

# Temporal Cycle-Consistency Learning

## Supplementary Material

Debidatta Dwibedi<sup>1</sup>, Yusuf Aytar<sup>2</sup>, Jonathan Tompson<sup>1</sup>, Pierre Sermanet<sup>1</sup>, and Andrew Zisserman<sup>2</sup>

<sup>1</sup>Google Brain <sup>2</sup>Deepmind

{debidatta, yusufaytar, tompson, sermanet, zisserman}@google.com

### 1. Synchronous Playback

One direct application of being able to align videos is that we can play multiple videos with the pace of a reference video. The task of synchronizing videos manually can be very time-consuming, often requiring multiple cuts and frame rate changes. We show how we can use self-supervised learning to reduce the effort required to synchronize videos. Please find the `supp.html` file in the supplementary folder which has links to the results videos. They can also be accessed directly through the `videos` folder. We produce these videos by first embedding all frames in all the videos using our trained encoder. We choose a reference video with whose pace we want to play all the other videos. For every other video, we choose the matching frame in the whole video using dynamic time warping. This is done to enforce temporal constraints on a per-frame basis. No other post processing steps are used.

### 2. Sound Transfer

We can also transfer other meta-data or modalities (that are synchronized with the frames in a video) only on the basis of the visual similarity. We showcase an example of such a transfer by using sound, which is arguably the most commonly available synchronized modality. Please find examples of sound transfer in the teaser video (1:38 onwards). In order to transfer the sound, we look up the nearest neighbor frames in a video that has sound. For each frame in the target video, we copy over the block of sound associated with the nearest neighbor frame. We concatenate these blocks of sound. Note, how the sound changes as the liquid flows into the container. This presents further evidence the embeddings are able to capture progress in a particular task. We use multiple frames in the sound synthesis. We average the embeddings for the multiple frames and concatenate the corresponding sounds to produce the sound blocks. We do this so that edge artifacts are reduced when we synthesize the sound for the whole video. No other post processing steps are used.

### 3. t-SNE Visualization

We also present examples of t-SNE[1] visualization of the embeddings in the teaser video (0:30 and 1:05) and Figure 1. For each action, we show trajectories of 4 videos in the embedding space. The borders are color-coded differently for each video. We sample two random time-steps for each video and show the corresponding frame and embedding location. Frames with the same border color are sampled from different time-steps in the same video. The visualization indicates how the embeddings change as an action is carried out. Additionally, they also highlight how corresponding frames from different videos in the validation set are closer to each other in the learned embedding space as compared to non-corresponding frames. This structure in the embedding space, induced by the self-supervised objectives during training, is why we are able to align different videos and perform fine-grained retrieval by simply using nearest neighbors.

### 4. Fine-grained Retrieval

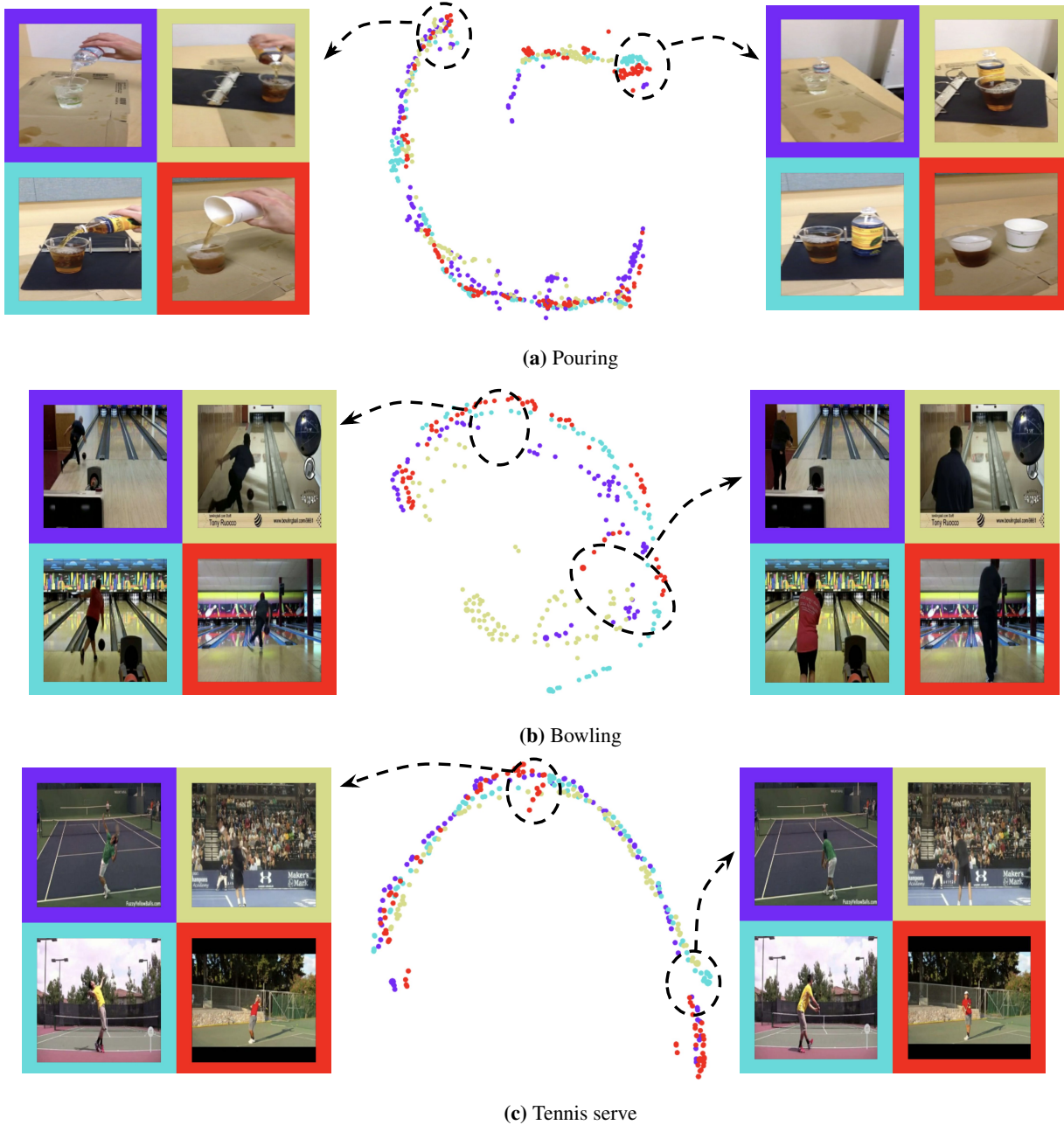
We provide additional results for fine-grained retrieval in Figure 2.

### 5. Data Augmentation

We use data augmentation during training. We randomly flip an entire video horizontally. We perturb brightness by adding a random number between  $-32$  and  $32$  to the raw pixels. We change contrast by a random factor sampled uniformly between  $0.5$  and  $1.5$ . All training algorithms have the same data augmentation pipeline.

### 6. Architecture Details

We describe the complete architecture of our encoder  $\phi$  in Table 1. It is composed of 2 parts: *Base Network* and *Embedder Network*. The *Base Network* acts on individual frames to extract convolutional features from them. Depending on the chosen base network,  $c_4$  ( $c$  in Table 1 of



**Figure 1: t-SNE Visualization of Embeddings.**

the main paper) is either 1024 or 512. The *Embedder Network* collects convolutional features of each frame and its context window and embeds them into a single 128 dimensional vector. All the different training algorithms in our experiments are applied on top of these 128 dimensional vectors. While initially we were experimenting with larger values of  $k$ , we found even with  $k = 2$  we can get good performance on both datasets. The gap between the two frames is approximately 0.75 seconds (15 frames at 20 fps) for the

Penn Action dataset and 0.3 seconds (9 frames at 30 fps) for the Pouring dataset.

## 7. Hyperparameters

In Table 2, we tabulate the list of values of the hyperparameters.



**Figure 2: Fine-grained retrieval.** Embeddings learned by self-supervised methods robustly capture fine-grained aspects of an action.



Model	Layer	Output Size	Pre-trained ResNet-50	VGG-M Like (Scratch)
Base Network	conv1	$112 \times 112 \times c_1$	$7 \times 7, 64, \text{stride } 2$	
			$3 \times 3 \text{ max pool, stride } 2$	
	conv2_x	$56 \times 56 \times c_2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 1$
	conv3_x	$28 \times 28 \times c_3$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 1$
Embedder Network	conv4_x	$14 \times 14 \times c_4$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 1$
	Temporal Stacking	$k \times 14 \times 14 \times c_4$	Stack $k$ context frame features in time axis	
	conv5_x	$k \times 14 \times 14 \times 512$	$\begin{bmatrix} 3 \times 3 \times 3, 512 \\ 3 \times 3 \times 3, 512 \end{bmatrix} \times 1$	
	Spatio-temporal Pooling	512	Global 3D Max-Pool	
	fc6_x	512	$\begin{bmatrix} 512 \\ 512 \end{bmatrix} \times 1$	
	Embedding	128	128	

**Table 1:** Architectures used in our experiments. The network produces an embedding for each frame (and its context window).  $c_i$  depends on the choice of the base network. Inside the square brackets, the parameters in the form of: (1)  $[n \times n, c]$  refers to 2D Convolution filter size and number of channels respectively (2)  $[n \times n \times n, c]$  refers to 3D Convolution filter size and number of channels respectively (3)  $[c]$  refers to channels in a fully-connected layer. Downsampling in ResNet-50 is done using convolutions with stride 2, while in VGG-M models we use MaxPool with stride 2 for downsampling.

Hyperparameter	Value
Batch Size	4
Number of frames	20
Optimizer	ADAM
Learning Rate	$1.0 \times 10^{-4}$
Weight Decay	$1.0 \times 10^{-5}$
Alignment Variance $\lambda$	0.001
TCN Positive Window Size	5
Frames per second	20 (Penn Action), 30 (Pouring)
SaL Classifier FC Sizes	128, 64
SaL Fraction Shuffled	0.75

**Table 2:** List of hyperparameters used

## References

- [1] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008. 1