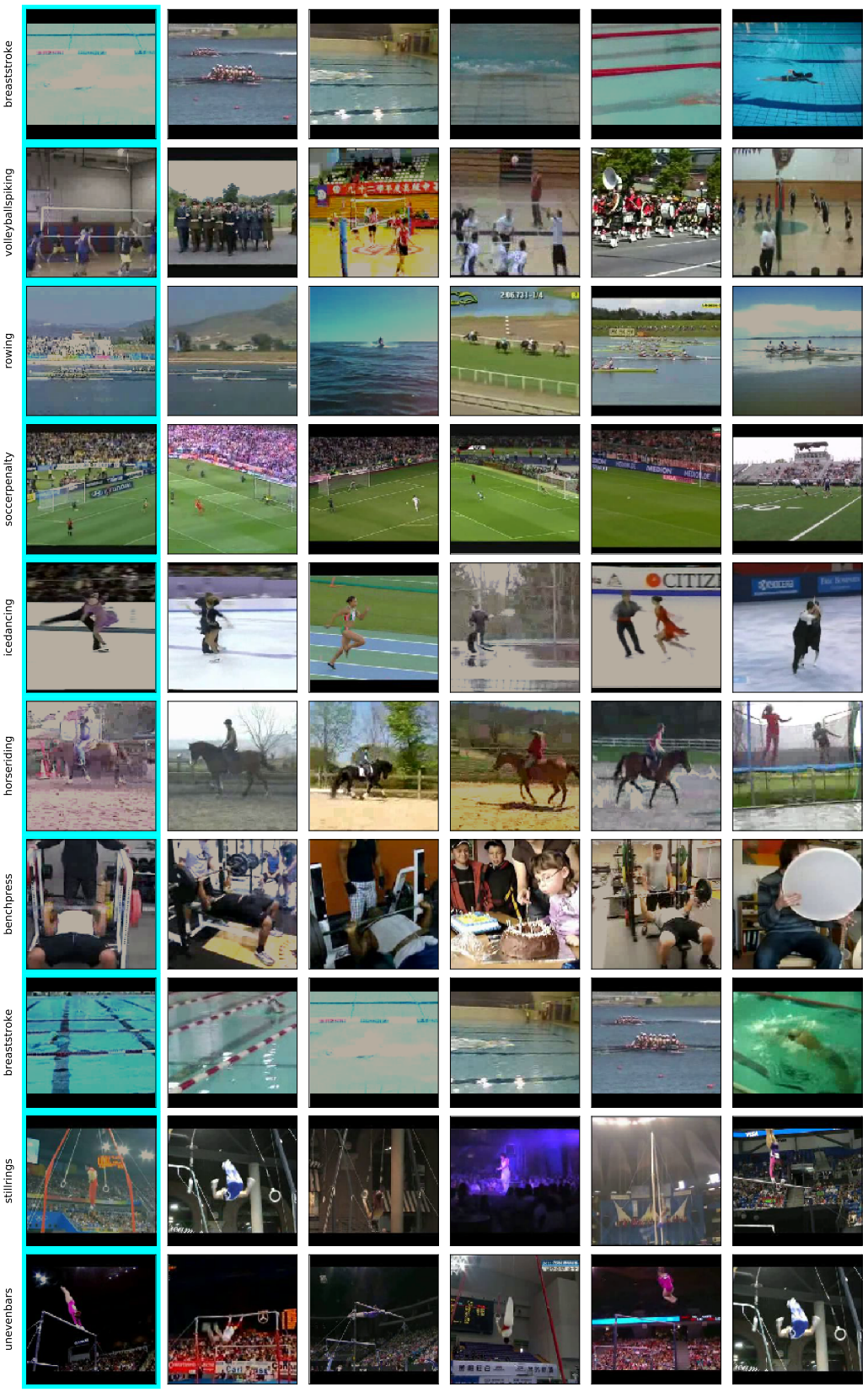


Figure 11: Nearest neighbors computed using RGB representations. Query video is highlighted on the left with Aqua Blue.