Learning Conditional Space-Time Prompt Distributions for Video Class-Incremental Learning

Supplementary Material

6. Image Continual Learning

Before the emergence of prompt-based methods, continual learning approaches in the image domain were broadly categorized into three types [40, 58]: regularization-based methods, exemplar-based methods, and architecture-based methods.

Regularization-based methods [1, 8, 15, 22, 27, 32, 42, 64] constrain the optimization of network parameters to preserve the important information of learned tasks, with knowledge distillation [17] being the most popular technique utilized. However, these methods cannot achieve satisfactory performance under challenging settings or with complex datasets [39].

Rehearsal-based methods [3, 4, 36, 49, 53, 56, 69] typically mitigate catastrophic forgetting by storing a small set of representative exemplars [3, 36, 49, 56] or generating synthetic samples from previous classes [23, 41, 43, 53]. However, methods relying on raw exemplars suffer performance drops with limited buffer sizes and pose data privacy concerns [67], while those using synthetic data struggle with significant domain gaps between synthetic and real distributions [41]. Note that although DiffClass [41] also employs a diffusion model, it generates synthetic images and requires fine-tuning Stable Diffusion [50], which is computationally expensive. In contrast, our method generates prompts instead, offering significantly greater computational and parameter efficiency.

Architecture-based methods [13, 20, 51, 52, 62] retain knowledge of old tasks by designing specific components in the architecture or expanding the current feature extractor for new data. Nevertheless, these methods require a substantial number of additional parameters and test-time task identities.

7. Algorithms for CoSTEP

We detail the training and test time algorithms for CoSTEP in Algorithms 1 and 2, respectively.

8. Details of prompt learning

We adhere to a widely recognized prompt tuning protocol [21, 67]. The prompt $\mathbf{P} \in \mathbb{R}^{N_p \times D}$ is trainable, while the ViT remaining fixed. $\mathbf{P}$ is appended to the patch embedding of each frame (or frame grid) before being input into the self-attention layers of ViT.

Consider a single frame/frame grid $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$. Suppose a ViT is represented as $f = f_e \circ f_r$, where f_e is the input embedding layer, and f_r is a stack of self-attention layers. The ViT first reshapes the image $\mathbf{x}$ into a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{L \times (S^2 \times C)}$, with L representing the number of patches, S the patch size, and C the original channel count. Here we assume the first token in $\mathbf{x}_p$ is the `[class]` token. The pre-trained embedding layer f_e then maps the patched image to a patch embedding feature $\mathbf{x}_e = f_e(x_p) \in \mathbb{R}^{L \times D}$. With the ViT frozen, we prepend the trainable prompts $\mathbf{P}$ to the patch embedding feature, resulting in an extended sequence $\mathbf{x}_{pt} = [\mathbf{P}; \mathbf{x}_e] \in \mathbb{R}^{(N_p+L) \times D}$. This sequence is then processed by $f_r(\mathbf{x}_{pt})$ to perform classification tasks. In this manner, f_r applies self-attention across the video frame/frame grid patches and prompts, enabling their interaction.

This process is repeated for all frames/frame grids, aligning with the purple block in Figure 3 on the right side, to compute the video classification loss. Backpropagation is then performed, and $\mathbf{P}$ is updated using a commonly used optimizer, in our case, SGD.

9. Inference with the Diffusion Model

During inference, the diffusion model [18] generates $\mathbf{P}_0$ by iteratively denoising over T steps, starting from random Gaussian noise $\mathbf{P}_T$. Reversing the forward step, $q(\mathbf{P}_{t-1}|\mathbf{P}_t)$, is computationally impractical, so the model instead focuses on maximizing the variational lower bound using parameterized Gaussian transitions, $p_\theta(\mathbf{P}_{t-1}|\mathbf{P}_t, \hat{\mathbf{E}})$, where θ are the model's parameters. The reverse process starts from random noise, $\mathbf{P}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and progresses through each step as:

$$p_\theta(\mathbf{P}_{0:T}|\hat{\mathbf{E}}) = p_\theta(\mathbf{P}_T|\hat{\mathbf{E}}) \prod_{t=1}^{T} p_\theta(\mathbf{P}_{t-1}|\mathbf{P}_t, \hat{\mathbf{E}}), \quad (6)$$

where

$$p_\theta(\mathbf{P}_{t-1}|\mathbf{P}_t, \hat{\mathbf{E}}) = \mathcal{N}(\mathbf{P}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{P}_t, \hat{\mathbf{E}}, t), \sigma_t^2 \boldsymbol{I}), \quad (7)$$

$$\boldsymbol{\mu}_\theta(\mathbf{P}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{P}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{P}_t, \hat{\mathbf{E}}, t) \right). \quad (8)$$

Therefore, to sample from $p_\theta(\mathbf{P}_{t-1}|\mathbf{P}_t)$, one can perform the following:

Algorithm 1 CoSTEP at training time

Require: Tasks $\{\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^K\}$, task-specific prompts $\{\mathbf{P}^k\}_{k=1}^K$, frozen pre-trained backbone, classifier parameters ϕ, diffusion parameters θ, diffusion steps T, total number of training epochs E, number of epochs for the first training stage E_1

for $k := 1 \to K$ **do**
  for $e := 1 \to E$ **do**
    for batch b in $\mathcal{T}^K$ **do**
      if $e \leq E_1$ **then**
        Calculate loss $\ell_{\text{stage-1}}$ (Section 3.3)
        Update $\mathbf{P}^k$ and ϕ
      else
        Extract video embeddings $\hat{\mathbf{E}}$ for use as conditions
        $\mathbf{P}_0^k = \mathbf{P}^k$
        $t \sim \text{Uniform}(1, \ldots, T)$
        Calculate $\mathbf{P}_t^k$ (Equation 3)
        Calculate diffusion loss $\ell_{\text{diffusion}}$ (Equation 4)
        Calculate the final loss $\ell_{\text{stage-2}}$ (Equation 3.4)
        Update θ and ϕ
      end if
    end for
  end for
end for

Algorithm 2 CoSTEP at test time

Require: frozen pre-trained backbone, trained classifier g_ϕ, trained diffusion network ϵ_θ, diffusion steps T

Extract video embeddings $\hat{\mathbf{E}}$ for use as conditions
$\mathbf{P}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t := T \to 1$ **do**
  Calculate $\mathbf{P}_{t-1}$ (Equation 9)
end for
Use $\mathbf{P}_0$ as a prompt for the frozen pre-trained backbone to calculate the final prediction

$$\begin{aligned}\mathbf{P}_{t-1} &= \boldsymbol{\mu}_\theta(\mathbf{P}_t, \hat{\mathbf{E}}, t) + \sigma_t \boldsymbol{\epsilon} \\ &= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{P}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{P}_t, \hat{\mathbf{E}}, t)\right) + \sigma_t \boldsymbol{\epsilon}\end{aligned} \tag{9}$$

10. Evaluation Metrics

We evaluate the model on all seen classes and report the average accuracy (Acc) and backward forgetting (BWF) on all K observed tasks:

$$\text{Acc} = \frac{1}{K}\sum_{i=1}^{K} A_{K,i}, \tag{10}$$

$$\text{BWF} = \frac{1}{K-1}\sum_{i=1}^{K-1} A_{i,i} - A_{K,i}, \tag{11}$$

where $A_{i,j}$ is the accuracy of task j after training task i.

11. More implementation details

For training CoSTEP's diffusion model, we use an AdamW optimizer with a linear warm-up scheme. Other trainable modules use an SGD optimizer, and both optimizers operate without decay. The initial learning rates are set at 0.001 for the diffusion model and 0.01 for the other components. All input video frames are resized to 224×224 and normalized to the range $[0, 1]$. The diffusion model undergoes $T = 50$ diffusion steps. All models are implemented in PyTorch.

The U-Net in our diffusion model consists of 3 layers, each layer comprising two residual blocks. Each block includes two convolutions, followed by a group norm and Mish activation, and is accompanied by either a downsampling or upsampling operation. Position embeddings for different prompts are integrated using a fully-connected layer, which is added to the output of the first convolution.

12. More Experiments

The length of prompts. The capacity of each learnable prompt is determined by its length. Figure 7 demonstrates the variation in average accuracy on UCF101 (10 tasks) as the length of prompts changes. It's clear that prompts that are too short invariably lead to poorer outcomes, whereas prompts that are too long can result in underfitting. Based on the curve observed, we choose a prompt length of 6.

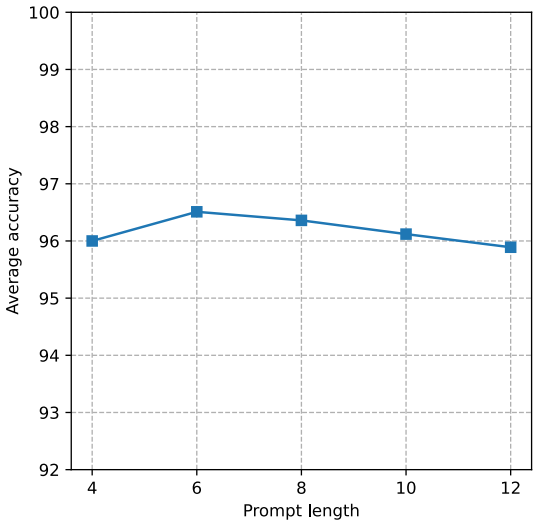

Figure 7. The effect of the prompt length on UCF101 (10 tasks).

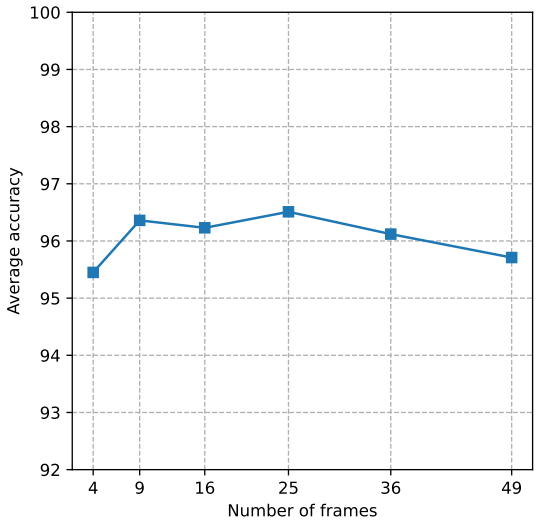

Figure 8. The effect of the number of frames in a frame grid on UCF101 (10 tasks).

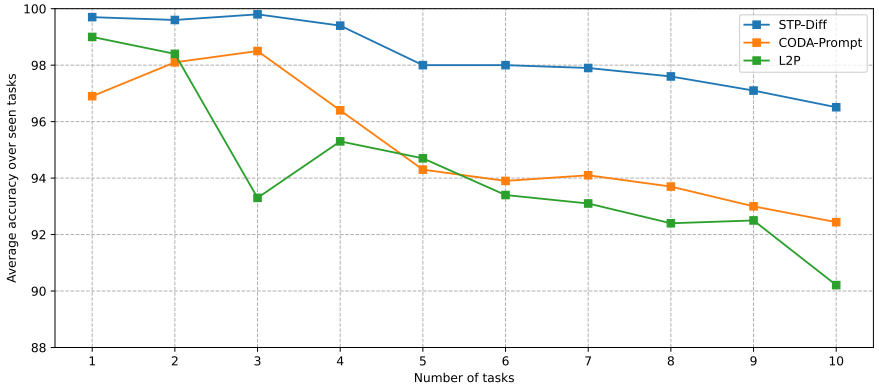

Figure 9. Average accuracy per task on UCF101 (10 tasks).

Table 7. Average accuracy with different numbers Monte Carlo samples on UCF101 (10 tasks).

Monte Carlo Samples			
1	5	10	20
96.51	96.80	97.02	96.18

Number of frames in a frame grid. We scale down the original frames to create frame grids, thereby controlling the number of frames observed by the model in a single frame grid. Figure 8 illustrates how varying the number of frames in a frame grid impacts performance on UCF101 (10 tasks). Increasing the frame count enhances temporal resolution but at the cost of diminishing the detail in each frame. In contrast, a smaller frame count preserves more information from each original frame but results in lower temporal resolution. We set the number of frames as 25 according to the curve.

Number of Monte Carlo samples. Typically, using more Monte Carlo samples leads to a more accurate approximation of the log-likelihood in Equation 1 and better gener-

Table 8. Average accuracy with different numbers of stored video representations/raw videos on UCF101 (10 tasks).

	Storage type	5 / class	10 / class	20 / class
L2P [67]	raw video	84.43	85.37	89.61
CODA-Prompt [54]	raw video	83.29	88.39	92.07
CoSTEP (Ours)	video representation	**90.26**	**94.49**	**96.51**

Table 9. Average accuracy of frame grids vs. a temporal transformer for temporal modeling on Something-Something v2.

	SSv2 10 $\times$ 9 stages	SSv2 5 $\times$ 18 stages
Temporal Transformer [61]	37.73	34.21
CoSTEP	**41.44**	**36.60**

alization to new data. We increased the number of Monte Carlo samples for CoSTEP on UCF101 (10 tasks) and report the results in Table 7. The results show a improvement in performance.

Number of stored video representations. As described in Section 3.4, to prevent forgetting during the training of the diffusion model, we store m video representations from previous tasks. These representations are used along with current task data to calculate $\ell_{\text{diffusion}}$ according to Equation 4. Unlike traditional rehearsal methods that store raw videos or selected frames, our approach stores only one representation vector per video, greatly improving memory efficiency and reducing privacy risks since these vectors take up less space and are less revealing than raw video data.

Table 8 shows comparisons for different values of m and contrasts the performance of our CoSTEP with L2P and CODA-Prompt under the same m. Notably, CoSTEP stores just one video representation per video, whereas L2P and CODA-Prompt store an entire raw video per instance, as they do not utilize global video representations in their methods. Despite the disadvantage for our CoSTEP, which relies on representation vectors from a frozen CLIP [48] compared to competitors using raw videos, CoSTEP still achieves significantly higher average accuracy than both L2P and CODA-Prompt with the same value of m. This highlights CoSTEP's efficiency and effectiveness, showing it can outperform while only storing representation vectors that require much less memory than raw videos or frames.

Frame grids vs. temporal transformer. In table 9, we compare the frame grid approach with a temporal transformer applied to all frame features for temporal modeling on the Something-Something v2. The architecture of the temporal transformer used in this experiment is identical to that used in PIVOT [61]. Using frame grids performs better, likely because: 1) the temporal transformer uses self-attention on high-level frame features, missing local patch interactions across frames, and 2) the temporal transformer adds more parameters, increasing its susceptibility to catastrophic forgetting.

Per-task analysis. Figure 9 shows the average accuracy across tasks after training each task for CoSTEP, L2P, and CODA-Prompt. CoSTEP consistently records the highest average accuracy after each task and shows the least decline in performance as more tasks are added. This highlights CoSTEP's enhanced learning capacity and adaptability, as well as its ability to retain previous knowledge effectively, thanks to its use of a diffusion model.

Efficiency in training and inference. The U-Net we use is small, as it reconstructs prompt vectors instead of raw images. It's only slightly larger than the combination of prompt pool and attention vectors used by CODA-Prompt [54] and significantly smaller than the temporal transformer and prompt pool in PIVOT [61]. Therefore, a forward pass through our U-Net is not slower than those in CODA-Prompt and PIVOT.

We report the average training and inference time per batch (batch size 50) in Table 10, calculated across all batches. The training time for CoSTEP is comparable to CODA-Prompt, L2P, and PIVOT because, during training, the diffusion model randomly selects a single timestep and computes the diffusion loss, requiring only one additional forward pass through the small U-Net per sample. During inference, CoSTEP requires more time than the others, as it performs T (in our experiments $T = 50$) reverse

Table 10. Average training/inference time per batch (seconds).

	CoSTEP	CODA-Prompt [54]	L2P [67]	PIVOT [61]
Training	0.41	0.42	0.40	0.57
Inference	0.67	0.37	0.36	0.53

Table 11. Accuracy and interence time with different diffusion steps on HMDB51 (5 tasks).

# Steps	10	25	50	100	150	200
Acc (↑)	52.72	59.83	61.70	61.72	61.08	**61.74**
Inference time (↓)	**0.50**	0.55	0.67	0.77	0.83	0.88

diffusion steps to generate a prompt.

Number of diffusion steps. Table 11 presents the inference time and average accuracy for different diffusion steps on HMDB51 (5 tasks). Using fewer than 50 steps degrades performance, as the diffusion model struggles to learn a valid prompt distribution. Beyond 50 steps, accuracy improvements are marginal, while inference time increases substantially.

13. More visualizations

Visualization of video representations. Figure 10 shows a t-SNE visualization of video representations from `CoSTEP` and `CODA-Prompt` on the UCF101 (10 tasks) test set after completing training on all tasks. The features generated using `CoSTEP`'s space-time prompts are better clustered and more discriminative compared to those generated with `CODA-Prompt`. This further explains why `CoSTEP` achieves better performance: it generates prompts that more effectively cluster video features of the same class.

14. Discussions

Limitations. CoSTEP enables temporal reasoning by incorporating a frame grid, enhancing pre-trained image models' capabilities to model temporal relationships between frames. It is simple to implement, easily adaptable to prompt learning frameworks, and introduces no additional parameters, minimizing the risk of forgetting. Ideally, this approach would benefit from selecting the most representative frames or patches associated with an action to construct the frame grid. However, CoSTEP currently uses a uniform sampling to select frames, which may overlook important temporal patterns. Additionally, scaling down frames to create the grid may lead to reduced resolution and loss of detail. While addressing these issues is not the primary focus of this paper, future work could explore improved methods for selecting representative frames or patches and mitigating detail loss.

Societal Impacts. Although CoSTEP stores only minimal video features from old tasks, there is still a possibility that it retains some information about the original video. Therefore, CoSTEP may not be suitable in scenarios where privacy concerns are extremely stringent.

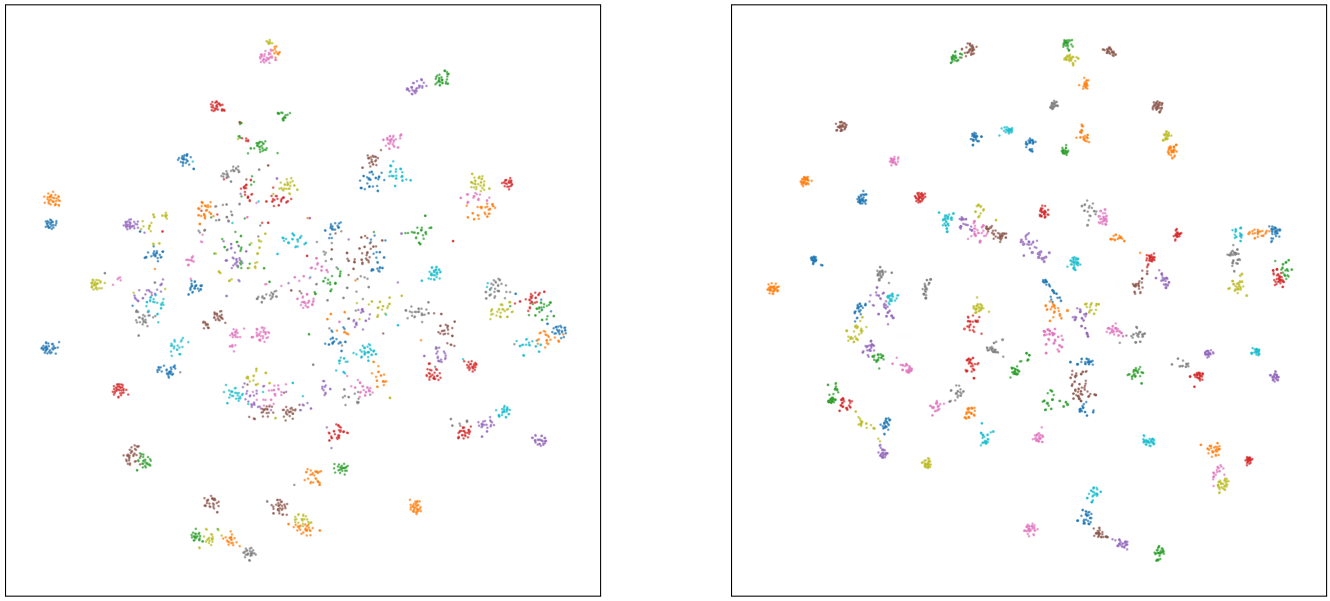

Figure 10. t-SNE visualization of video representations produced by CODA-Prompt (left) and CoSTEP (right) on UCF101 (10 tasks). Each point represents a D_e-dimensional feature, with different colors indicating distinct classes.