COMPRESSED CP-CONV: CONVOLUTIONAL LEARNING FROM FACTORIZED HIGH-DIMENSIONAL INPUTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Convolutional neural networks conventionally operate on explicitly materialized tensors, making high-resolution and high-dimensional inputs increasingly costly or infeasible to process. We introduce *CP-Conv*, a convolutional operator that applies an arbitrary dense, trainable kernel directly to inputs represented in Canonical Polyadic (CP) form, without reconstructing the dense tensor or factorizing the convolutional kernel. For the tensor represented by the supplied CP factors, CP-Conv is algebraically equivalent to dense convolution and therefore introduces no additional approximation of the operator. Because its output is generally dense, we further introduce *Compressed CP-Conv* (CCP-Conv), which fuses convolution with mode-wise output compression and directly produces a compact representation suitable for standard neural-network layers. We derive the computational and storage complexity of both operators and verify numerical agreement with their dense counterparts. Controlled benchmarks yield operator-level speedups of up to 3.9×10^3. Across supervised experiments on synthetic data, UCF101, Ego4D, high-dimensional Fokker–Planck problems, and nonlinear shallow-water dynamics from PDEBench, CCP-Conv closely matches matched dense baselines when dense processing is feasible, while substantially reducing training cost. These experiments include both inputs available directly in CP form and dense data represented through approximate CP decomposition. CCP-Conv also enables convolutional learning on inputs whose implicit dense representation contains up to 2.94×10^{11} values, well beyond the scale that can be processed densely within the available resources. CCP-Conv integrates into a pretrained R3D-18 architecture with comparable classification performance and supports unsupervised representation learning without dense reconstruction. Together, these results show that exploiting structure directly in the input representation can extend dense-kernel convolutional learning to regimes where materializing the input tensor is computationally prohibitive. Reference implementations in JAX and PyTorch, together with reproduction scripts, are available in the supplementary material.

1 INTRODUCTION

High-dimensional data are increasingly common in machine learning, including videos, scientific simulations, probability densities, physical fields, and dynamical systems. Such data are naturally represented as multi-way tensors whose dimensions may correspond to space, time, parameters, or physical variables. A variety of tensor formats, including Canonical Polyadic (CP), Tucker, Tensor-Train, and Tensor-Ring, have been proposed to represent high-dimensional data compactly (Kolda & Bader, 2009; Oseledets, 2011; Zhao et al., 2016). These formats are widely used in scientific computing to represent and manipulate high-dimensional functions (Chinesta et al., 2011; Khoromskij, 2012; Hackbusch, 2014), and have been applied across a broad range of domains (Sidiropoulos et al., 2017). In some scientific settings, factorized representations also arise directly from tensor-structured or separated-representation PDE solvers, rather than being obtained through post-processing of dense data.

Convolution is a powerful tool for extracting local structure from such data, but standard dense convolution typically operates on an explicitly materialized input tensor. As tensor order and resolution increase, both storage and computational costs grow rapidly with the product of the input dimen-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

sions, making high-resolution videos and high-dimensional scientific tensors expensive, and in some cases infeasible, to process directly.

In the machine learning literature, many techniques have been proposed to reduce the computational and memory costs of convolutional learning. These approaches generally address the problem from either the model or the input side. Model-side approaches, including separable and depthwise convolutions, low-rank kernels, and tensorized neural-network layers, impose structural constraints on the trainable operator to reduce its computational cost (Lebedev et al., 2015; Novikov et al., 2015; Kim et al., 2016; Howard et al., 2017; Feichtenhofer, 2020). While effective, such constraints may affect the expressiveness of the resulting operator. Moreover, these approaches primarily reduce the cost or size of the model itself and generally still assume that the input tensor is available in dense form. Input-side approaches instead rely on techniques such as downsampling, patchification, tokenization, temporal subsampling, masking, or compressed-domain representations (Wu et al., 2018; Feichtenhofer et al., 2019; Tong et al., 2022; Dosovitskiy et al., 2021; He et al., 2022). Downsampling, subsampling, and masking reduce the amount of processed information, while patchification and tokenization alter how the input is represented and processed. In scientific machine learning, neural operators such as the Fourier Neural Operator provide another route for learning mappings between physical fields (Li et al., 2021). Our objective is different: we retain a conventional trainable dense convolutional kernel while exploiting an already factorized representation of the input.

This raises a natural question: when the input is already available in a general-purpose low-rank tensor representation, can a conventional trainable convolution act directly on that representation without first reconstructing the dense tensor? Prior work in numerical tensor computation has considered multidimensional convolution directly in low-rank tensor formats (Rakhuba & Oseledets, 2015), but these methods are primarily designed for numerical computation rather than trainable convolutional learning. Conversely, other works have developed learning operations that act directly on factorized tensor inputs (He et al., 2014; Rakhshan & Rabusseau, 2020), but do not provide trainable convolution with an unrestricted dense kernel. In this work, we address this gap by performing convolutional learning directly on a factorized representation of the input tensor.

We focus on inputs represented in Canonical Polyadic (CP) form,

$$X(i_1, \dots, i_d) = \sum_{r=1}^{R} \prod_{\ell=1}^{d} U_\ell^r(i_\ell),$$

where R is the CP rank and U_ℓ^r are the corresponding factor vectors. We introduce *CP-Conv*, which applies an arbitrary dense and trainable convolutional kernel directly to these factors without reconstructing the dense input tensor. CP-Conv does not factorize or constrain the kernel and, when the input is exactly represented in CP form, produces the same output as conventional dense convolution up to numerical precision. More generally, when the CP representation approximates the original input, CP-Conv introduces no additional approximation of the convolutional operator beyond that already present in the input representation. Since the dense high-dimensional tensor need not be materialized during convolution, CP-Conv can substantially reduce memory and computational costs, particularly for high-dimensional inputs and large kernels. Because CP-Conv exploits structure in the input representation rather than imposing structure on the convolutional kernel, it is complementary to model-side kernel factorization methods, and the approaches could in principle be combined.

With an arbitrary dense kernel, however, CP-Conv generally produces a dense output. Consequently, CP-Conv layers cannot be stacked directly while retaining a compact factorized representation without recompressing the intermediate output. To address this limitation, we further introduce *Compressed CP-Conv* (CCP-Conv), which fuses convolution with mode-wise output compression and directly produces a compact dense representation suitable for subsequent neural-network layers. CCP-Conv therefore provides a practical front-end for high-dimensional CP-structured inputs, allowing the resulting compact representation to be processed by standard neural-network architectures.

We evaluate CCP-Conv on synthetic data, UCF101 (Soomro et al., 2012), Ego4D (Grauman et al., 2022), high-dimensional Fokker–Planck problems, and nonlinear shallow-water dynamics from PDEBench (Takamoto et al., 2022). Across these experiments, we consider both inputs available directly in CP form and dense data represented through approximate CP decomposition, thereby evaluating the method under both exact and approximate input representations. On moderate-scale prob-

lems where dense convolution remains feasible, CCP-Conv achieves comparable task performance while substantially reducing computation time. On larger inputs containing up to 10^{11} scalar values per sample, CCP-Conv remains applicable in regimes where dense convolution is computationally impractical or infeasible within the available resources. We further demonstrate its integration into a deep R3D-18 video architecture (Tran et al., 2018), investigate unsupervised representation learning with CCP-Conv autoencoders, and provide additional analyses of CP rank, preprocessing cost, alternative compact input representations, output compression, and multi-layer integration in the appendix. Together, CP-Conv and CCP-Conv provide an exact dense-kernel convolutional formulation for CP-represented inputs and a practical compressed interface to conventional neural architectures.

Our main contributions can be summarized as follows:

- We derive an exact contraction formulation of dense convolution for CP-represented inputs, in which an unrestricted dense kernel is evaluated through successive one-dimensional contractions without reconstructing the dense input tensor.
- We introduce CCP-Conv, which moves mode-wise output compression inside this contraction structure, avoiding materialization of both the dense input and the full dense convolutional output. The resulting compact representation provides a practical interface to standard neural-network architectures, for which we derive the computational and storage complexity.
- We validate the formulation numerically and evaluate CCP-Conv across supervised and unsupervised learning settings, including integration into a pretrained R3D-18, high-resolution video, seven-dimensional Fokker–Planck inputs containing up to 2.94×10^{11} implicit scalar values per sample, and nonlinear shallow-water dynamics from approximately factorized PDEBench simulations, complemented by ablations on rank, output compression, and alternative compact input representations.

2 METHODOLOGY

We consider high-dimensional inputs available in Canonical Polyadic (CP) form. Let $X_{\mathrm{CP}} \in \mathbb{R}^{N_1\times N_2\times\ldots\times N_d}$ denote the tensor implicitly represented by a set of CP factors. Importantly, X_{CP} does not need to be explicitly materialized as a dense tensor. In many applications, such as tensor-based representations of high-dimensional functions or probability distributions, the CP factors are directly available and constitute the input representation itself. For a CP representation of rank R, the tensor represented by these factors can be written as,

$$X_{\mathrm{CP}}(i_1, i_2, \ldots, i_d) = \sum_{r=1}^{R} \prod_{\ell=1}^{d} U_\ell^r(i_\ell), \tag{1}$$

where $U_\ell^r \in \mathbb{R}^{N_\ell}$ denotes the factor vector associated with mode ℓ and rank component r. In other settings, such as dense image or video data, these factors may instead be obtained by computing a finite-rank CP approximation of an original dense sample ($X_{\mathrm{CP}} \approx X_{\mathrm{dense}}$). However, this approximation is not intrinsic to CP-Conv. Let $K \in \mathbb{R}^{k_1\times k_2\times\ldots\times k_d}$ be a dense convolutional kernel. If X_{CP} were explicitly materialized, a valid convolution would produce an output $Y \in \mathbb{R}^{M_1\times M_2\times\ldots\times M_d}$, where $M_\ell = N_\ell - k_\ell + 1$. At each output position, this convolution would be defined as,

$$Y(i_1, i_2, \ldots, i_d) = \sum_{j_1=0}^{k_1-1} \sum_{j_2=0}^{k_2-1} \cdots \sum_{j_d=0}^{k_d-1} K(j_1, j_2, \ldots, j_d) X_{\mathrm{CP}}(i_1 + j_1, i_2 + j_2, \ldots, i_d + j_d), \tag{2}$$

where $i_\ell \in \{0, 1, \ldots, M_\ell - 1\}$.

CP-Convolution. The objective of CP-Conv is to evaluate this same convolution directly from the CP factors, without ever materializing the dense tensor X_{CP}. Substituting the CP representation (equation 1) into the convolution (equation 2) gives the following,

$$Y(i_1, i_2, \ldots, i_d) = \sum_{j_1=0}^{k_1-1} \sum_{j_2=0}^{k_2-1} \cdots \sum_{j_d=0}^{k_d-1} K(j_1, \ldots, j_d) \sum_{r=1}^{R} \prod_{\ell=1}^{d} U_\ell^r(i_\ell + j_\ell). \tag{3}$$

For each mode ℓ, rank component r, and output coordinate i_ℓ, define the one-dimensional sliding window,

$$W_{\ell,r,i_\ell}(j_\ell) = U_\ell^r(i_\ell + j_\ell), \qquad j_\ell = 0, \ldots, k_\ell - 1.$$

The key observation is that the separability of each CP rank component allows the sums in equation 3 to be reordered into a sequence of one-dimensional contractions with the unrestricted dense kernel. In particular,

$$Y(i_1, \ldots, i_d) = \sum_{r=1}^{R} \sum_{j_1=0}^{k_1-1} W_{1,r,i_1}(j_1) \sum_{j_2=0}^{k_2-1} W_{2,r,i_2}(j_2) \cdots \sum_{j_d=0}^{k_d-1} W_{d,r,i_d}(j_d) K(j_1, \ldots, j_d). \tag{4}$$

Equation 4 is the central computational form underlying CP-Conv. For each CP rank component and output location, the dense kernel is successively contracted along its modes with the corresponding one-dimensional factor windows. Each contraction removes one kernel index, until a scalar remains, and summing these scalars over the CP rank gives the convolution value. Thus, the convolution is evaluated directly from the CP factors without reconstructing the dense input tensor, while the convolutional kernel remains fully dense and unrestricted. Equation 4 is written pointwise for clarity, but in the implementation, the same contractions are evaluated simultaneously over the banks of output coordinates, with intermediate contraction results reused across locations. For multiple output channels, the same contraction is applied independently to each corresponding dense trainable kernel. The derivation is written for a single input channel for notational simplicity. Multiple input channels introduce an additional channel index and summation without changing the mode-wise contraction principle.

Since equation 4 is obtained solely by reordering finite sums, CP-Conv introduces no approximation of the convolutional operator itself. For the tensor represented by the supplied CP factors, it is algebraically equivalent to dense convolution. Any approximation therefore arises only from the input representation when a truncated CP decomposition is used.

Compressed CP-Conv. Although CP-Conv avoids materializing the dense input, its output Y is generally dense. We therefore introduce mode-wise compression matrices $P_\ell \in \mathbb{R}^{M_\ell \times m_\ell}$, with $m_\ell \ll M_\ell$, and define the following compressed output,

$$Z(q_1, \ldots, q_d) = \sum_{i_1=0}^{M_1-1} \cdots \sum_{i_d=0}^{M_d-1} Y(i_1, \ldots, i_d) \prod_{\ell=1}^{d} P_\ell(i_\ell, q_\ell), \tag{5}$$

where $q_\ell \in \{0, \ldots, m_\ell - 1\}$. In our main neural-network setting, the matrices P_ℓ are trainable[1]. Substituting the CP-Conv expression into equation 5 and rearranging the finite sums shows that compression can be applied directly to the sliding-window banks. Specifically, define

$$\widetilde{W}_{\ell,r,q_\ell}(j_\ell) = \sum_{i_\ell=0}^{M_\ell-1} P_\ell(i_\ell, q_\ell) W_{\ell,r,i_\ell}(j_\ell),$$

the key consequence is that the output compression can therefore be performed before the kernel contractions. Substituting the compressed windows into the same contraction structure as equation 4 gives the following,

$$Z(q_1, \ldots, q_d) = \sum_{r=1}^{R} \sum_{j_1=0}^{k_1-1} \widetilde{W}_{1,r,q_1}(j_1) \sum_{j_2=0}^{k_2-1} \widetilde{W}_{2,r,q_2}(j_2) \cdots \sum_{j_d=0}^{k_d-1} \widetilde{W}_{d,r,q_d}(j_d) K(j_1, \ldots, j_d). \tag{6}$$

Equation 6 shows that convolution and compression do not need to be evaluated as two consecutive dense operations. Instead, the output-coordinate dependence is compressed at the level of the one-dimensional factor windows before the dense-kernel contractions are performed. Consequently,

[1]Learned projections fix geometry and may break equivariance. Pooling can help (Appendix A.12).

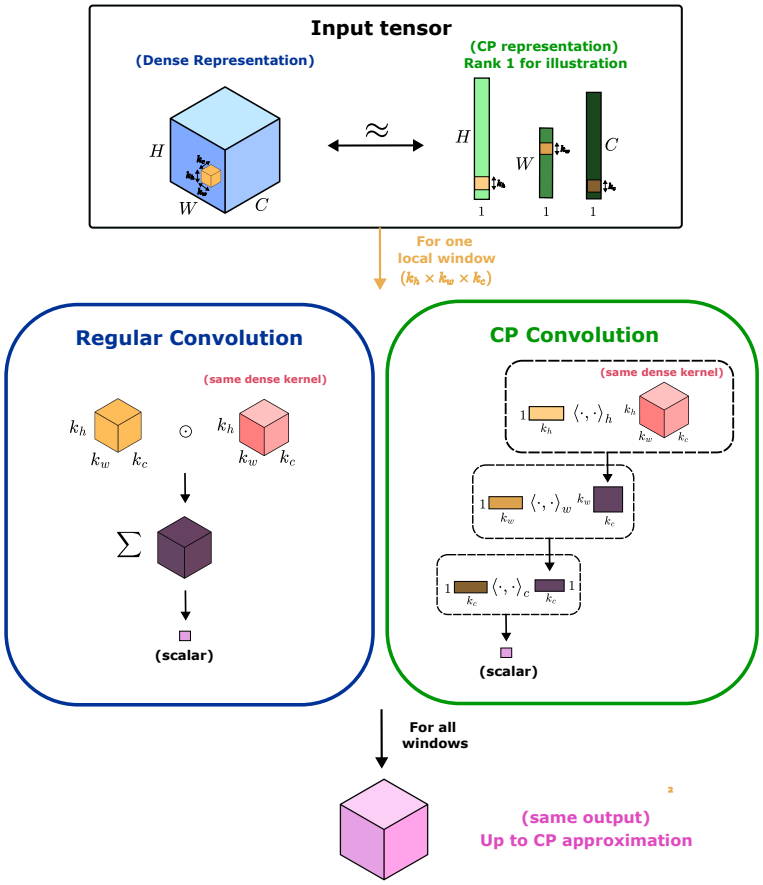


Figure 1: Illustration of CP-Conv for a 3D rank-one input. The same dense kernel is applied as in conventional convolution, but the operation is evaluated directly from the CP factors. Higher CP ranks introduce an additional summation over rank components, and the construction extends directly to higher dimensions.

CCP-Conv directly produces $Z \in \mathbb{R}^{m_1 \times \cdots \times m_d}$ without materializing either the dense input tensor or the full dense convolution output. A detailed recursive derivation of equations 4 and 6, including the explicit sequence of mode-wise contractions, is provided in Appendix A.1.

Computational complexity. To expose the main scaling behavior, consider the isotropic case $N_\ell = N$, $M_\ell = M$, $k_\ell = k$, and $m_\ell = m$, with $M \gg k$. Dense convolution requires $\mathcal{O}(M^d k^d)$ operations. For CP-Conv, the sequence of mode-wise contractions requires

$$\mathcal{O}\left(R \sum_{\ell=1}^{d} M^\ell k^{d-\ell+1}\right),$$

which reduces to the leading-order cost $\mathcal{O}\big(RM^d k\big)$ when $M \gg k$. This corresponds to an approximate arithmetic reduction of k^{d-1}/R relative to dense convolution. For CCP-Conv, constructing the compressed sliding-window banks requires $\mathcal{O}(RdMmk)$ operations, while the subsequent kernel contractions require $\mathcal{O}\Big(R \sum_{\ell=1}^{d} m^\ell k^{d-\ell+1}\Big)$. The resulting total cost is therefore

$$\mathcal{O}\left(RdMmk + R \sum_{\ell=1}^{d} m^\ell k^{d-\ell+1}\right).$$

Thus, CCP-Conv avoids both the dense M^d convolution output and the k^d factor in the dominant dense-convolution cost when the CP rank and compressed dimensions remain moderate. The

corresponding general non-isotropic expressions and detailed storage accounting are given in Appendix A.2. Table 1 summarizes the computational and storage costs and gives a representative 4D example. Although CP-Conv avoids storing the dense input, its dense output retains an $\mathcal{O}(N^d)$ leading-order storage cost for fixed kernel sizes, since $M = N - k + 1$ scales with N. CCP-Conv removes this remaining dense-output bottleneck through fused compression.

Table 1: Computational and memory/storage complexity in the isotropic case $N_\ell = N$, $M_\ell = M$, $k_\ell = k$, and $m_\ell = m$, assuming $M \gg k$. The example uses a 4D tensor with $N = 512$, $k = 32$, $M = 481$, CP rank $R = 8$, and compression size $m = 32$. The example storage values include the input representation, kernel, output, compression matrices, window terms, and implementation workspace. The detailed accounting is given in Appendix A.2.

Method	Computational cost	Leading memory/storage	Example operations	Example stored scalars
Regular Conv	$\mathcal{O}(M^d k^d)$	$\mathcal{O}(N^d)$	5.61×10^{16}	1.22×10^{11}
CP-Conv	$\mathcal{O}\left(R \sum_{\ell=1}^{d} M^\ell k^{d-\ell+1}\right)$	$\mathcal{O}(N^d)$	1.47×10^{13}	5.35×10^{10}
CCP-Conv	$\mathcal{O}\left(RdMmk + R \sum_{\ell=1}^{d} m^\ell k^{d-\ell+1}\right)$	$\mathcal{O}(RdN + k^d + dMm + Rdmk + m^d)$	1.09×10^{9}	2.21×10^{6}

3 Experiments

We evaluate CP-Conv and CCP-Conv along three complementary directions. First, we verify that the proposed operators reproduce their dense counterparts up to numerical precision and study their computational scaling. We then evaluate CCP-Conv as a trainable front-end in supervised learning, from controlled dense-feasible settings to regimes where dense processing becomes impractical or infeasible. Finally, we investigate CCP-Conv in unsupervised representation learning.

3.1 Validation and Computational Scaling

We first validate the correctness of CP-Conv and CCP-Conv and evaluate their computational scaling against dense convolution baselines. We generate synthetic tensors directly from their CP representation as in equation 1, and compare

$$Y_{\text{conv}} = \text{Conv}(X_{\text{CP}}), \qquad Z_{\text{conv+comp}} = \text{Comp}(Y_{\text{conv}}),$$

with

$$Y_{\text{CP}} = \text{CPConv}\left(\{U_\ell^r\}_{\substack{\ell=1,\dots,d \\ r=1,\dots,R}}\right), \qquad Z_{\text{CCP}} = \text{CCPConv}\left(\{U_\ell^r\}_{\substack{\ell=1,\dots,d \\ r=1,\dots,R}}\right).$$

Here, Comp denotes the same mode-wise compression used by CCP-Conv, applied after dense convolution. Correctness is measured using

$$\varepsilon_{\text{CP}} = \frac{\|Y_{\text{CP}} - Y_{\text{conv}}\|_2}{\|Y_{\text{conv}}\|_2}, \qquad \varepsilon_{\text{CCP}} = \frac{\|Z_{\text{CCP}} - Z_{\text{conv+comp}}\|_2}{\|Z_{\text{conv+comp}}\|_2}.$$

Table 2 reports representative validation results and runtimes. All timings are median wall-clock times over 20 runs after JIT compilation. CP-Conv agrees with dense convolution up to numerical precision, while CCP-Conv agrees with dense convolution followed by the same compression. The computational advantage grows particularly strongly with input dimension and kernel size.

3.2 Supervised Learning with CCP-Conv

We next evaluate CCP-Conv as a trainable front-end for supervised learning. We consider controlled settings where CCP-Conv can be compared directly with an equivalent dense convolution–compression front-end, as well as scales where dense processing becomes impractical. Unless otherwise stated, the classification experiments use a single convolution–compression stage followed by a fully connected classifier, with the aim of isolating the input-processing operator rather than achieving state-of-the-art classification.

Table 2: Validation and computational scaling of dense convolution, dense convolution followed by compression, CP-Conv, and CCP-Conv. CP speedup is measured against dense convolution, while CCP speedup is measured against dense convolution followed by the same compression. All runtimes are median wall-clock times after JIT compilation. Minor runtime variation may arise from device and runtime caching effects.

Setting	Median runtime (ms)				Speedup		Relative ℓ_2 error	
	Conv	Conv+Comp	CP-Conv	CCP-Conv	CP	CCP	$\varepsilon_{\mathrm{CP}}$	$\varepsilon_{\mathrm{CCP}}$
Input-size sweep ($d = 3$, $k = 7$, $R = 100$, $m = 16$)								
$N = 48$	27.04	23.25	2.27	0.752	11.9×	**30.9**×	5.3×10^{-7}	5.8×10^{-7}
$N = 64$	60.72	59.94	2.74	0.653	22.2×	**91.7**×	5.3×10^{-7}	6.2×10^{-7}
$N = 96$	202.6	203.6	5.41	0.770	37.5×	**265**×	5.3×10^{-7}	6.5×10^{-7}
$N = 128$	502.5	500.7	10.44	0.826	48.1×	**607**×	5.3×10^{-7}	6.8×10^{-7}
$N = 160$	1 000	1 001	18.80	0.814	53.2×	**1 231**×	5.3×10^{-7}	7.1×10^{-7}
Kernel-size sweep ($d = 3$, $N = 128$, $R = 100$, $m = 16$)								
$k = 3$	51.64	50.92	7.66	0.185	6.7×	**275**×	3.5×10^{-7}	5.8×10^{-7}
$k = 5$	196.8	195.4	7.53	0.559	26.2×	**350**×	4.5×10^{-7}	6.3×10^{-7}
$k = 7$	503.7	504.3	10.27	0.758	49.1×	**666**×	5.3×10^{-7}	6.8×10^{-7}
$k = 11$	1 200	1 203	14.29	0.895	83.9×	**1 344**×	4.9×10^{-7}	7.7×10^{-7}
$k = 15$	2 167	2 161	19.97	1.31	109×	**1 647**×	5.6×10^{-7}	8.4×10^{-7}
Dimension sweep ($N = 128$, $k = 7$, $R = 100$, $m = 16$)								
$d = 2$	1.93	1.60	0.451	0.145	4.3×	**11.0**×	4.7×10^{-7}	6.1×10^{-7}
$d = 3$	507.2	501.4	10.55	0.865	48.1×	**580**×	5.3×10^{-7}	6.8×10^{-7}
$d = 4$	33 296	37 110	2 371	9.42	14.0×	**3 941**×	8.4×10^{-7}	7.7×10^{-7}
Rank sweep ($d = 3$, $N = 128$, $k = 7$, $m = 16$)								
$R = 8$	507.7	504.1	2.23	0.167	228×	**3 017**×	2.0×10^{-7}	4.6×10^{-7}
$R = 16$	503.9	504.6	2.76	0.165	183×	**3 062**×	2.4×10^{-7}	4.7×10^{-7}
$R = 32$	504.2	503.1	3.85	0.133	131×	**3 794**×	3.2×10^{-7}	5.2×10^{-7}
$R = 64$	502.0	501.8	6.76	0.283	74.3×	**1 775**×	4.3×10^{-7}	5.9×10^{-7}
$R = 128$	502.3	501.6	13.40	1.06	37.5×	**472**×	5.9×10^{-7}	7.3×10^{-7}
$R = 256$	496.4	496.9	24.63	1.23	20.2×	**405**×	8.3×10^{-7}	9.8×10^{-7}
Compression-size sweep (does not affect Conv and CP-Conv) ($d = 3$, $N = 128$, $k = 7$, $R = 100$)								
$m = 8$	–	494.7	–	0.551	–	**899**×	–	6.0×10^{-7}
$m = 12$	–	494.4	–	0.895	–	**553**×	–	6.6×10^{-7}
$m = 16$	–	496.8	–	0.969	–	**513**×	–	6.8×10^{-7}
$m = 24$	–	494.8	–	1.13	–	**440**×	–	5.4×10^{-7}
$m = 32$	–	494.9	–	1.69	–	**293**×	–	6.8×10^{-7}

3.2.1 Controlled Classification from Moderate to Extreme Scale

We evaluate CCP-Conv on synthetic videos of rotating objects, UCF101 (Soomro et al., 2012), Ego4D (Grauman et al., 2022), and high-dimensional Fokker–Planck distributions. Dense-feasible inputs range from $64 \times 64 \times 32$ synthetic videos to $128 \times 96 \times 24$ Ego4D clips. We additionally consider $1920 \times 2560 \times 128$ Ego4D inputs and seven-dimensional Fokker–Planck tensors of implicit size $48^6 \times 24$, corresponding to 2.94×10^{11} values, or approximately 1.17 TB in float32. Full dataset and optimization details are given in Appendices A.3 and A.4. Table 3 reports test accuracy and elapsed time across random seeds.

Across dense-feasible settings, CCP-Conv achieves comparable predictive performance while substantially reducing elapsed time, from approximately $10\times$–$22\times$ on the synthetic and UCF101 experiments to approximately $156\times$ on low-resolution Ego4D. At larger scales, the high-resolution Ego4D experiment serves primarily as a feasibility demonstration, while CCP-Conv also operates directly on the Fokker–Planck inputs whose implicit dense representation reaches 2.94×10^{11} values per sample. These results support CCP-Conv as a representation-aware front-end that exploits input structure without constraining the trainable kernel.

To test the method when CP structure is only approximate, we additionally consider nonlinear shallow-water dynamics from PDEBench (Takamoto et al., 2022), and the results are shown in Table 4. The matched CCP and dense predictors obtain relative ℓ_2 errors of 0.02316 and 0.02314,

Table 3: Mean test accuracy and elapsed time across the supervised classification experiments. Results are reported as mean ± standard deviation. Chance denotes uniform random classification. A dash indicates that the dense baseline could not be executed within the available resources. Higher accuracy and lower time are better.

			Test accuracy (%)		Elapsed time (s)	
Dataset	Setting	Chance	CCP	Dense	CCP	Dense
Synthetic	Easy, 2 cls.	50.0	100.0 ± 0.0	100.0 ± 0.0	$\mathbf{19.0 \pm 0.9}$	359.6 ± 1.5
Synthetic	Medium, 4 cls.	25.0	85.0 ± 7.0	85.0 ± 8.0	$\mathbf{17.0 \pm 0.7}$	367.9 ± 6.2
Synthetic	Hard, 5 cls.	20.0	73.0 ± 12.0	73.0 ± 10.0	$\mathbf{17.1 \pm 0.3}$	366.2 ± 9.6
UCF101	3 cls.	33.3	75.0 ± 4.0	73.0 ± 7.0	$\mathbf{4.3 \pm 2.6}$	73.4 ± 0.5
UCF101	10 cls.	10.0	31.6 ± 2.0	30.9 ± 2.0	$\mathbf{39.7 \pm 4.4}$	407.1 ± 17.1
UCF101	101 cls.	1.0	9.5 ± 1.0	9.5 ± 2.0	$\mathbf{199.8 \pm 12.9}$	4178.8 ± 182.1
Ego4D	Low res., 15 cls.	6.7	16.0 ± 1.0	11.0 ± 5.0	$\mathbf{39.9 \pm 0.4}$	6211.4 ± 105.2
Ego4D	High res., 5 cls.	20.0	29.0 ± 3.0	–	$\mathbf{13.1 \pm 0.2}$	–
FP	FP-05, 3 cls.	33.3	41.0 ± 2.0	–	$\mathbf{18.0 \pm 0.4}$	–
FP	FP-035, 3 cls.	33.3	38.0 ± 2.0	–	$\mathbf{24.6 \pm 0.4}$	–

Table 4: One-step shallow-water prediction on PDEBench using rank-10 CP inputs.

Method	Rel. ℓ_2 error	Time (s)
CCP-Conv	0.02316	**17.2**
Dense Conv	0.02314	164.1

respectively, with $9.55\times$ reduction in training time for CCP. Full results and qualitative predictions are provided in Appendix A.5.

3.2.2 Integration into a Deep Video Architecture

The preceding experiments use shallow models to isolate the front-end operator. We therefore integrate CCP-Conv into a Kinetics-400-pretrained R3D-18 (Tran et al., 2018) on the complete 101-class UCF101 dataset, using $16 \times 112 \times 112$ grayscale clips and CP rank $R = 300$. Both the CCP stem and its dense counterpart produce features of the same shape for the downstream R3D-18 blocks (details in Appendix A.6). This experiment aims to demonstrate compatibility with an established deep architecture rather than achieve state-of-the-art video recognition. CCP-Conv achieves 84.8% test accuracy, comparable to 84.2% for the dense front-end, while reducing measured training time by $\approx 5.2\%$. The smaller whole-network speedup is expected because the rank required for near-dense predictive performance is relatively high and only the input-processing stage is replaced; the downstream R3D-18 blocks remain dense. Possible deeper integration is discussed in Appendix A.7.

3.3 Unsupervised Representation Learning

We train CCP-Conv autoencoders on UCF101 and Ego4D without class labels. The encoder operates directly on the CP factors, while the decoder predicts a CP representation of the input using a reconstruction objective evaluated directly in factorized form. Architectural and optimization details are given in Appendix A.8. After training, the encoder is frozen and its latent representations are evaluated using a regularized linear Ridge classifier fitted only on the training-set latents and labels. As a control, we apply the same probe to representations produced by the corresponding randomly initialized, untrained encoder using the same CP inputs. On UCF101, reconstruction training increases linear-probe accuracy from 4.94% for the random-init encoder to 8.09%; on Ego4D it increases from 9.84% to 15.25%. Thus, reconstruction training makes additional class information linearly accessible beyond that provided by the untrained architecture under the same probe. These experiments are not intended as competitive unsupervised video benchmarks, but show that

Table 5: Integration of CCP-Conv into R3D-18 on the complete UCF101 dataset. Results are reported as mean ± standard deviation over three seeds.

Method	Test accuracy (%)	Elapsed time (s)
Random chance	0.99	–
CCP-Conv + R3D-18	84.8 ± 0.5	$\mathbf{265 \pm 2.3}$
Dense Conv + R3D-18	84.2 ± 5.1	279.6 ± 3.4

Table 6: Unsupervised representation learning with CCP-Conv. The autoencoder is trained without class labels. Frozen latent representations are evaluated using a regularized linear Ridge classifier fitted on the training-set. Results are mean ± standard deviation across five random seeds.

Dataset	Test probe accuracy (%)	Random-init AE (%)	Chance (%)	Training time (s)
UCF101 (101 cls.)	8.09 ± 0.98	4.94 ± 1.40	0.99	4303 ± 42
Ego4D (15 cls.)	15.25 ± 4.27	9.84 ± 1.78	6.67	1167 ± 100

CCP-Conv can also support unsupervised representation learning. A matched dense autoencoder was additionally profiled but was computationally prohibitive to train fully. Details are provided in Appendix A.8.

3.4 Further Analysis and Limitations

Several aspects determine the practical operating regime of CCP-Conv:

1. **CP representation rank.** Increasing rank generally improves representation fidelity but also increases CCP-Conv cost. Appendix A.9 examines this trade-off across different ranks.
2. **Offline CP decomposition.** When initially dense data must be factorized, a one-time decomposition cost is incurred. This is unnecessary when data are generated or stored directly in CP form (Appendix A.10).
3. **Alternative compact input representations.** Appendix A.11 compares CCP-Conv with spatiotemporal resizing, temporal subsampling, patch pooling, and truncated SVD at approximately matched storage.
4. **Choice of output reduction.** Appendix A.12 compares learned projections with fixed adaptive average pooling, which performs comparably on UCF101 while learned projections improve the synthetic problems.
5. **Multi-layer integration.** CCP-Conv currently exploits structure mainly at the network front-end. Extensions that preserve structured computation deeper in the architecture are discussed in Appendix A.7.

4 Conclusion

We introduced CP-Conv, an exact dense-kernel convolutional operator for CP-represented inputs, and CCP-Conv, which additionally fuses mode-wise compression to avoid materializing both the dense input and full convolutional output. Unlike low-rank or separable-kernel approaches, our formulation keeps the trainable kernel unrestricted and exploits structure already present in the input representation. Experiments show that this representation-aware formulation remains effective both when CP structure is directly available and when dense data are only approximately factorized. CCP-Conv achieves predictive performance comparable to matched dense counterparts when these remain feasible, while extending computation to regimes where dense processing becomes impractical. Integration into R3D-18 and autoencoder settings further demonstrates compatibility with deep and unsupervised learning. When factorization must be performed offline, its cost remains an important practical consideration. Future work includes extending structured computation deeper into neural architectures, supporting other tensor formats, and reducing or learning the factorization stage.

AI USE STATEMENT

In this work, we used generative AI tools to assist with writing and debugging code for synthetic dataset generation, improving the clarity and readability of mathematical proofs, providing feedback on research methodology and experimental design, and assisting with the implementation of methods, including identifying efficient ways of expressing convolution operations. Generative AI was also used to assist with dataset cleaning and reformatting, including preparing files suitable for the training pipeline, and to support discussions concerning the interpretation of experimental results.

Additionally, generative AI tools were used during manuscript preparation to improve clarity, readability, and organization, and to assist with drafting, correcting, and redrafting parts of the paper. These tools also supported literature discovery by suggesting potentially relevant publications and helping identify related work. All AI-assisted manuscript content was reviewed and revised by the authors, and the suggested references were independently verified against the original publications.

We did not use generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, assist with translation, or support qualitative or thematic data analysis.

We have reviewed all AI-assisted work. In particular, the AI-assisted presentation and writing of mathematical proofs were independently checked by the authors for correctness and consistency with the stated assumptions. AI-assisted code was inspected, tested, and validated numerically against reference implementations where applicable, including comparisons with standard dense convolution. Synthetic data generation and dataset-processing scripts were similarly reviewed to ensure that the resulting data and transformations matched the intended experimental protocols. AI-assisted suggestions concerning methodology and experimental design were independently evaluated by the authors before being incorporated into the work. Finally, AI-assisted manuscript content and literature suggestions were manually reviewed to ensure that scientific claims were appropriately supported, references were accurate and relevant, and no unattributed material was introduced.

We take full responsibility for the final content of this work, including all text, mathematical claims, implementations, datasets, and experimental results produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We have made extensive efforts to ensure the reproducibility of the results presented in this work. The main paper and appendix provide detailed descriptions of the proposed operators, computational complexity, experimental protocols, preprocessing procedures, model architectures, hyperparameters, and additional ablation studies. Complete derivations and proofs are also provided in the appendix where applicable. In addition, the source code used to implement the proposed methods and reproduce the experiments is publicly available in the supplementary material.

REFERENCES

Francisco Chinesta, Pierre Ladeveze, and Elias Cueto. A short review on model order reduction based on proper generalized decomposition. *Archives of Computational Methods in Engineering*, 18(4):395–404, 2011. doi: 10.1007/s11831-011-9064-7.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Christoph Feichtenhofer. X3d: Expanding architectures for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 203–213, 2020.

Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6202–6211, 2019.

Kristen Grauman, Andrew Westbury, Eugene Byrne, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Wolfgang Hackbusch. Numerical tensor calculus. *Acta Numerica*, 23:651–742, 2014. doi: 10.1017/S0962492914000087.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16000–16009, 2022.

Lifang He, Xiangnan Kong, Philip S. Yu, Ann B. Ragin, Zhifeng Hao, and Xiaowei Yang. DuSK: A dual structure-preserving kernel for supervised tensor learning with applications to neuroimages. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pp. 127–135, 2014. doi: 10.1137/1.9781611973440.15.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Boris N. Khoromskij. Tensors-structured numerical methods in scientific computing: Survey on recent advances. *Chemometrics and Intelligent Laboratory Systems*, 110(1):1–19, 2012. ISSN 0169-7439. doi: 10.1016/j.chemolab.2011.09.001. URL https://www.sciencedirect.com/science/article/pii/S0169743911001808.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations*, 2016.

Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. doi: 10.1137/07070111X. URL https://doi.org/10.1137/07070111X.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research*, 20(26):1–6, 2019. URL https://www.jmlr.org/papers/v20/18-277.html.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations*, 2015.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. doi: 10.1137/090752286. URL https://doi.org/10.1137/090752286.

Beheshteh Rakhshan and Guillaume Rabusseau. Tensorized random projections. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 3306–3316, 2020.

Maxim V. Rakhuba and Ivan V. Oseledets. Fast multidimensional convolution in low-rank tensor formats via cross approximation. *SIAM Journal on Scientific Computing*, 37(2):A565–A582, 2015. doi: 10.1137/140958529.

Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017. doi: 10.1109/TSP.2017.2690524.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks*, 2022. URL https://arxiv.org/abs/2210.07182.

Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. VideoMAE: Masked autoencoders are data-efficient learners for self-supervised video pre-training. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6450–6459, 2018.

Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R. Manmatha, Alexander J. Smola, and Philipp Krähenbühl. Compressed video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6026–6035, 2018.

Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*, 2016. URL https://arxiv.org/abs/1606.05535.

A Appendix

A.1 Detailed contraction derivation

This section provides the detailed recursive contraction form underlying equations 4 and 6. Recall the sliding windows

$$W_{\ell,r,i_\ell}(j_\ell) = U_\ell^r(i_\ell + j_\ell), \qquad j_\ell = 0, \dots, k_\ell - 1.$$

As discussed in the main paper, substituting these windows into equation 3 and rearranging the finite sums yields

$$Y(i_1, \dots, i_d) = \sum_{r=1}^{R} \sum_{j_1=0}^{k_1-1} W_{1,r,i_1}(j_1) \sum_{j_2=0}^{k_2-1} W_{2,r,i_2}(j_2) \cdots \sum_{j_d=0}^{k_d-1} W_{d,r,i_d}(j_d) K(j_1, \dots, j_d).$$

For a fixed output position $\mathbf{i} = (i_1, \dots, i_d)$ and rank component r, define

$$T_{r,\mathbf{i}}^{(d)}(j_1, \dots, j_d) = K(j_1, \dots, j_d).$$

Then, for $m = d, d-1, \dots, 1$, recursively define

$$T_{r,\mathbf{i}}^{(m-1)}(j_1, \dots, j_{m-1}) = \sum_{j_m=0}^{k_m-1} W_{m,r,i_m}(j_m) T_{r,\mathbf{i}}^{(m)}(j_1, \dots, j_m).$$

Each contraction removes one kernel-offset index. After all d contractions, $T_{r,\mathbf{i}}^{(0)}$ is a scalar, and the convolution value is

$$Y(i_1, \dots, i_d) = \sum_{r=1}^{R} T_{r,\mathbf{i}}^{(0)}.$$

For compact notation, define the contraction of an intermediate tensor $A(j_1, \dots, j_m)$ with a vector $w(j_m)$ along j_m as

$$\mathrm{Contr}_{j_m}(A, w)(j_1, \dots, j_{m-1}) = \sum_{j_m=0}^{k_m-1} A(j_1, \dots, j_m) w(j_m).$$

CP-Conv can then be written as

$$Y(i_1, \dots, i_d) = \sum_{r=1}^{R} \mathrm{Contr}_{j_1}\Big(\cdots \mathrm{Contr}_{j_{d-1}}\big(\mathrm{Contr}_{j_d}(K, W_{d,r,i_d}), W_{d-1,r,i_{d-1}} \big) \cdots, W_{1,r,i_1} \Big). \tag{7}$$

For CCP-Conv, substituting equation 7 into the mode-wise compression equation 5 and exchanging the finite sums gives the compressed windows

$$\widetilde{W}_{\ell,r,q_\ell}(j_\ell) = \sum_{i_\ell=0}^{M_\ell-1} P_\ell(i_\ell, q_\ell) W_{\ell,r,i_\ell}(j_\ell) = \sum_{i_\ell=0}^{M_\ell-1} P_\ell(i_\ell, q_\ell) U_\ell^r(i_\ell + j_\ell).$$

Therefore,

$$Z(q_1, \dots, q_d) = \sum_{r=1}^{R} \mathrm{Contr}_{j_1}\Big(\cdots \mathrm{Contr}_{j_{d-1}}\big(\mathrm{Contr}_{j_d}(K, \widetilde{W}_{d,r,q_d}), \widetilde{W}_{d-1,r,q_{d-1}} \big) \cdots, \widetilde{W}_{1,r,q_1} \Big). \tag{8}$$

Equation 8 makes explicit that CCP-Conv has the same kernel-contraction structure as CP-Conv, while the output-coordinate dependence is compressed before the contractions are performed. Hence, when the input is supplied through its CP factors, neither the dense input tensor nor the full dense convolution output needs to be materialized during the CCP-Conv forward operation.

A.2 General computational and storage complexity

In this section, we present the non-isotropic complexity expressions underlying the simplified analysis in the main text. Define

$$\mathcal{N}_{l_0,l_f} = \prod_{\ell=l_0}^{l_f} N_\ell, \qquad \mathcal{M}_{l_0,l_f} = \prod_{\ell=l_0}^{l_f} M_\ell,$$
$$\mathcal{K}_{l_0,l_f} = \prod_{\ell=l_0}^{l_f} k_\ell, \qquad \mathcal{Q}_{l_0,l_f} = \prod_{\ell=l_0}^{l_f} m_\ell.$$

For valid convolution, $M_\ell = N_\ell - k_\ell + 1$. Conventional dense convolution applies a kernel containing $\mathcal{K}_{1,d}$ entries at each of the $\mathcal{M}_{1,d}$ output positions, giving

$$\mathcal{C}_{\text{dense}} = \mathcal{O}(\mathcal{M}_{1,d}\mathcal{K}_{1,d})\,.$$

CP-Conv evaluates the convolution through successive contractions between the dense kernel and the CP-factor windows. With the contraction order used in Appendix A.1, its arithmetic cost is

$$\mathcal{C}_{\text{CP}} = \mathcal{O}\left(R\sum_{p=1}^{d}\mathcal{K}_{1,p}\mathcal{M}_{p,d}\right).$$

When $M_\ell \gg k_\ell$, the leading term is

$$\mathcal{C}_{\text{CP}} = \mathcal{O}(R\mathcal{K}_{1,1}\mathcal{M}_{1,d})\,,$$

which gives the approximate arithmetic reduction

$$\frac{\mathcal{C}_{\text{dense}}}{\mathcal{C}_{\text{CP}}} \approx \frac{\mathcal{K}_{2,d}}{R}.$$

For CCP-Conv, constructing the compressed sliding windows requires

$$\mathcal{O}\left(R\sum_{\ell=1}^{d}M_\ell m_\ell k_\ell\right)$$

operations. The subsequent sequence of kernel contractions requires

$$\mathcal{O}\left(R\sum_{p=1}^{d}\mathcal{K}_{1,p}\mathcal{Q}_{p,d}\right).$$

Thus,

$$\mathcal{C}_{\text{CCP}} = \mathcal{O}\left(R\sum_{\ell=1}^{d}M_\ell m_\ell k_\ell + R\sum_{p=1}^{d}\mathcal{K}_{1,p}\mathcal{Q}_{p,d}\right).$$

For storage, a conventional dense implementation stores the dense input, kernel, and output, giving

$$\mathcal{S}_{\text{dense}} = \mathcal{O}(\mathcal{N}_{1,d} + \mathcal{K}_{1,d} + \mathcal{M}_{1,d})\,.$$

CP-Conv replaces the dense input by its factors, but its output remains dense. If the one-dimensional window banks are explicitly stored, a representative storage bound is

$$\mathcal{S}_{\text{CP}} = \mathcal{O}\left(R\sum_{\ell=1}^{d}N_\ell + \mathcal{K}_{1,d} + \mathcal{M}_{1,d} + R\sum_{\ell=1}^{d}M_\ell k_\ell\right),$$

up to lower-order implementation-dependent contraction workspace.

CCP-Conv avoids the dense output and stores instead the CP factors, dense kernel, mode-wise compression matrices, compressed windows, and compact output:

$$\mathcal{S}_{\text{CCP}} = \mathcal{O}\left(R\sum_{\ell=1}^{d}N_\ell + \mathcal{K}_{1,d} + \sum_{\ell=1}^{d}M_\ell m_\ell + R\sum_{\ell=1}^{d}m_\ell k_\ell + \mathcal{Q}_{1,d}\right),$$

again up to implementation-dependent contraction buffers.

In the isotropic setting $N_\ell = N$, $M_\ell = M$, $k_\ell = k$, and $m_\ell = m$, with $M \gg k$, these expressions reduce to

$$\mathcal{C}_{\text{dense}} = \mathcal{O}(M^d k^d), \qquad \mathcal{C}_{\text{CP}} = \mathcal{O}(RM^d k),$$

and

$$\mathcal{C}_{\text{CCP}} = \mathcal{O}\left(RdMmk + R\sum_{\ell=1}^{d} m^\ell k^{d-\ell+1} \right),$$

with leading storage

$$\mathcal{S}_{\text{dense}} = \mathcal{O}(N^d), \qquad \mathcal{S}_{\text{CP}} = \mathcal{O}(N^d), \qquad \mathcal{S}_{\text{CCP}} = \mathcal{O}(RdN + k^d + dMm + Rdmk + m^d).$$

A.3 Dataset Construction

Synthetic rotating-shape data. We generated balanced datasets of videos with spatial resolution 64×64 and 32 time steps, corresponding to $131{,}072$ scalar values per sample. Each video contains a single smooth geometric object undergoing a rotation together with a small translation. Initial orientation, angular velocity and direction, initial position, translational velocity, scale, and contrast were randomized across samples. Three levels of classification difficulty were considered. The easy dataset contains two classes (ellipse and ring), the medium dataset additionally contains crescent and two-lobed shape categories, and the hard dataset further includes a barbell shape, which is visually similar to the two-lobed class. Each dataset contains $10{,}000$ samples, split once into $8{,}000$ training and $2{,}000$ test samples. A CP rank of $R = 20$ was used.

UCF101. We considered three-class, ten-class, and full 101-class subsets of UCF101 (details about the subsets can be found in the supplementary material). Each video was converted to grayscale and represented by 16 approximately uniformly spaced frames sampled over its full duration. Frames were resized to 128×96 and pixel intensities were normalized to $[0, 1]$, resulting in tensors of size $128 \times 96 \times 16$.

To prevent clips originating from the same UCF101 group from appearing in both training and evaluation sets, we used a group-wise split as advised on their website. This resulted in $333/106$, $983/335$, and $10{,}160/3{,}160$ train/test samples for the 3-, 10-, and 101-class settings, respectively. The full split therefore contains all $13{,}320$ UCF101 videos. A CP rank of $R = 150$ was used in all three settings.

Ego4D. We constructed two classification settings from Ego4D using the official training and validation assignments, with the validation partition used for held-out evaluation.

The low-resolution setting contains 15 classes and consists of grayscale videos represented by 24 frames at spatial resolution 128×96, giving tensors of size $128 \times 96 \times 24$. It contains $2{,}693$ training samples and 750 held-out samples. The videos were represented using CP rank $R = 200$.

The high-resolution setting contains five classes and uses grayscale segments represented by 128 frames at spatial resolution 1920×2560. Each dense sample therefore contains

$$1920 \times 2560 \times 128 = 629{,}145{,}600$$

scalar values, corresponding to approximately 2.52 GB in float32 before accounting for intermediate activations or gradients. For this setting, directly computing a single high-rank CP decomposition was slow. We therefore constructed the representation greedily by successively fitting rank-8 CP decompositions to the current residual and subtracting the reconstructed contribution after each block. Forty such blocks were concatenated, yielding a final CP rank of $R = 320$. The resulting dataset contains 90 training and 25 held-out samples. Dense-model training could not be executed for this setting within the available memory resources.

PDEBench shallow-water data. We use the two-dimensional radial dam-break shallow-water dataset from PDEBench (Takamoto et al., 2022). The dataset contains $1{,}000$ simulated trajectories, each consisting of 101 states on a 128×128 spatial grid. For each trajectory, the first 10 states are used as the model input, forming a dense tensor of size $128 \times 128 \times 10$, while the immediately

following state is used as the prediction target and retained in dense form with size 128×128. We use a fixed split of 900 training and 100 test trajectories. The input histories are approximately represented using CP rank $R = 10$, whereas the prediction targets are not CP-decomposed.

Fokker–Planck data. We considered two balanced three-class datasets, denoted FP-05 and FP-035. Each sample represents the evolution of a probability density over six state dimensions, with 48 discretization points per dimension and 24 time steps. The samples are generated directly in exact CP form from mixtures of separable Gaussian densities evolving according to an Ornstein–Uhlenbeck Fokker–Planck model. Consequently, their generation does not require materializing the corresponding dense tensors.

The six state dimensions are divided into three blocks. For each class, one block exhibits a positive dependence between the dimension-dependent drift rates and target states, a second block exhibits a negative dependence, and the remaining block is independent. In addition, the drift rates of the positive and negative blocks are multiplied by $1 + s$ and $1 - s$, respectively. We use $s = 0.5$ for FP-05 and $s = 0.35$ for FP-035. The smaller value reduces the class-dependent block-speed separation, making the classification problem more difficult.

Each sample has dense shape

$$48 \times 48 \times 48 \times 48 \times 48 \times 48 \times 24,$$

corresponding to

$$48^6 \times 24 = 293{,}534{,}171{,}136$$

scalar values, or approximately 1.17 TB in float32. We use an exact CP representation with $R = 48$ components. Each dataset contains $1{,}000$ samples and uses a fixed random $80/20$ train–test split, giving 800 training and 200 test samples. Because of the dense tensor size, a conventional dense baseline could not be evaluated.

A.4 Supervised Experimental Setup

Architecture and optimization. The controlled supervised experiments use a deliberately shallow architecture in order to isolate the effect of the convolutional front-end.

The CCP-Conv model contains a single unrestricted dense convolutional kernel followed by trainable mode-wise compression matrices. For three-dimensional inputs, the convolutional kernel has shape $(5, 5, 3)$. For a general d-dimensional input, the kernel shape is

$$(5, \ldots, 5, 3),$$

where the final dimension corresponds to the temporal or evolution coordinate.

The compressed CCP-Conv output is flattened and passed to a two-hidden-layer multilayer perceptron with hidden width 64 and ReLU activations, followed by the class logits.

For the synthetic dataset, the compressed representation was of shape $(5, 5, 2)$. All UCF101 datasets used a compressed shape of $(10, 10, 5)$. The low- and high-resolution Ego4D experiments instead use $(10, 10, 5)$ and $(2, 2, 2)$, respectively. For the seven-dimensional Fokker–Planck inputs, the compressed shape is $(5, 5, 5, 5, 5, 5, 2)$. Both the convolutional kernel and the compression matrices are learned jointly with the classifier.

The dense baseline uses the same convolutional kernel size, compressed feature dimensions, classifier architecture, optimizer, learning rate, batch size, and training duration. Its principal difference is that the original dense input sample is processed using a conventional dense convolution before the same type of mode-wise output compression is applied, while CCP-Conv operates on the corresponding CP representation. The comparison therefore isolates the computational effect of operating directly on the factorized input representation. The matched dense baseline deliberately uses the same unstrided convolution and output-compression structure to isolate this difference. More aggressively optimized dense pipelines using striding or pooling could reduce the absolute runtime gap.

All classifiers were trained using cross-entropy loss and AdaBelief with learning rate 10^{-3} and otherwise default optimizer parameters. At each optimization step, a minibatch was sampled randomly

Dataset	Train	Eval.	Batch	Epochs
Synthetic (each)	8,000	2,000	64	16
UCF101 – 3 classes	333	106	64	19
UCF101 – 10 classes	983	335	64	32
UCF101 – 101 classes	10,160	3,160	64	32
Ego4D low-res	2,693	750	64	11
Ego4D high-res	90	25	20	11
FP-05	800	200	20	3
FP-035	800	200	20	3

Table 7: Dataset sizes, mini-batch sizes, and approximate numbers of training epochs used in the supervised classification experiments.

without replacement from the training set. For each experiment, five independent runs were performed. The train/test partition was kept fixed across runs, while model initialization and training randomness were varied. The approximate number of epochs was computed from the number of optimization steps, batch size, and training-set size. The resulting training settings are summarized in Table 7.

Timing protocol. The elapsed training times reported in the supervised experiments correspond to the accumulated wall-clock time of the optimization iterations. Timing begins before mini-batch sampling at each iteration and includes batch extraction, the forward pass, loss and gradient computation, back-propagation, and the parameter update. The first compiled training step is therefore included.

Dataset loading, offline CP decomposition, model construction, optimizer initialization, checkpoint writing, and the final train/test accuracy evaluation are excluded from the reported times. The same timing procedure is used for CCP-Conv and for the executable dense baselines.

A.5 PDEBench Shallow-Water Experiment

We additionally evaluate CCP-Conv on nonlinear shallow-water dynamics from PDEBench (Takamoto et al., 2022). Unlike the Fokker–Planck experiments, where the inputs are generated directly in exact CP form, the PDEBench trajectories are initially dense and must therefore be approximated by a finite-rank CP representation. This experiment evaluates whether CCP-Conv remains effective when the factorized input representation is approximate rather than exact.

Prediction task. We use the two-dimensional radial dam-break shallow-water dataset. The dataset contains $1{,}000$ trajectories, each consisting of 101 states on a 128×128 spatial grid. For each trajectory, the first ten states form the model input,

$$X \in \mathbb{R}^{128\times128\times10},$$

and the immediately following state is used as the prediction target,

$$Y \in \mathbb{R}^{128\times128}.$$

We use 900 trajectories for training and 100 for testing. Only the input history is CP-decomposed. The prediction target remains the original dense PDE solution.

Approximate CP representation. Each complete $128 \times 128 \times 10$ input history is decomposed as a single tensor, without spatial or temporal blocking. Table 8 reports the reconstruction error and measured decomposition time for several candidate ranks.

We use rank $R = 10$ for the prediction experiment. This gives a mean relative reconstruction error of approximately 0.82% while keeping the factorized representation compact. A dense input contains

$$128 \times 128 \times 10 = 163{,}840$$

scalar values, whereas the rank-10 CP representation contains

$$10(128 + 128 + 10) = 2{,}660$$

CP rank	Relative reconstruction error	Decomposition time (s/sample)
5	0.015752 ± 0.004517	0.269
10	0.008215 ± 0.002756	0.264
20	0.004580 ± 0.002539	7.444
40	0.001461 ± 0.000724	7.533

Table 8: Approximation quality and offline decomposition time for different CP ranks on the PDEBench shallow-water input histories.

Method	Relative ℓ_2 error	RMSE	Time (s)
CCP-Conv	0.023163	0.024467	**17.18**
Dense Conv	**0.023136**	**0.024442**	164.11

Table 9: One-step prediction results on the PDEBench shallow-water dataset using rank-10 CP inputs for CCP-Conv. The dense model receives the corresponding original dense histories.

factor values. The input representation is therefore approximately $61.6\times$ smaller in terms of stored scalar values. The measured CP decomposition time is an offline preprocessing cost and is not included in the reported model training times.

Prediction models. The CCP-Conv predictor uses one unrestricted convolutional filter with kernel shape $(5, 5, 3)$ and learned mode-wise compression to an output shape of $(10, 10, 5)$. The resulting representation is flattened and passed to an MLP with hidden width 256, whose output is reshaped to the 128×128 predicted state.

The matched dense predictor uses the original dense $128 \times 128 \times 10$ history as input. It applies a conventional dense convolution with the same kernel size, followed by the same output-compression dimensions and downstream MLP. The two models therefore differ primarily in whether the convolutional front-end operates on the dense input or directly on its approximate CP representation.

Both models are trained for $1{,}000$ optimization steps using a batch size of 16 and a learning rate of 10^{-3}. Training minimizes the relative squared prediction error

$$\mathcal{L} = \frac{\|\widehat{Y} - Y\|_F^2}{\|Y\|_F^2}.$$

The experiment was performed with one training seed and is used primarily as a matched computational comparison rather than as a benchmark of state-of-the-art shallow-water forecasting.

Prediction results. Table 9 reports the test relative ℓ_2 error, RMSE, and elapsed training time.

CCP-Conv closely matches the prediction error of the corresponding dense model, with a relative ℓ_2 error of 0.023163 compared with 0.023136 for dense convolution. Its elapsed training time is reduced from 164.11 s to 17.18 s, corresponding to a $9.55\times$ reduction. This result indicates that the computational advantage of CCP-Conv is retained even when the CP representation is obtained approximately from initially dense nonlinear PDE data.

Qualitative comparison. Figure 2 shows a representative held-out trajectory. The displayed example is selected so that its CCP-Conv relative error is closest to the median CCP-Conv error over the test set. The ground truth, CCP-Conv prediction, and dense-model prediction are shown using a shared color scale.

A.6 R3D-18 Experimental Setup

To assess compatibility with a conventional deep video architecture, we integrated CCP-Conv into a Kinetics-400-pretrained R3D-18 (Tran et al., 2018). We used the complete 101-class UCF101 setting with the same fixed train/test split as in the preceding UCF101 experiments, containing $10{,}160$ training and $3{,}160$ test videos. The input videos were represented in grayscale using CP

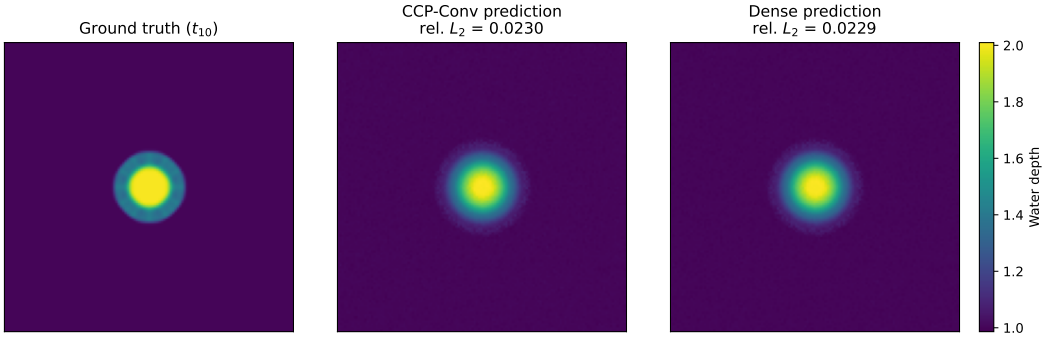


Figure 2: Representative one-step prediction on the PDEBench shallow-water dataset. From left to right, the panels show the true next state, the CCP-Conv prediction using a rank-10 approximate CP input, and the prediction of the matched dense model. All panels use the same color scale.

rank $R = 300$. Both the CCP and dense branches used the same preprocessing and data augmentation. Inputs were resized to $16 \times 112 \times 112$ in (T, H, W) order and normalized using the grayscale equivalent of the Kinetics-400 normalization. During training, random cropping with a minimum retained scale of 0.8 and horizontal flipping with probability 0.5 were applied. The corresponding transformations were performed directly on the CP factors for the CCP branch. A stratified 10% subset of the training split was used for validation.

The pretrained R3D-18 convolutional stem was replaced by either a CCP-Conv stem or its mathematically equivalent dense convolution–compression counterpart. In both cases, the pretrained RGB stem kernel was converted to a single-channel kernel by averaging over the RGB input channels. The stem first performs a stride-one three-dimensional convolution and then applies trainable separable projections along the temporal and spatial modes, reducing the feature-grid resolution from $16 \times 112 \times 112$ to $16 \times 56 \times 56$, followed by BatchNorm3D and ReLU. For the dense baseline, the full convolutional output is explicitly materialized before projection, whereas CCP-Conv pushes the same projections inside the CP contraction and directly produces the compressed feature grid. The projection matrices are initialized as deterministic selection operators corresponding to the sampling pattern of the original R3D-18 stride-$(1, 2, 2)$ stem, and are subsequently trainable. Thus, at initialization, the stride-one convolution followed by these projections is equivalent to applying a stride-$(1, 2, 2)$ convolution with the same grayscale-adapted kernel. All downstream R3D-18 blocks are identical between the two models.

We fine-tuned the modified stem, the final residual stage of R3D-18, and the classification head, while keeping the earlier residual stages frozen. Training was performed for 30 epochs with batch size 4 using AdamW and a cosine-annealing learning-rate schedule. Learning rates of 10^{-4}, 2×10^{-5}, and 10^{-3} were used for the stem, final residual stage, and classification head, respectively, with weight decay 10^{-4}. Cross-entropy loss with label smoothing 0.1 was used, together with gradient clipping at 5 and mixed-precision training. Results were averaged over three random seeds. The best checkpoint for each seed was selected according to validation balanced accuracy. Training times were measured using the same protocol as in the preceding supervised experiments and exclude the one-time offline CP decomposition.

A.7 Extension to Multi-Layer Architectures

In the experiments presented in this work, CCP-Conv is used primarily as a compressed front end. The first convolution operates directly on the CP representation of the input, after which the resulting compact dense tensor can be passed to conventional downstream layers. This design makes CCP-Conv easy to integrate into existing architectures without requiring modifications to the remainder of the network. However, it does not exploit tensor structure throughout the full depth of the model.

Extending this principle to multiple layers is a natural direction for future work. One possibility is to periodically recompress intermediate feature maps, for example by computing a new low-rank tensor decomposition after one or several conventional layers and subsequently applying another CP-aware

convolutional block. Such recompression would generally introduce an additional approximation and decomposition cost, creating a trade-off between representation fidelity and computational savings. More generally, architectures could alternate between factorized and dense representations, with recompression performed only when the reduction in memory or computational cost justifies its overhead. Another possibility is to design layers that propagate a structured tensor representation directly between successive convolutions, together with factor-aware nonlinearities, normalization operations, or low-rank projection steps that control rank growth. Such approaches would allow the advantages of CP-based computation to extend beyond the input layer while avoiding repeated materialization of large intermediate tensors. Developing stable and efficient strategies for maintaining, adapting, or restoring low-rank structure across deep nonlinear networks remains an interesting direction for future research.

A.8 Unsupervised Learning Details

Experimental setting. The unsupervised experiments use the complete 101-class UCF101 dataset and the 15-class lower-resolution Ego4D dataset described in Appendix A.3. The same preprocessed tensors, CP representations, ranks, and train/test splits as in the corresponding supervised experiments are used. Five random seeds are considered for each experiment. The data partition is fixed across seeds, while the network initialization and minibatch sampling vary. No class information is provided to the autoencoder during training.

CCP autoencoder architecture. Let the input be represented in CP form as

$$X_{\mathrm{CP}}(i_1, \ldots, i_d) = \sum_{r=1}^{R} \prod_{\ell=1}^{d} U_\ell^r(i_\ell).$$

The encoder first applies CCP-Conv directly to the CP factors. For the video experiments, the convolutional kernel has size

$$5 \times 5 \times 3,$$

and learned mode-wise projection matrices compress the convolutional output to a tensor of size

$$3 \times 3 \times 3.$$

The compressed tensor is passed to a convolutional encoder consisting of a single convolutional stage with 32 channels, kernel size 3, stride 2, and padding 1, followed by an MLP mapping to a latent representation of dimension 2000. ReLU nonlinearities are used throughout the hidden layers.

The decoder first maps the latent vector through an MLP followed by a transposed-convolutional decoder with channel widths

$$32,\ 64,\ 128,$$

using kernel size 3, stride 2, padding 1, and output padding 1. The resulting compact tensor is then passed to a CP-factor decoder with hidden width 128 and depth 2. This decoder predicts a set of CP factors

$$\left\{\widehat{U}_\ell^r\right\}_{\ell=1,\ldots,d,\, r=1,\ldots,R},$$

whose output rank is chosen to match the rank of the input representation. These factors define the reconstructed tensor

$$\widehat{X}_{\mathrm{CP}}(i_1, \ldots, i_d) = \sum_{r=1}^{R} \prod_{\ell=1}^{d} \widehat{U}_\ell^r(i_\ell).$$

Neither the input nor the reconstructed tensor needs to be materialized densely during training.

Training protocol. The autoencoder is optimized using AdaBelief with a constant learning rate of 10^{-3} and batch size 5. For both UCF101 and Ego4D, we train the autoencoder for 30 epochs.

The five runs use seeds $0, \ldots, 4$. Model initialization and minibatch sampling use seed-dependent random keys, while the underlying train/test partition remains fixed. No additional input or output normalization is applied to the stored CP factors.

Dataset	Trained AE (%)	Random-init AE (%)	Chance (%)
UCF101	8.09 ± 0.98	4.94 ± 1.40	0.99
Ego4D	15.25 ± 4.27	9.84 ± 1.78	6.67

Table 10: Frozen-latent classification accuracy using a regularized linear Ridge classifier. Values are mean $\pm$ standard deviation over five seeds. The random-initialization control evaluates the corresponding encoder before reconstruction training using the same frozen-latent probe.

Dataset	CCP time (s)	Dense AE projected time (s)
UCF101	4303 ± 42	$\approx 5.28 \times 10^6$
Ego4D	1167 ± 100	$\approx 2.75 \times 10^6$

Table 11: Unsupervised autoencoder training times. CCP values correspond to complete training runs and are reported as mean $\pm$ standard deviation over five seeds. Dense-autoencoder values are projections obtained from short profiling runs and are included only to indicate the computational regime. They are not complete end-to-end measurements.

Reconstruction objective. The autoencoder is trained using the mean relative squared Frobenius reconstruction error

$$\mathcal{L}_{\mathrm{rec}} = \frac{1}{B} \sum_{b=1}^{B} \frac{\left\| \widehat{X}_b - X_b \right\|_F^2}{\|X_b\|_F^2 + \epsilon}, \qquad \epsilon = 10^{-8}.$$

This quantity is evaluated directly from the CP factors. In particular, for two tensors represented by CP factors $\{U_\ell^r\}$ and $\{V_\ell^s\}$, their inner product can be computed as

$$\langle X, V \rangle = \sum_r \sum_s \prod_{\ell=1}^{d} \langle U_\ell^r, V_\ell^s \rangle .$$

Consequently,

$$\left\| \widehat{X} - X \right\|_F^2 = \left\| \widehat{X} \right\|_F^2 + \|X\|_F^2 - 2\langle \widehat{X}, X \rangle$$

can be evaluated without dense reconstruction. The reconstruction objective is the only loss used during autoencoder training.

Frozen latent-space evaluation. After reconstruction training, all autoencoder parameters are frozen and the 2000-dimensional latent representations are extracted for the training and test sets. A regularized linear Ridge classifier is then fitted using only the training-set latent representations and their corresponding class labels. Classification accuracy is evaluated on the held-out test latents. Labels are therefore used only for the downstream frozen-latent probe and are never provided to the autoencoder during representation learning.

To assess how much class information is induced by reconstruction training, we additionally evaluate the corresponding randomly initialized encoder before any autoencoder optimization. For each seed, the random-initialization control uses the same architecture, initialization seed, CP inputs, and Ridge evaluation procedure as the trained model. The input CP representations themselves are unchanged. The control therefore measures the linear class information available from the randomly initialized encoder rather than from randomized input data.

Reconstruction training increases the frozen-latent classification accuracy from 4.94% to 8.09% on UCF101 and from 9.84% to 15.25% on Ego4D. These results indicate that reconstruction training makes additional class information linearly accessible relative to the corresponding randomly initialized encoder, without using class labels during autoencoder training.

Dense autoencoder profiling. For comparison, we implemented a conventional autoencoder operating on the corresponding dense video tensors. Its encoder uses a conventional dense convolution followed by the same form of learned mode-wise projection used to obtain the compact $3 \times 3 \times 3$ representation, after which the same 32-channel convolutional encoder and 2000-dimensional latent

space are used. The decoder maps the latent vector directly back to the full dense video using the transposed-convolutional decoder, rather than predicting CP factors.

Executing the complete dense-autoencoder training budget was impractical. We therefore profiled five optimization steps using the same batch size, optimizer, learning rate, and timing convention as in the CCP experiments, and extrapolated the measured per-step cost to the full training budget. First-step compilation overhead was checked separately and was negligible relative to the measured optimization steps, so it did not materially inflate the extrapolated times.

The resulting projected times are approximately

$$5.28 \times 10^6 \text{ s}$$

for UCF101 and

$$2.75 \times 10^6 \text{ s}$$

for Ego4D, corresponding to approximately 61 and 32 days, respectively. These projections are intended only to establish the computational regime and should not be interpreted as measured end-to-end training times or precise runtime benchmarks. For this reason, no dense-autoencoder representation-learning accuracy is reported.

A.9 Effect of CP Representation Rank

The CP rank R is a property of the input representation rather than a hyperparameter introduced by CCP-Conv. In applications where the CP factors are obtained from an initially dense tensor, the rank can be selected upstream using standard low-rank approximation criteria, for example by prescribing a target reconstruction error, approximation tolerance, or storage budget. Likewise, when data are generated directly in factorized form, the available rank is determined by the underlying representation or generation procedure. The purpose of the following experiments is therefore not to propose a task-specific rank-selection strategy, but to quantify how CCP-Conv behaves when supplied with CP representations of different ranks and fidelities.

We investigate this effect on both the shallow UCF101 10-class experiment and the UCF101 experiment using R3D-18.

For the shallow model, test accuracy remains remarkably stable over a wide range of representation ranks. Increasing R from 10 to 400 yields accuracies between 30.0% and 32.0%, compared with the 10% chance baseline. Training time, however, increases from 4.47 ± 0.12 s at $R = 10$ to 78.38 ± 7.60 s at $R = 400$. This indicates that, for this task, even relatively compact CP representations already preserve most of the information required for classification, so increasing the representation fidelity beyond this point provides little additional predictive benefit while increasing computational cost. Note that we documented the results for $R = 150$ in the main text, which is representative of the entire range of ranks considered.

A different behavior is observed when CCP-Conv is integrated into the deeper R3D-18 architecture. Increasing the input-representation rank from $R = 100$ to $R = 220$ and $R = 300$ improves test accuracy from $77.0 \pm 1.0\%$ to $83.0 \pm 0.4\%$ and $84.8 \pm 0.5\%$, respectively. At $R = 300$, the mean accuracy is comparable to that of the dense baseline ($84.2 \pm 5.1\%$), while the measured training time remains lower (265 ± 2.3 s versus 279.6 ± 3.4 s).

These results show that CCP-Conv does not require a specific CP rank in order to operate. Rather, its behavior reflects the fidelity and compactness of the input representation supplied to it. When a low-rank representation already preserves the task-relevant information, as in the shallow UCF101 experiment, performance is largely insensitive to further increases in rank. When greater representation fidelity is beneficial, as in the deeper R3D-18 setting, performance improves with rank and eventually approaches that of the dense input. In both cases, the computational cost increases with the size of the factorized representation, as expected.

A.10 Offline Cost of CP Decomposition

The proposed method operates directly on tensors represented in CP form. Importantly, the CP decomposition itself is *not* part of the proposed convolutional operator. When the input data are already available in factorized form, as is often the case in tensor-based numerical methods, low-rank

Table 12: One-time CP preprocessing cost for datasets originally available in dense form. Timing is reported per sample. For all datasets, the timing column reports the median and interquartile range (IQR), while the reconstruction column reports mean $\pm$ standard deviation of the relative reconstruction error over 10 samples.

Dataset	Input tensor size	CP rank	Time / sample (s)	Relative error
Synthetic	$64 \times 64 \times 32$	20	0.097 (0.065–0.113)	0.0226 ± 0.013
UCF101	$128 \times 96 \times 16$	150	10.944 (10.700–11.348)	0.0609 ± 0.028
Ego4D low-res	$128 \times 96 \times 24$	200	2.129 (1.721–2.383)	0.0982 ± 0.032
Ego4D high-res	$1920 \times 2560 \times 128$	320	1519 (1510–1529)	0.2632 ± 0.052

simulation pipelines, or data generated directly using separated representations, no decomposition step is required at all. In such settings, CP-Conv and CCP-Conv can be applied directly to the available factors, without ever constructing the corresponding dense tensor.

For datasets that are instead provided initially in dense form, we compute a CP representation once as an offline preprocessing step and reuse the resulting factors throughout training. This preprocessing cost is excluded from all training-time comparisons reported in the main text. The purpose of the measurements reported here is therefore not to characterize the cost of our convolutional method, but rather to quantify the additional one-time preprocessing cost incurred in the particular experiments for which a dense dataset had to be converted to CP form.

Because each sample can be decomposed independently, CP preprocessing can be parallelized across samples. The wall-clock time required to convert an entire dataset consequently depends strongly on the available parallelism. Our preprocessing implementation was executed sequentially and was not optimized for throughput. We therefore report the decomposition time per sample rather than the total preprocessing time of each dataset. These numbers should thus be interpreted as indicative measurements of the one-time preprocessing cost rather than as benchmarks of CP decomposition algorithms. Dataset generation, video decoding and resizing, and disk I/O are excluded from the reported timings.

Table 12 summarizes the measured costs. For the synthetic dataset, tensors of size $64 \times 64 \times 32$ were approximated with rank 20. For UCF101, the preprocessed video tensors have size $128 \times 96 \times 16$ and were decomposed with rank 150. The low-resolution Ego4D experiment uses grayscale tensors of size $128 \times 96 \times 24$ with rank 200. For the high-resolution Ego4D experiment, each grayscale sample has size $1920 \times 2560 \times 128$ and is represented using a greedy blockwise decomposition of total rank 320.

The PDEBench experiment provides an additional example in which the data are initially dense and are approximately factorized offline. Each input history has size $128 \times 128 \times 10$ and is represented using rank $R = 10$. This yields a relative reconstruction error of 0.008215 ± 0.002756 and a measured decomposition time of approximately 0.264 s per sample. Additional rank-dependent decomposition results are reported in Appendix A.5.

The preprocessing cost varies substantially with tensor dimensions, target rank, and decomposition procedure. In particular, the high-resolution Ego4D example illustrates that obtaining a high-rank CP approximation of a very large dense tensor can itself be expensive. This cost, however, should be distinguished from the computational cost of the proposed convolution. Once the factors have been obtained, the dense input is no longer required during training or inference, and the same factorization can be reused for all subsequent passes through the network.

More importantly, this preprocessing overhead is absent whenever the data are natively available in CP form. This is precisely the case for the Fokker–Planck experiments considered in this work. The distributions are generated directly as sums of separable components and therefore admit an exact CP representation without first forming and decomposing the corresponding dense tensor. The proposed method is thus not predicated on CP decomposition being an efficient compression procedure for arbitrary dense data. Rather, it provides a way to exploit a CP representation whenever such a representation is already available or can be obtained as part of the upstream data-generation pipeline.

Table 13: Comparison with alternative compact input representations on the 10-class UCF101 experiment. All alternatives are chosen to approximately match the 36,000 scalars per sample required by the rank-150 CP representation. CCP-Conv operates directly on the CP factors, whereas the alternative representations are processed using conventional dense convolution. Training times exclude the one-time offline preprocessing cost.

Method	Stored scalars	Storage / CP	Test acc. (%)	Train time (s)
CCP-Conv	36,000	1.000	31.6 ± 2.0	$\mathbf{39.7 \pm 4.4}$
Spatiotemporal resize	36,864	1.024	31.9 ± 4.0	318.2 ± 5.3
Temporal subsampling	36,864	1.024	30.5 ± 3.5	140.3 ± 0.3
Patch pooling	36,864	1.024	$\mathbf{33.1 \pm 2.0}$	315.0 ± 0.3
Truncated SVD	36,915	1.025	31.7 ± 3.0	1906.3 ± 11.3

For completeness, the CP decompositions reported in Table 12 were computed using the TensorLy library (Kossaifi et al., 2019) with the alternating least-squares algorithm and default parameters.

A.11 Comparison with Alternative Compact Input Representations

To distinguish the benefit of CCP-Conv from that of input compression alone, we compared CCP-Conv with several simple alternatives that reduce the number of stored scalars per input while retaining a conventional dense convolutional operator. We considered the 10-class UCF101 experiment and matched the storage of each alternative as closely as possible to the rank-150 CP representation used by CCP-Conv. For an input of size $128 \times 96 \times 16$, the CP factors require

$$150(128 + 96 + 16) = 36{,}000$$

scalars per sample.

We considered four alternative representations. *Spatiotemporal resizing* interpolates each video to $64 \times 48 \times 12$. *Temporal subsampling* retains three approximately uniformly spaced full-resolution frames, resulting in an input of size $128 \times 96 \times 3$. *Patch pooling* performs deterministic average pooling to obtain a $64 \times 48 \times 12$ representation. Finally, *truncated matrix SVD* matricizes each video as $(HW) \times T$ and retains the rank whose factor storage is closest to that of the CP representation. In the present setting this gives rank 3, requiring 36,915 scalars per sample.

For all four alternatives, the resulting input is processed using the same conventional dense-convolution classifier and training protocol as the dense baseline in Sec. 3.2. For the SVD representation, the truncated factors are reconstructed into a dense tensor before convolution. This SVD experiment should therefore be interpreted as a storage-matched reconstruction baseline rather than as the strongest possible factor-aware SVD convolutional method. Offline preprocessing time is not included in the reported training time. Results are averaged over five random seeds and are reported in Table 13.

The different compact representations achieve similar predictive performance, with mean test accuracies between 30.5% and 33.1%. In particular, the experiment does not indicate that the CP representation is intrinsically more accurate than the alternative reductions at this storage level. The principal distinction instead appears in computational cost. Despite requiring essentially the same number of stored scalars, spatiotemporal resizing and patch pooling are approximately $8\times$ slower to train than CCP-Conv, while temporal subsampling is approximately $3.5\times$ slower.

The SVD experiment further illustrates the distinction between a compact *representation* and a computation that directly exploits that representation. The truncated SVD requires almost exactly the same storage as the CP factors, but in the baseline considered here its factors are reconstructed into a dense tensor before conventional convolution. This results in a training time of approximately 1906 s, about $48\times$ that of CCP-Conv. This comparison does not rule out specially designed factor-aware operators for SVD representations. Rather, it shows that compact factor storage alone does not automatically reduce the cost of a conventional dense convolution. CCP-Conv obtains its computational advantage by carrying out the convolution directly in the structured CP representation without first reconstructing the dense tensor.

Table 14: Effect of replacing the learned mode-wise projection matrices in CCP-Conv with fixed adaptive average pooling. Test accuracy is reported as mean ± standard deviation over five random seeds.

Dataset	Learned projection (ours) Test acc. (%)	Average pooling Test acc. (%)
Synthetic Medium	$\mathbf{85.0 \pm 7.0}$	54.5 ± 1.2
Synthetic Hard	$\mathbf{73.0 \pm 12.0}$	48.3 ± 1.9
UCF101, 10 cls.	$\mathbf{31.6 \pm 2.0}$	31.0 ± 2.6

For completeness, the average one-time preprocessing costs per sample were approximately 1.21 ms for spatiotemporal resizing, 0.22 ms for temporal subsampling, 1.07 ms for patch pooling, and 6.91 ms for truncated SVD, compared with approximately 10.94 s per sample for CP decomposition. Although obtaining the CP representation is more expensive when starting from dense data, this preprocessing is performed only once and can be parallelized independently across samples. This cost is absent when the input is already available natively in CP form.

These comparisons highlight that the main advantage of CCP-Conv is not compression alone, but the ability to perform convolution directly on the factorized representation. Resizing, temporal subsampling, and pooling reduce the size of the dense input but still rely on conventional dense convolution. The SVD baseline uses a factorized representation for storage but reconstructs it before convolution. In contrast, CCP-Conv performs the convolution directly on the CP factors and avoids dense reconstruction. This distinction is particularly important for high-dimensional inputs, where the dense representation may be too large to materialize at all.

A.12 Learned Projection versus Fixed Pooling

The CCP-Conv formulation combines convolution on the CP factors with a mode-wise reduction of the convolutional output. In the main experiments, this reduction is implemented using trainable projection matrices. A natural question is whether the observed performance depends specifically on learning these projections, or whether CCP-Conv can also be combined with a conventional fixed pooling operation.

To investigate this, we replace every learned mode-wise projection matrix by a fixed adaptive average-pooling operator while leaving the remainder of the architecture unchanged. In particular, we use the same dense convolutional kernel, CP input representation, output compression dimensions, classifier, optimization procedure, and number of training updates. The pooling operation is represented as a fixed linear map along each tensor mode and can therefore be incorporated directly into the CCP-Conv contractions without materializing the dense convolutional output.

Table 14 compares the resulting classifier with the learned-projection CCP-Conv used in the main experiments. Results are reported over five random seeds.

The synthetic experiments indicate that learning the compression operators can be beneficial when the shallow classifier must retain a relatively specific discriminative signal through a small compressed representation. Replacing the learned projections by uniform average pooling reduces test accuracy from 85.0% to 54.5% on the Medium dataset and from 73.0% to 48.3% on the Hard dataset. Thus, the trainable projections provide useful flexibility beyond that of a fixed local averaging scheme in these controlled problems.

Importantly, this behavior is not universal. On the 10-class UCF101 experiment, fixed adaptive average pooling obtains $31.0 \pm 2.6\%$ test accuracy, compared with $31.6 \pm 2.0\%$ for the learned-projection variant. The difference is small relative to the variation across random seeds. Hence, on this real video dataset, the predictive performance of CCP-Conv is essentially preserved even when the learned compression matrices are replaced by a parameter-free pooling operation.

The choice of reduction operator also affects the spatial inductive bias. The learned projection matrices assign trainable weights to absolute output coordinates and are therefore not generally translation equivariant. Fixed pooling avoids such learned location-specific weights and can preserve a stronger translation-equivariant structure, subject to the usual effects of pooling stride, binning, and

boundary handling. This provides a useful alternative when preserving spatial equivariance is more important than the additional flexibility of learned projections.

These results demonstrate that the computational formulation of CCP-Conv is not tied to learned projections. Any suitable linear mode-wise reduction can be fused with the convolution on the CP factors. Learned projections provide additional adaptivity and can substantially improve performance in some settings, while fixed average pooling provides a simpler parameter-free alternative that performs comparably on UCF101.