Unsupervised Motion Representation Learning with Capsule Autoencoders

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose the Motion Capsule Autoencoder (MCAE), which addresses a key challenge in the unsupervised learning of motion representations: transformation invariance. MCAE models motion in a two-level hierarchy. In the lower level, a spatio-temporal motion signal is divided into short, local, and semantic-agnostic *snippets*. In the higher level, the snippets are aggregated to form full-length semantic-aware *segments*. For both levels, we represent motion with a set of learned transformation invariant templates and the corresponding geometric transformations by using capsule autoencoders of a novel design. This leads to a robust and efficient encoding of viewpoint changes. MCAE is evaluated on a novel Trajectory20 motion dataset and various real-world skeleton-based human action datasets. Notably, it achieves better results than baselines on Trajectory20 with considerably fewer parameters and state-of-the-art performance on the unsupervised skeleton-based action recognition task.

1 Introduction

Real-world movements contain a plethora of information beyond the literal sense of moving. For example, honeybees “dance" to communicate the location of a foraging site and human gait alone can reveal activities and identities [6]. Understanding these movements is vital for an artificial intelligent agent to comprehend and interact with the ever-changing world. Studies on social behavior analysis [4, 5], action recognition [50, 55], and video summarizing [51] have also acknowledged the importance of movement.

A key step towards understanding movements is to analyze its patterns. However, learning motion pattern representations is non-trivial due to (1) the curse of dimensionality from input data, (2) difficulties in modeling long-term dependencies in motion sequences, (3) high intra-class variation as a result of subject or viewpoint change, and (4) insufficient data annotation. The first two challenges have been ameliorated by the advances in keypoint detection methods and spatial-temporal feature extractors [32, 37, 43]. The third and the fourth nonetheless remain hurdles and call for unsupervised transformation-invariant motion models.

Inspired by the viewpoint-invariant capsule-based representation for images [7, 11], we exploit capsule network and introduce the Motion Capsule Autoencoder (MCAE), an unsupervised capsule framework that learns the transformation-invariant motion representation for keypoints. MCAE models motion signals in a two-level snippet-segment hierarchy. At the lower level, motion signal is encoded as the snippet capsules (SniCap) that describe movement in narrow time spans (i.e. snippets). The motion snippets are then temporally concatenated to form movement with wider time spans (i.e. segments). At the higher level, the segment capsules (SegCap) maintain a set of segment templates (i.e. learned canonical motion patterns) and transform them to reconstruct the input segment motion. Both SniCaps and SegCaps learn transformation-invariant motion representation in their temporal receptive field. The SegCaps, which are built upon SniCaps, produce a high-level abstraction of the given motion signal, where the semantics are represented by transformation-invariant capsule activations.

The contributions of this work are as follows:

- We propose MCAE, an unsupervised capsule framework that learns a transformation-invariant, discriminative, and compact representation of motion signals. Two motion capsules are designed to generate representation at different abstraction levels. The lower-level representation captures the local short-time movements, which are aggregated into higher-level representation that is discriminative for motion with wider time spans.
- We propose Trajectory20, a novel and challenging synthetic dataset with a wide class of motion patterns and controllable intra-class variations.
- Extensive experiments on both the synthetic and real-world skeleton human action datasets show the efficacy of MCAE. In addition, we perform an ablation study to examine the effect of different regularizers and some key hyperparameters of the proposed MCAE.

2 Related Works

Motion Representation A variety of methods have been proposed to learn (mostly human) motion representation from video frames [1, 17, 22, 46], depth maps [8, 16, 21, 31, 41], or keypoints/skeletons [2, 15, 18, 20, 23, 27, 35, 45, 48, 49]. Earlier works use handcrafted features like Fourier coefficients [41], dense trajectory features [24, 40], and Lie group representations [38]. Some works use canonical human pose [26] or view-invariant short tracklets to learn robust feature for recognition [13]. The development of deep learning brings the usage of convolution networks (ConvNet) and recurrent networks for motion representation. Simonyan *et al.* [33] proposes a two-stream ConvNet which combines video frame with optical flow. C3D [37] proposes to use 3D convolution on the spatial-temporal cubes. Srivastava *et al.* [34] uses an LSTM-based encoder to map input frames to a fixed-length vector and apply task-dependent decoders for applications such as frame reconstruction and frame prediction. The combined use of convolution module and LSTM has also been proved effective in [1, 32, 44].

A series of works [9, 12, 14, 39, 42] have been proposed to address the problem of learning viewpoint-invariant motion representation from videos or keypoint sequences. Most of these methods rely on multi-modality input of RGB frames, depth maps or keypoint trajectories. Some other works [18, 27, 35, 54] focus on the unsupervised learning of keypoint/skeleton motion. In these works, LSTM is widely used for motion modelling which generally results in a heavy memory footprint.

Different from prior works, MCAE takes only keypoint motion as input and uses LSTM to model snippet-segment relations only. This results in a lightweight yet capable model.

Capsule Network MCAE is closely related to the Capsule Network [28], which is designed to represent objects in images using automatically discovered constituent parts and poses. The explicit modeling of poses helps learning viewpoint-invariant visual features that are more compact and flexible than traditional ConvNets. Kosiorek *et al.* [11] proposed the unsupervised stacked capsule autoencoder (SCAE), which learns view-invariant representation for images. More recently, capsule network has also been applied to point cloud processing [52, 53] for 3D object classification and reconstruction.

Despite the success of capsule networks in various vision tasks, the study of capsule networks on motion representation is scarce. VideoCapsuleNet [3] proposes to generalize capsule networks from 2D to 3D for action detection in videos. Yu *et al.* [47] proposed a limited study on supervised skeleton-based action recognition using Capsule Network. Sankisa *et al.* [30] proposed to use Capsule Network for error concealment in videos.

Different from these works, MCAE performs unsupervised learning of motion represented as coordinates rather than pixels. It aims at learning an appearance-agnostic transformation-invariant motion representation.

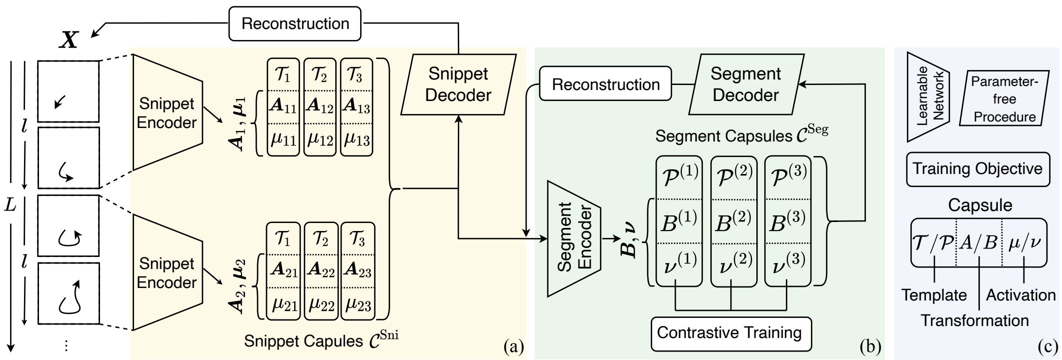


Figure 1: Overview of MCAE (best viewed in color). (a) The Snippet Autoencoder, which learns the semantic-agnostic short-time representation (snippet capsules) by reconstructing the input signal $\boldsymbol{X}$. (b) The Segment Autoencoder, which learns the semantic-aware long-time representation (segment capsules) by aggregating and reconstructing snippet capsule parameters. The activation values in segment capsules are used as semantic information for self-supervised contrastive training. (c) Meanings for different shapes and variables.

3 Methodology

We consider a single point[1] in d-dimension space. The motion of the point, i.e. a trajectory, is described by $\boldsymbol{X} = \{\boldsymbol{x}_i | i = 1, \ldots, L\}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is the coordinates at time i. Semantically, $\boldsymbol{X}$ belongs to a motion pattern, subject to an arbitrary and unknown geometric transformation. Given sufficient samples of $\boldsymbol{X}$, we aim to learn a discriminative (in particular, transformation-invariant) representation for those motion samples without supervision.

3.1 Framework Overview

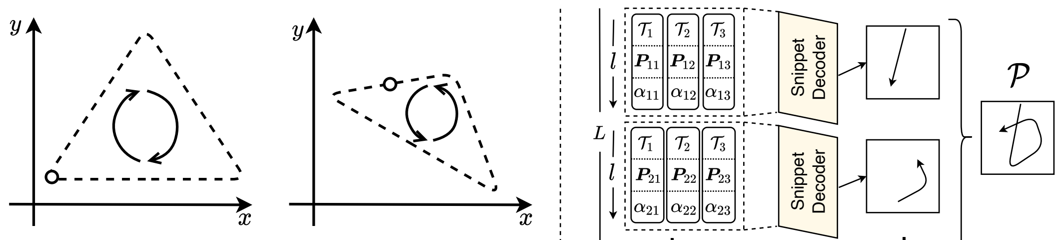
We solve this problem in two steps, namely *snippet learning* and *segment learning*. Snippets and segments correspond to the lower and higher levels in the hierarchy of how MCAE views the motion signal. Both snippets and segments are temporally consecutive subsets of $\boldsymbol{X}$, but snippets have a shorter time span than segments. In the snippet learning step, the input $\boldsymbol{X}$ is first divided into L/l temporally non-overlapping snippets, where l is the length of snippets. Each of these snippets will be mapped into a semantic-agnostic representation by the **Snippet Autoencoder**. In the segment learning step, the snippet representations are combined and fed into the **Segment Autoencoder**, where the full motion is represented as a weighted mixture of the transformed canonical representations. The segment activations are used as the motion representation for downstream tasks. An overview of the framework is shown in Fig. 1. In the following section, we delineate the details for each module and explain the training procedure.

3.2 Snippet Autoencoder

To encode the snippets' motion variation, we propose the Snippet Capsule (SniCap), which we denote as $\mathcal{C}^{\mathtt{Sni}}$. SniCap is parameterized as $\boldsymbol{C}^{\mathtt{Sni}} = \{\mathcal{T}, \boldsymbol{A}, \mu\}$, where $\mathcal{T}$, $\boldsymbol{A}$, and μ are the **snippet template**, **snippet transformation parameter**, and **snippet activation**, respectively. The snippet template $\mathcal{T} = \left\{\boldsymbol{t}_i | \boldsymbol{t}_i \in \mathbb{R}^d, i = 1, ..., l\right\}$ describes a motion template of length l and is the identity information of a SniCap. $\boldsymbol{A}$ and μ depend on the input snippet. The transformation parameter $\boldsymbol{A} \in \mathbb{R}^{(d+1)\times(d+1)}$ descries the geometric relation between the input snippet and the snippet template. The snippet activation $\mu \in [0, 1]$ denotes whether the snippet template is used, i.e. activated, to represent the input snippet.

Snippet Encoding/Decoding For a given snippet $\boldsymbol{x}_{i:i+l}$, the snippet module performs the following steps: (1) encode motion properties with Snippet Encoder into SniCaps, and (2) decode SniCaps to reconstruct the original $\boldsymbol{x}_{i:i+l}$. For the encoding step, a 1D-ConvNet f_{CONV} is used to extract

[1] We show a way to generalize MCAE to multi-point systems in Section 4.2



(a) A reference pattern. (b) A variant of (a). (c)

Figure 2: (a) and (b) show a reference motion pattern and a variant of it. The circle and the arrow shows the start and the direction of motion respectively. (c) Interpretation of a segment template $\mathcal{P}$. $\mathcal{P}$ is functionally the same as S snippet parameters $(\boldsymbol{A}, \boldsymbol{\mu})$. When combined with $\mathcal{T}$, it can be decoded into an L-long sequence. The segment autoencoder maintains multiple segment templates, which can be transformed and mixed to reconstruct the input snippet parameters.

the motion information from $\boldsymbol{x}_{i:i+l}$ and predict SniCap parameters, i.e. $\{(\boldsymbol{A}_j, \mu_j) | j = 1, \ldots, N\} = f_{\texttt{CONV}}(\boldsymbol{x}_{i:i+l})$ where N is the number of SniCaps. For the decoding step, we first apply the transformation $\boldsymbol{A}$ to the snippet templates as

$$\begin{pmatrix} \hat{\boldsymbol{t}}_{ij} \\ 1 \end{pmatrix} = \boldsymbol{A}_i \begin{pmatrix} \boldsymbol{t}_j \\ 1 \end{pmatrix}, \quad i = 1, \ldots, N, \quad j = 1, \ldots, l. \tag{1}$$

Then, the transformed templates from different SniCaps are mixed, according to their activations, and the corresponding reconstructed input is

$$\hat{\boldsymbol{x}}_j = \sum_{i=1}^{N} \mu_i \hat{\boldsymbol{t}}_{ij}, \quad j = 1, \ldots, l, \tag{2}$$

where $\hat{\boldsymbol{t}}_{ij}$ indicates the transformed coordinate of the i^{th} SniCap at j^{th} time step.

3.3 Segment Autoencoder

The motion information encoded in SniCaps is agnostic to the segment level motion patterns. This makes it less biased towards the training data domain. However, its utility on high-level applications, such as activity analysis or motion classification, is greatly undermined. For example, consider Fig. 2(a) as a reference "triangle" trajectory. Fig. 2(b) illustrates a possible variation. Since the two trajectories differ a lot in their local movement, they could be considered as different classes without transformation-invariant information from the full trajectory.

Hence, we introduce a segment encoder to gain a holistic understanding of motion and encapsulate such information in the segment capsules (SegCap). A SegCap is parameterized as $\boldsymbol{C}^{\texttt{Seg}} = \{\mathcal{P}, \boldsymbol{B}, \nu\}$, where $\mathcal{P}$, $\boldsymbol{B}$, and ν are the **segment template**, **segment transformation parameter**, and **segment activation**, respectively. The segment template $\mathcal{P}$ is fixed for a SegCap w.r.t the training domain. It describes a set of canonical motion patterns in terms of all the snippet templates and is defined as $\mathcal{P} = \{(\boldsymbol{P}_i, \boldsymbol{\alpha}_i) \mid i = 1, \ldots, S\}$, where $\boldsymbol{P}_i \in \mathbb{R}^{N \times (d+1) \times (d+1)}$ and $\boldsymbol{\alpha}_i \in \mathbb{R}^N$. $S = L/l$ is the number of snippets. Each $\boldsymbol{P}_{ij} \in \mathbb{R}^{(d+1) \times (d+1)}$ describes how the j^{th} snippet template in SniCaps is aligned to form the motion pattern. The weight α_{ij} controls the importance of j^{th} snippet template in the i^{th} snippet. In other words, a $(\boldsymbol{P}_i, \boldsymbol{\alpha}_i)$ describes how the N snippet templates are used to construct an l-long snippet and a SegCap requires S such parameters to describe a full L-long sequence. Fig. 2(c) illustrates the interpretation of $\mathcal{P}$. Both $\boldsymbol{B}$ and ν are dependent on the input. $\boldsymbol{B} \in \mathbb{R}^{(d+1) \times (d+1)}$ is a transformation on $\boldsymbol{P}$, and $\nu \in [0, 1]$ is the activation of the segment template.

Segment Encoding/Decoding Assume we have M SegCaps with which we hope to reconstruct the low-level motion encoded in the SniCap parameters. This is equivalent to reconstructing all the data-dependent SniCap parameters $[\mathcal{C}_1^{\texttt{Sni}}, \ldots, \mathcal{C}_S^{\texttt{Sni}}]$, where $\mathcal{C}_i^{\texttt{Sni}} = \{(\boldsymbol{A}_j, \mu_j) \mid j = 1, \ldots, N\}$ is the set of SniCap parameters for the i^{th} snippet. To obtain the SegCap parameters, we encode the S-long sequence of SniCap parameters with an LSTM model $f_{\texttt{LSTM}}$ shared by all SegCaps, and M

fully-connected layers (one for each SegCap) to produce $\{\boldsymbol{B}, \nu\}$. Formally,

$$\begin{aligned} \boldsymbol{h} &= f_{\texttt{LSTM}}\Big(\big[\mathcal{C}_1^{\texttt{Sni}}, \dots, \mathcal{C}_S^{\texttt{Sni}}\big]\Big), \\ \{\boldsymbol{B}^{(k)}, \nu^{(k)}\} &= f_{\texttt{FC}}^{(k)}(\boldsymbol{T}, \boldsymbol{h}), \quad k = 1, \dots, M, \end{aligned} \tag{3}$$

where superscript (k) refers to the k^{th} SegCap. The transformation and activation parameters are then applied to $\mathcal{P}$ to reconstruct snippet parameters

$$\begin{aligned} \hat{\boldsymbol{P}}_{ij}^{(k)} &= \boldsymbol{B}^{(k)} \times \boldsymbol{P}_{ij}^{(k)}, \quad i = 1, \dots, S, \quad j = 1, \dots, N, \quad k = 1, \dots, M, \\ \hat{\mathcal{C}}_i^{\texttt{Sni}} &= (\hat{\boldsymbol{A}}_i, \hat{\boldsymbol{\mu}}_i) = \Big(\sum_k^M \nu^{(k)} \hat{\boldsymbol{P}}_i^{(k)}, \sum_k^M \nu^{(k)} \boldsymbol{\alpha}_i^{(k)}\Big), \quad i = 1, \dots, S, \end{aligned} \tag{4}$$

where $\hat{\boldsymbol{A}}_i \in \mathbb{R}^{N \times (d+1) \times (d+1)}$ and $\hat{\boldsymbol{\mu}}_i \in \mathbb{R}^N$ are the reconstructed snippet transformation and activation of the snippet templates for the i^{th} snippet. Note that $S = L/l$, which means $f_{\texttt{LSTM}}$ can have a much smaller footprint than a recurrent network that handles the whole L-long sequence.

The above formulation enables SegCap to learn a transformation-invariant representation of motion. Intuitively, $\mathcal{P}$ describes snippet-segment relation, and $\boldsymbol{B}$ can be regarded as the spatial relation between a segment template pattern and the observed trajectory. The segment activation $\boldsymbol{\nu} \in \mathbb{R}^M$ reveals the semantics of the input trajectory and can be used for self-supervised training.

3.4 Training

As delineated in Section 3.2 and 3.3, SniCap and SegCap play different roles by capturing information at two different abstraction levels. SniCap focuses on short-time motion while SegCap is defined upon SniCap to model long-time semantic information. Hence, the two autoencoders are trained using different objective functions.

The only objective of the snippet autoencoder is to faithfully reconstruct the original input. Therefore, for a training sample $\boldsymbol{X} = \{\boldsymbol{x}_i | i = 1, \dots, L\}$, we use a self-supervised reconstruction loss:

$$\mathcal{L}_{\texttt{Rec}}^{\texttt{Sni}} = \sum_{i=1}^{L} ||(\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i)||_2^2, \tag{5}$$

where $\hat{\boldsymbol{x}}_i$ denotes the reconstructed coordinate following Equation (2).

The segment autoencoder's primary goal is to reconstruct the input SniCap parameters, hence the reconstruction loss

$$\mathcal{L}_{\texttt{Rec}}^{\texttt{Seg}} = \sum_{i=1}^{S} ||(\hat{\boldsymbol{A}}_i - \boldsymbol{A}_i)||_2^2 + ||(\hat{\boldsymbol{\mu}}_i - \boldsymbol{\mu}_i)||_2^2. \tag{6}$$

Furthermore, we use unsupervised contrastive training to learn semantic meaningful activations $\boldsymbol{\nu}$. For a batch of B samples, the contrastive loss is

$$\mathcal{L}_{\texttt{Con}}^{\texttt{Seg}} = -\frac{1}{B} \sum_{i=1}^{B} \log \frac{\exp\left(\texttt{cossim}(\boldsymbol{\nu}_i', \boldsymbol{\nu}_i'')/\tau\right)}{\sum_j \exp\left(\texttt{cossim}(\boldsymbol{\nu}_i', \boldsymbol{\nu}_j'')/\tau\right)}, \tag{7}$$

where $\tau = 0.1$ is the temperature used for all experiments, $\boldsymbol{\nu}_i'$ and $\boldsymbol{\nu}_i''$ is the segment activation of sample $\boldsymbol{X}_i'$ and $\boldsymbol{X}_i''$, respectively. Here, $\boldsymbol{X}_i'$ and $\boldsymbol{X}_i''$ are the spatial-temporally disturbed version of $\boldsymbol{X}_i$. The disturbance is dataset-dependent and will be discussed in supplementary materials.

In additional to the above loss terms, we impose two regularizers: a smoothness constraint on reconstructed sequence, and a sparsity regularization on the segment activations

$$\mathcal{L}_{\texttt{Smt}}^{\texttt{Reg}} = \sum_{i=2}^{L} ||\hat{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_{i-1}||_2^2, \quad \mathcal{L}_{\texttt{Sps}}^{\texttt{Reg}} = \frac{1}{B} \sum_{i=1}^{B} ||\boldsymbol{\nu}_i||_2^2. \tag{8}$$

The final training objective is:

$$\mathcal{L} = \lambda^{\texttt{Sni}} \mathcal{L}_{\texttt{Rec}}^{\texttt{Sni}} + \lambda^{\texttt{Seg}} \mathcal{L}_{\texttt{Rec}}^{\texttt{Seg}} + \mathcal{L}_{\texttt{Con}}^{\texttt{Seg}} + 0.5 \mathcal{L}_{\texttt{Smt}}^{\texttt{Reg}} + 0.05 \mathcal{L}_{\texttt{Sps}}^{\texttt{Reg}}, \tag{9}$$

where $\lambda^{\texttt{Sni}}$ and $\lambda^{\texttt{Seg}}$ are empirically determined.

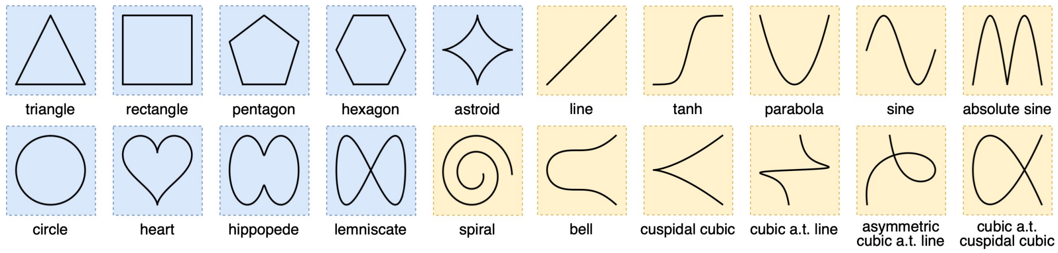


Figure 3: The 20 motion patterns in the Trajectory20 (T20) dataset. a.t. is short for "asymptotic to".

4 Experiments

In this section, we first assess the proposed MCAE on a synthetic motion dataset to show its ability in learning transformation-invariant robust representations. Then, we generalize MCAE to multi-point systems and show its efficacy in real-world skeleton-based human action datasets. We report the mean accuracy and standard error based on three runs with random initialization. The experiments are run on an NVIDIA Titan V GPU, where we use a batch size of 64, and the Adam [10] optimizer with a learning rate of 10^{-3}. Please refer to the supplementary material for details.

4.1 Learning from Synthesized Motion

The Trajectory20 Dataset While datasets like movingMNIST [34] have been commonly used in the motion representation learning literature, it is innately *linear* and has limited motion variations. Moreover, its prediction-oriented setting makes it difficult to examine the motion category of each trajectory. In this paper, we introduce the Trajectory20 (T20), a synthetic trajectory dataset based on 20 distinct motion templates (as shown in Fig. 3). Each sample in T20 is a 32-step-long sequence of coordinates in $[-1, 1]^2$. In the data generating process, a motion template is picked randomly and is randomly rotated, scaled, and translated to a random position to produce a trajectory. A closed trajectory (marked blue in Fig. 3) starts at a random point on the trajectory and end at the same point, whereas an open trajectory (marked yellow in Fig. 3) starts at a random end's vicinity. The randomized generating process ensures the trajectories are controllably diverse in scale, rotation, and position. The training data is generated on-the-fly and a fixed test set of 10,000 samples is used for evaluation. Examples of T20 are shown in the supplementary material.

Table 1: Ablation study on T20.

Reg.	l	#Sni	#Seg	Acc. (%)
Full	8	8	80	69.30 ± 0.76
	4	8	80	41.01 ± 8.81
	16	8	80	45.83 ± 8.36
	8	2	80	64.02 ± 2.10
	8	4	80	68.17 ± 0.36
	8	16	80	48.11 ± 1.60
	8	8	32	42.36 ± 3.15
	8	8	64	63.94 ± 1.41
	8	8	128	69.44 ± 1.69
w/o $\mathcal{L}^{\texttt{Reg}}_{\texttt{Smt}}$	8	8	80	67.60 ± 1.69
w/o $\mathcal{L}^{\texttt{Reg}}_{\texttt{Sps}}$	8	8	80	65.92 ± 1.63

Ablation Study We perform an ablation study of MCAE on T20 to examine the effect of different regularizers and three key hyperparameters: snippet length l, the numbers of SniCap (#Sni) and SegCap (#Seg). The result is shown in Table 1. The length of snippet l plays a vital role in learning a useful representation. A very small l results in a narrow receptive field for snippet capsules, which makes it less useful for inferring semantics of the whole sequence. At the other end, a large l makes snippets challenging to reconstruct. The numbers of SniCap and SegCap also have major effect on the outcome. Too few SniCaps makes it difficult to reconstruct the input motion signal. Too few SegCaps undermines the expressiveness of the segment autoencoder. Too many SniCaps could cause difficulty in learning proper alignments between SegCaps and SniCaps. Both degrade the quality of the learned features. Moreover, increasing #Seg from 80 to 128 does not bring further improvements. As the result shows, (l, #Sni, #Seg) = (8, 8, 80) performs well and we will use it in all experiments below. As for the regularizers, while both regularizers improve the performance, the sparsity regulation ($\mathcal{L}^{\texttt{Reg}}_{\texttt{Sps}}$) on segment activation is more helpful for learning discriminative features.

Motion Classification We compare MCAE with the following baseline models, namely KMeans, DTW-KMeans, k-Shape [25], LSTM and 1D-Conv[2]. KMeans, DTW-KMeans, and k-Shape are parameter-free time series clustering algorithms. Briefly, KMeans uses Euclidean distance to measure the similarity between signals. DTW-KMeans normalizes input signals using dynamic time warping [29], and performs KMeans on the normalized signals. k-Shape uses cross-correlation based distance measure to cluster time series. We use the implementation by `tslearn` [36] for the three clustering methods. LSTM, 1D-Conv, and MCAE are used as backbone networks, which take the raw coordinate sequence as input and output a feature vector of a pre-defined dimension. The feature vector is used for contrastive learning following Equation (7). Upon each model we attach an auxiliary linear classifier (i.e. a single layer perceptron), which is trained with labels but the gradient is blocked from back-propagating into the backbone. The corresponding accuracy reflects the quality of the learned representation.

For LSTM and 1D-Conv backbone, different numbers of hidden units/channels have been explored (shown as Hidden Param. in Table 2), which has resulted in different model sizes (measured by #Param. in Table 2).

As shown in Table 2, since the spatial variance (e.g. viewpoint changes) within motion signal cannot be directly captured by temporal warping/correlation, all the three parameter-free clustering methods perform poorly on T20. On the other hand, with considerably fewer parameters, MCAE outperforms LSTM and 1D-CNN by a large margin. This provides quantitative evidence that MCAE can capture the transformation-invariant semantic information more efficiently than the compared baselines.

Table 2: Unsupervised learning performance of MCAE and baselines on T20.

	Hidden Param.	#Param.	Acc. (%)
KMeans	–	–	8.57 ± 0.04
DTW-KMeans	–	–	9.12 ± 0.20
k-Shape [25]	–	–	12.94 ± 0.34
LSTM	128	600k	29.17 ± 2.45
	256	669k	40.03 ± 0.57
	512	805k	45.59 ± 1.37
	1,024	1,078k	53.47 ± 1.52
	2,048	1,625k	54.32 ± 0.55
1D-Conv	128	588k	44.78 ± 0.57
	256	787k	53.69 ± 0.53
	512	1,185k	57.57 ± 0.56
	1,024	1,982k	57.58 ± 0.08
	(#Sni, #Seg)	#Param.	Acc. (%)
MCAE	(8, 80)	**277k**	**69.30 ± 0.76**

4.2 Generalizing to Multiple Points

The MCAE running on T20 dataset handles a single moving point while most real-world problems involve multiple points. This section presents a naive (yet effective) extension of MCAE, which we name MCAE-MP, to enable processing the motion of multi-point systems. Such motion can be described as $\mathcal{X} = \{\boldsymbol{X}_i | i = 1, \ldots, K\}$, where K is the number of moving points. The extension works as follows:

1. The K moving points are processed separately by an MCAE. This results in K segment activation vectors $\{\boldsymbol{\nu}_i, | i = 1, \ldots, K\}$.
2. The K activation vectors are concatenated into a single representation $\boldsymbol{\nu} \in \mathbb{R}^{KM}$, which is used for unsupervised learning following Equation (9).

Skeleton-based Human Action Recognition We apply MCAE-MP to solve the unsupervised skeleton-based action recognition problem, where a human skeleton is a system consisting of multiple moving joints (points). Three widely-used datasets are used for evaluation: NW-UCLA [42], NTU-RGBD60 (NTU60) [31], and NTU-RGBD120 (NTU120) [19]. The three datasets consist of sequences with 1 or 2 subjects whose movement is measured in 3D space. For NW-UCLA, we follow previous works [35] to train the model on view 1 and 2, and test the model on view 3. For NTU60, we follow the official data split for the cross-subject (XSUB) and cross-view (XVIEW) protocols. The similar is implemented on NTU120 for the cross-subject (XSUB) and cross-setting (XSET) protocol. For ease of implementation, we project the 3D sequence into three orthonormal 2D spaces and use an MCAE defined on 2D space to process the three views of the sequences. Then the segment activations from the three views are concatenated to form the representation. Four types

[2] Architectures of LSTM and 1D-Conv are detailed in the supplementary material.

Table 3: Performance (%) for unsupervised skeleton-based action classification. Column "Mod." shows the data modality, where "S" indicates skeleton and "D" indicates depth map. Column "Cls." shows the auxiliary classifier used for supervised training.

			NTU60		NTU120		NW-UCLA
Model	Mod.	Cls.	XSUB	XVIEW	XSUB	XSET	V1&V2 $\rightarrow$ V3
Luo *et al.* [21]	S+D	SLP	61.4	53.2	–	–	50.7
Li *et al.* [14]	S+D	SLP	68.1	63.9	–	–	62.5
SeBiReNet [23]	S	LSTM	–	79.7	–	–	80.3
LongT GAN [54]	S	SLP	39.1	48.1	–	–	74.3
MS^{2}L [18]	S	SLP	52.6	–	–	–	76.8
CAE+ [27]	S	SLP	58.5	64.8	48.6	49.2	–
MCAE-MP (SLP)	S	SLP	**65.6**	74.7	**52.8**	**54.7**	83.6
P&C [35]	S	1-NN	50.7	76.1	–	–	**84.9**
MCAE-MP (1-NN)	S	1-NN	51.9	**82.4**	42.3	46.1	79.1

of disturbance are introduced for contrastive learning, namely jittering, spatial rotation, masking, and temporal smoothing. The readers are referred to the supplementary material for details.

The classification accuracy is put into three groups in Table 3. In the first group are the prior works that are not directly comparable as they use depth map [14, 21] or stronger auxiliary classifiers for supervised training [23]. In the second group, where our model is marked as MCAE-MP (SLP), a single layer perceptron (SLP) is trained as the auxiliary classifier with backbone parameters frozen. In the third group, where our model is marked as MCAE-MP (1NN), a 1-nearest-neighbor classifier is used instead of an SLP. Although MCAE-MP is a naive extension as it encodes joints separately and largely ignores their interactions, it achieves better or competitive performance compared with the baselines. Notably, on NTU60-XVIEW and NTU120-XSET where the training set and test set have different viewpoints, our model outperforms baselines by a clear margin thanks to the capsule-based representation which effectively captures viewpoint changes as transformations on input.

4.3 What does MCAE Learn?

To better understand what is encoded, we plot the learned snippet templates $\mathcal{T}$ and segment templates $\mathcal{P}$ in Fig. 4. Note that $\mathcal{T}$ are initialized as random straight lines, and $\mathcal{P}$ are initialized as arbitrary patterns composed randomly of $\mathcal{T}$. As shown in Fig. 4a, the snippets are mainly simple lines and hook-like curves that does not carry semantic information. Segment templates in Fig. 4b, however, bear some resemblance to the patterns shown in Fig. 3. This suggests that semantic-agnostic snippets are being aggregated into semantic-aware segments.

(a) Snippet templates $\mathcal{T}$.

(b) Samples of segment templates $\mathcal{P}$.

Figure 4: Templates learned from Trajectory20 dataset. Color indicates time.

We proceed to explore the information in SegCaps. In particular, we would like to see if SegCaps have learned transformation-invariant information. To this purpose, we randomly sample a trajectory from T20 dataset. The trajectory is first normalized so that its centroid is at $(0, 0)$, then rotated clockwise by an angle θ, and finally fed into the model. We examine the segment templates with the highest activation values (which reflects the trajectory's semantics) and calculate the rotation angle ϕ from those templates' parameter $\boldsymbol{B}$. As shown in Table 4, the calculated ϕ reveals two types of segments templates as we rotate the input. One type yields constant ϕ (e.g. segment ID 2 for sample "absolute sine"), which indicates its rotation-invariance, the other has ϕ that changes monotonically with θ (e.g. segment ID 8 for sample "hexagon"), which shows its rotation-awareness. As for the activation values, samples from different categories activates different set of segments.

Table 4: Top-5 segment templates (sorted by segment activation ν then segment ID for better visualization), and the rotation ϕ calculated from their parameters $\boldsymbol{B}$. Bold IDs are segments repeating across different θ.

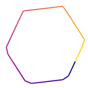
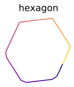
	$\theta=-10°$		$\theta=-5°$		$\theta=0°$		$\theta=5°$		$\theta=10°$	
Input	ID	ϕ	ID	ϕ	ID	ϕ	ID	ϕ	ID	ϕ
hexagon 	**2**	6.3	**2**	6.7	**2**	6.8	**2**	7.0	**2**	7.1
	8	6.9	**8**	9.0	**8**	11.2	**8**	13.9	**8**	16.5
	12	54.9	**12**	55.5	**12**	55.8	**12**	56.5	**12**	56.8
	37	-20.8	**37**	-19.8	**37**	-18.9	**37**	-17.9	**37**	-16.9
	66	50.2	**66**	52.5	**66**	55.4	**66**	59.0	**66**	62.4
abs_sine 	**2**	12.1	**2**	12.3	**2**	12.2	**2**	12.1	**2**	11.9
	7	8.2	5	-10.7	5	-10.1	5	-9.9	**7**	17.2
	33	65.1	**7**	10.7	**7**	13.4	**7**	15.4	32	-9.7
	37	-22.9	**37**	-22.3	**37**	-21.8	**37**	-21.3	**37**	-19.9
	46	45.7	**46**	47.5	**46**	48.6	**46**	50.2	**46**	51.6

Table 5: Top-5 segment templates (sorted by segment activation ν then segment ID for better visualization), and the translation (x, y) calculated from their parameters $\boldsymbol{B}$.

	$(\Delta x, \Delta y)$ = (-0.2, 0)			$(\Delta x, \Delta y)$ = (-0.1, 0)			$(\Delta x, \Delta y)$ = (0, 0)			$(\Delta x, \Delta y)$ = (0, 0.1)			$(\Delta x, \Delta y)$ = (0, 0.2)		
Input	ID	x	y	ID	x	y	ID	x	y	ID	x	y	ID	x	y
hexagon 	**2**	0.05	0.18	**2**	0.17	0.19	**2**	0.27	0.19	**2**	0.28	0.28	**2**	0.27	0.37
	8	0.01	-0.07	**8**	0.09	-0.06	**8**	0.18	-0.04	**8**	0.19	0.04	**8**	0.19	0.12
	12	-0.09	0.13	**12**	0.00	0.13	**12**	0.09	0.13	**12**	0.09	0.23	**12**	0.09	0.32
	37	0.10	-0.11	**37**	0.18	-0.11	**37**	0.27	-0.11	**37**	0.27	-0.03	**37**	0.27	0.05
	66	-0.12	0.16	**66**	-0.03	0.16	**66**	0.05	0.17	**66**	0.06	0.26	**66**	0.06	0.35
abs_sine 	**2**	0.04	0.2	**2**	0.14	0.19	**2**	0.24	0.19	**2**	0.24	0.28	**2**	0.23	0.38
	5	-0.01	0.30	**5**	0.07	0.29	**5**	0.16	0.29	**5**	0.16	0.38	**5**	0.15	0.46
	7	0.20	-0.16	**7**	0.28	-0.16	**7**	0.37	-0.15	**7**	0.36	-0.06	**7**	0.36	0.04
	37	0.04	-0.17	**37**	0.12	-0.16	**37**	0.21	-0.16	**37**	0.20	-0.07	**37**	0.20	0.01
	46	0.02	0.01	**46**	0.13	0.02	**46**	0.23	0.04	**46**	0.23	0.13	**46**	0.22	0.23

Meanwhile, the same sample under different rotation angle θ gives stable segment activation, despite some changes which are found to have no effect on the classification result.

We do a similar study on the translation component (x, y), where we translate the input by $(\Delta x, \Delta y)$. As shown in Table 5, (x, y) changes monotonically with $(\Delta x, \Delta y)$ while the activated segments remain stable. These results suggest that the semantics and transformation information has been encoded separately in the segment activation $\boldsymbol{\nu}$ and transformation parameters $\boldsymbol{B}$. In other words, the encoded semantic information is robust against geometric transformations.

5 Conclusion

In this paper, we introduce MCAE, a framework that learns robust and discriminative representation for keypoint motion. To resolve the intra-class variation of motion, we propose to learn a compact and transformation-invariant motion representation using a two-level capsule-based representation hierarchy. The efficacy of the learned representation is shown through an experimental study on synthetic and real-world datasets. The output of MCAE could serve as mid-level representation in other frameworks, e.g. Graph Convolution Network, for tasks that involve more context than classification. We anticipate this work to inspire further studies that apply capsule-based models to other time series processing tasks, such as joint modeling of visual appearance and motion in video. The software and the T20 dataset of our research will be released to the research community.

Motion analysis techniques are in the foreground of the misuse of machine learning methods, among which adverse societal impacts and privacy breach are two major concerns. Regarding the societal impacts, admittedly, our method has both upside and downside. On one hand, a transformation-invariant motion representation enables us better decode the information implicit in the trajectory, which has applications for example in ethology. On the other hand, it could also be misused in mass surveillance. Appropriate boundaries of use and ethical review are required to prevent potential malicious applications. Regarding the privacy concerns, our method isolates the subjects' motion from their sensitive information, such as gender and race.

References

[1] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691, 2017.

[2] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *CVPR*, pages 1110–1118, 2015.

[3] Kevin Duarte, Yogesh Rawat, and Mubarak Shah. VideoCapsuleNet: A simplified network for action detection. In *NeurIPS*, pages 7610–7619, 2018.

[4] Tian Gan, Yongkang Wong, Daqing Zhang, and Mohan Kankanhalli. Temporal encoded f-formation system for social interaction detection. In *ACM Multimedia*, pages 937–946, 2013.

[5] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social GAN: Socially acceptable trajectories with generative adversarial networks. In *CVPR*, pages 2255–2264, 2018.

[6] Ahmed Refaat Hawas, Heba A. El-Khobby, Mohammed Abd-Elnaby, and Fathi E. Abd El-Samie. Gait identification by convolutional neural networks and optical flow. *Multimedia Tools and Applications*, 78(18):25873–25888, 2019.

[7] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. In *ICANN*, volume 6791 of *Lecture Notes in Computer Science*, pages 44–51. Springer, 2011.

[8] Mariano Jaimez, Mohamed Souiai, Javier Gonzalez-Jimenez, and Daniel Cremers. A primal-dual framework for real-time dense RGB-D scene flow. In *ICRA*, pages 98–104, 2015.

[9] Yanli Ji, Feixiang Xu, Yang Yang, Ning Xie, Heng Tao Shen, and Tatsuya Harada. Attention transfer (ANT) network for view-invariant action recognition. In *ACM Multimedia*, pages 574–582, 2019.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[11] Adam Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E Hinton. Stacked capsule autoencoders. In *NeurIPS*, pages 15512–15522, 2019.

[12] Mohamed Ilyes Lakhal, Oswald Lanz, and Andrea Cavallaro. View-lstm: Novel-view video synthesis through view decomposition. In *ICCV*, pages 7576–7586, 2019.

[13] Binlong Li, Octavia I. Camps, and Mario Sznaier. Cross-view activity recognition using hankelets. In *CVPR*, pages 1362–1369, 2012.

[14] Junnan Li, Yongkang Wong, Qi Zhao, and Mohan S. Kankanhalli. Unsupervised learning of view-invariant action representations. In *NeurIPS*, pages 1262–1272, 2018.

[15] Maosen Li, Siheng Chen, Xu Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. Actional-structural graph convolutional networks for skeleton-based action recognition. In *CVPR*, pages 3595–3603, 2019.

[16] Wanqing Li, Zhengyou Zhang, and Zicheng Liu. Action recognition based on a bag of 3D points. In *CVPR Workshops*, pages 9–14, 2010.

[17] Yan Li, Bin Ji, Xintian Shi, Jianguo Zhang, Bin Kang, and Limin Wang. TEA: Temporal excitation and aggregation for action recognition. In *CVPR*, pages 906–915, 2020.

[18] Lilang Lin, Sijie Song, Wenhan Yang, and Jiaying Liu. MS2L: multi-task self-supervised learning for skeleton based action recognition. In *ACM Multimedia*, pages 2490–2498, 2020.

[19] Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C. Kot. NTU RGB+D 120: A large-scale benchmark for 3D human activity understanding. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 42(10):2684–2701, 2020.

[20] Ziyu Liu, Hongwen Zhang, Zhenghao Chen, Zhiyong Wang, and Wanli Ouyang. Disentangling and unifying graph convolutions for skeleton-based action recognition. In *CVPR*, pages 143–152, 2020.

[21] Zelun Luo, Boya Peng, De-An Huang, Alexandre Alahi, and Li Fei-Fei. Unsupervised learning of long-term motion dynamics for videos. In *CVPR*, pages 7101–7110. IEEE Computer Society, 2017.

[22] Joe Yue-Hei Ng, Matthew J. Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, pages 4694–4702, 2015.

[23] Qiang Nie, Ziwei Liu, and Yunhui Liu. Unsupervised 3d human pose representation with viewpoint and pose disentanglement. In *ECCV*, volume 12364 of *Lecture Notes in Computer Science*, pages 102–118. Springer, 2020.

[24] Dan Oneata, Jakob Verbeek, and Cordelia Schmid. Action and event recognition with fisher vectors on a compact feature set. In *ICCV*, pages 1817–1824, 2013.

[25] John Paparrizos and Luis Gravano. k-shape: Efficient and accurate clustering of time series. In *SIGMOD*, pages 1855–1870, 2015.

[26] Vasu Parameswaran and Rama Chellappa. View invariance for human action recognition. *International Journal of Computer Vision*, 66(1):83–101, 2006.

[27] Haocong Rao, Shihao Xu, Xiping Hu, Jun Cheng, and Bin Hu. Augmented skeleton based contrastive action learning with momentum LSTM for unsupervised action recognition. *Information Sciences*, 569:90–109, August 2021.

[28] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *NIPS*, pages 3856–3866, 2017.

[29] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

[30] Arun Sankisa, Arjun Punjabi, and Aggelos K. Katsaggelos. Temporal capsule networks for video motion estimation and error concealment. *Signal Image Video Process.*, 14(7):1369–1377, 2020.

[31] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. NTU RGB+D: A large scale dataset for 3D human activity analysis. In *CVPR*, pages 1010–1019, 2016.

[32] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NIPS*, pages 802–810, 2015.

[33] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, pages 568–576, 2014.

[34] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using LSTMs. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 843–852, 2015.

[35] Kun Su, Xiulong Liu, and Eli Shlizerman. PREDICT & CLUSTER: unsupervised skeleton based action recognition. In *CVPR*, pages 9628–9637, 2020.

[36] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, and Eli Woods. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1–6, 2020.

[37] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3D convolutional networks. In *ICCV*, pages 4489–4497, 2015.

[38] Raviteja Vemulapalli, Felipe Arrate, and Rama Chellappa. Human action recognition by representing 3D skeletons as points in a Lie group. In *CVPR*, pages 588–595, 2014.

[39] Shruti Vyas, Yogesh Singh Rawat, and Mubarak Shah. Multi-view action recognition using cross-view video prediction. In *ECCV*, volume 12372 of *Lecture Notes in Computer Science*, pages 427–444. Springer, 2020.

[40] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *ICCV*, pages 3551–3558, 2013.

[41] Jiang Wang, Zicheng Liu, Ying Wu, and Junsong Yuan. Mining actionlet ensemble for action recognition with depth cameras. In *CVPR*, pages 1290–1297, 2012.

[42] Jiang Wang, Xiaohan Nie, Yin Xia, Ying Wu, and Song-Chun Zhu. Cross-view action modeling, learning, and recognition. In *CVPR*, pages 2649–2656, 2014.

[43] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, pages 7794–7803, 2018.

[44] Yunbo Wang, Lu Jiang, Ming-Hsuan Yang, Li-Jia Li, Mingsheng Long, and Li Fei-Fei. Eidetic 3D LSTM: A model for video prediction and beyond. In *ICLR*, 2019.

[45] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*, 2018.

[46] Ceyuan Yang, Yinghao Xu, Jianping Shi, Bo Dai, and Bolei Zhou. Temporal pyramid network for action recognition. In *CVPR*, pages 588–597, 2020.

[47] Yue Yu, Niehao Tian, Xiangru Chen, and Ying Li. Skeleton capsule net: An efficient network for action recognition. In *ICVRV*, pages 74–77, 2018.

[48] Pengfei Zhang, Cuiling Lan, Wenjun Zeng, Junliang Xing, Jianru Xue, and Nanning Zheng. Semantics-guided neural networks for efficient skeleton-based human action recognition. In *CVPR*, pages 1112–1121, 2020.

[49] Xikun Zhang, Chang Xu, and Dacheng Tao. Context aware graph convolution for skeleton-based action recognition. In *CVPR*, pages 14333–14342, 2020.

[50] Yiyi Zhang, Li Niu, Ziqi Pan, Meichao Luo, Jianfu Zhang, Dawei Cheng, and Liqing Zhang. Exploiting motion information from unlabeled videos for static image action recognition. In *AAAI*, pages 12918–12925, 2020.

[51] Yujia Zhang, Xiaodan Liang, Dingwen Zhang, Min Tan, and Eric P Xing. Unsupervised object-level video summarization with online motion auto-encoder. *Pattern Recognition Letters*, 130:376–385, 2020.

[52] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3D point capsule networks. In *CVPR*, pages 1009–1018, 2019.

[53] Yongheng Zhao, Tolga Birdal, Jan Eric Lenssen, Emanuele Menegatti, Leonidas J. Guibas, and Federico Tombari. Quaternion equivariant capsule networks for 3D point clouds. In *ECCV*, volume 12346 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2020.

[54] Nenggan Zheng, Jun Wen, Risheng Liu, Liangqu Long, Jianhua Dai, and Zhefeng Gong. Unsupervised representation learning with long-term dynamics for skeleton based action recognition. In *AAAI*, pages 2644–2651, 2018.

[55] Tao Zhuo, Zhiyong Cheng, Peng Zhang, Yongkang Wong, and Mohan Kankanhalli. Explainable video action reasoning via prior knowledge and state transitions. In *ACM Multimedia*, pages 521–529, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Please see Section 1.
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Please check Section 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The source code and dataset will be released with the publication of the paper. Implementation details are covered in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Training details are mostly provided in Section 4 while some extra information is in the supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We provided error bar on our T20 dataset. For experiments on NW-UCLA, NTURGBD-60, and NTURGBD-120, we followed the protocol in prior works.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please see the first paragraph of Section 4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We used the publicly available NTURGBD60/120 and NW-UCLA dataset. The original papers have been cited.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] The T20 dataset will be released with the publication of the paper.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] The data is publicly available and the consent has been described in the published paper.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No such information was found in the dataset we use.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No crowdsourcing research was conducted.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]