(Supplementary) Data-Free Class-Incremental Hand Gesture Recognition

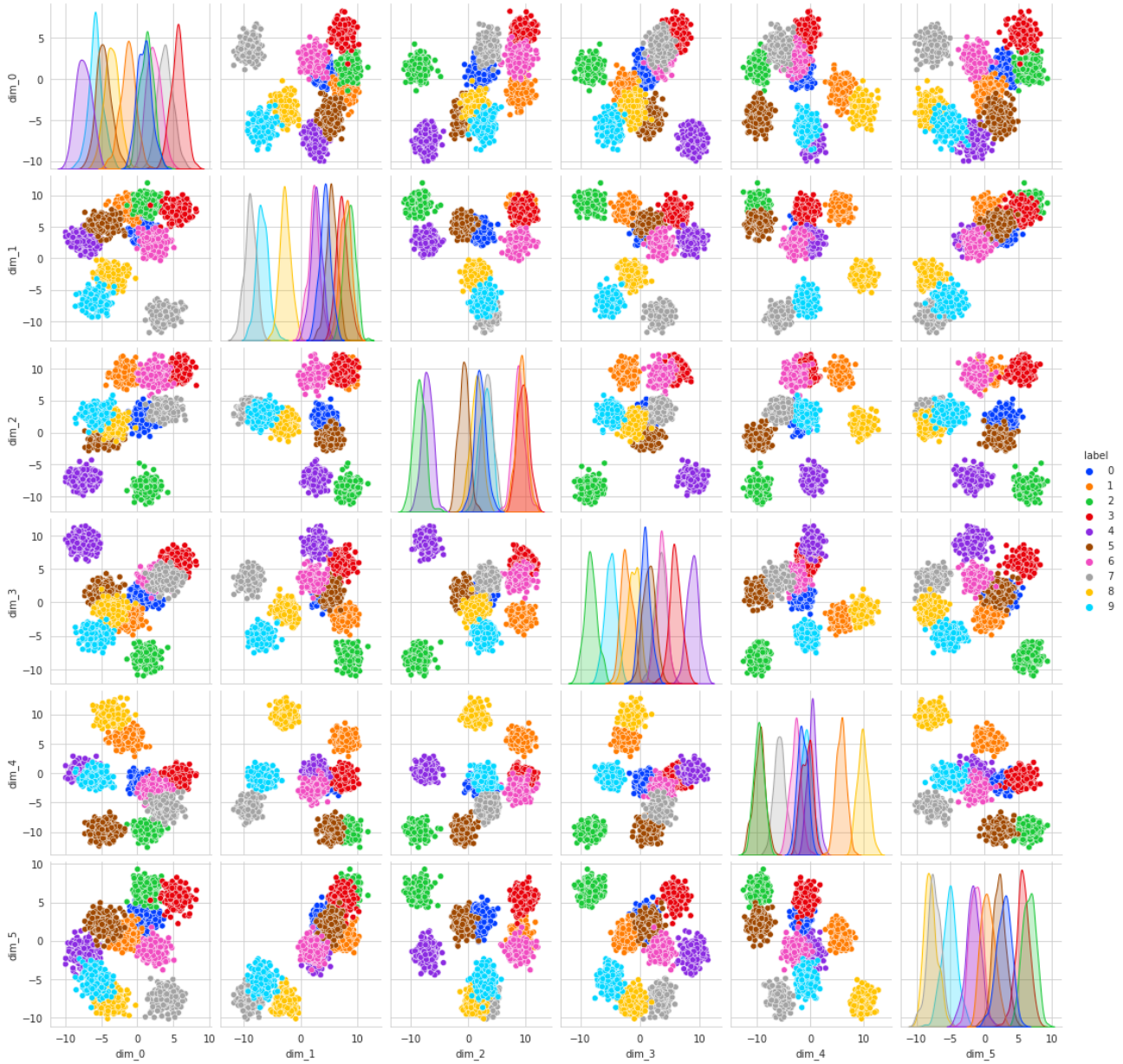

Figure 1: The pair plot of the synthetic dataset (6D Gaussian blobs) used for the preliminary proof of concept in Section 3 (BOAT-MI: Our Methodology) of the main paper. Different colors indicate different class indices (labels on the right). For the DFCIL setup, the first 5 classes (label $0 \rightarrow 4$) are considered the old set $\mathcal{C}_{\mathcal{O}}$, and the other 5 (label $5 \rightarrow 9$) new $\mathcal{C}_{\mathcal{N}}$.

Algorithm: Boundary-Aware Prototypical Model Inversion (Part 1)

Function `Main` ($\mathcal{F}$, $\mathcal{H}$, c, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, m, α_f, α, *it*, ε, n, δ) **:**
- # $\mathcal{F}$; $\mathcal{H}$: feature extractor; SVM classifier
- # c: index of class to invert
- # $\boldsymbol{\mu}$; $\boldsymbol{\Sigma}$: prototypical mean, covariance for class c
- # m; α_f, α: momentum; forward/reverse LR
- # it; ε: max iterations; tolerance threshold
- # n, δ: max # of samples per class; margin
- # Use inverted mean for initialization later
- $\mathbf{x}_\mu \leftarrow$ `Invert` ($\mathcal{F}$, $\boldsymbol{\mu}$, m, α, *iter*, ε, `NULL`)
- $\mathbf{ys} \leftarrow [\,]$ # initialize feature list for inversion
- # Extend the list with support vectors for class c
- $\mathbf{ys}$ += $\mathcal{H}$.`support_vectors`[c]
- # Get principal directions of prototypical
- # covariance and their unique linear combinations
- # as prototypical features for inversion
- $\mathbf{ps} \leftarrow$ `GetProtoDirections`($\boldsymbol{\Sigma}, n$)
- $\mathbf{ys}$ += `GetProtoFeatures`($\mathbf{ps}, \boldsymbol{\mu}, c, \alpha_f, \delta$)
- $n \leftarrow$ `len`($\mathbf{ys}$) # Final # of samples to invert
- $\mathbf{xs} \leftarrow [\mathbf{x}_\mu] * n$ # initialize input list for inversion
- **for** $i \leftarrow 1 : n$ **do**
  - $\mathbf{xs}[\mathbf{i}] \leftarrow$ `Invert` ($\mathcal{F}$, $\mathbf{ys}[\mathbf{i}]$, m, α, *iter*, ε, $\mathbf{x}_\mu$)
- **end**
- **return** $\mathbf{xs}$

End Function

Function `GetProtoDirections` ($\boldsymbol{\Sigma}$, n) **:**
- # $\boldsymbol{\Sigma}$: class prototypical covariance
- # n: maximum # of samples allowed
- # Get principal directions retaining 95% variance
- $\mathbf{ps} \leftarrow$ `PCA`($\boldsymbol{\Sigma}, 0.95$)
- $\mathbf{ps}$ += $(-\mathbf{ps})$ # extend with negative directions
- **if** *ClampNumSamples*($\mathbf{ps}, n$) **then**
  - **return** $\mathbf{ps}$
- **end**
- $n \leftarrow n -$ `len`($\mathbf{ps}$) # update max # of samples
- $ord = 2$ # start from adding 2nd order interactions
- **while** *TRUE* **do**
  - # add unique linear interactions of order ord
  - # $p_{new} = \mathbf{ps}[i] + \mathbf{ps}[j]$ for $ord = 2$
  - # $p_{new} = \mathbf{ps}[i] + \mathbf{ps}[j] + \mathbf{ps}[k]$ for $ord = 3$
  - $\mathbf{ps\_l} \leftarrow$ `LinearInteractions`($\mathbf{ps}, ord$)
  - # normalize to get the unit vectors
  - $\mathbf{ps\_l} \leftarrow$ `Normalize`($\mathbf{ps\_l}$)
  - **if** *ClampNumSamples*($\mathbf{ps\_l}, n$) **then**
    - $\mathbf{ps}$ += $\mathbf{ps\_l}$ # list extension
    - **return** $\mathbf{ps}$
  - **end**
  - $\mathbf{ps}$.`append`($\mathbf{ps\_l}$)
  - $n \leftarrow n -$ `len`($\mathbf{ps\_l}$) # update max # of samples
  - $ord \leftarrow ord + 1$ # increment order
- **end**

End Function

Algorithm: Boundary-Aware Prototypical Model Inversion (Part 2)

Function `Invert` ($\mathcal{F}$, $\mathbf{y}$, m, α, *iter*, ε, $\mathbf{x_0}$) **:**
- # $\mathcal{F}$; $\mathbf{y}$: feature extractor; target feature
- # m; α: momentum; learning rate
- # iter; ε: max # of iterations; tolerance threshold
- # $\mathbf{x_0}$: input initialization, can be `NULL`
- **if** $\mathbf{x_0} ==$ *NULL* **then**
  - $\mathbf{x} \leftarrow$ input prediction tensor (zero-initialized)
- **else**
  - $\mathbf{x} \leftarrow \mathbf{x_0}$
- **end**
- $\mathbf{v} \leftarrow$ gradient update tensor (zero-initialized)
- **for** $i \leftarrow 1$: *iter* **do**
  - $\hat{\mathbf{y}} \leftarrow \mathcal{F}(\mathbf{x})$ # predict feature
  - $\mathcal{L} \leftarrow ||\mathbf{y} - \hat{\mathbf{y}}||_2 / ||\mathbf{y}||_2$ # distance norm
  - **if** $\mathcal{L} < \varepsilon$ **then**
    - **break**
  - **end**
  - $\mathbf{v} \leftarrow m\,\mathbf{v} + \nabla_{\mathbf{x}}\mathcal{L}$ # gradient tensor update
  - $\mathbf{x} \leftarrow \mathbf{x} - \alpha\,\mathbf{v}$ # gradient descent
- **end**
- **return** $\mathbf{x}$

End Function

Function `GetProtoFeatures` ($\mathbf{ps}, \boldsymbol{\mu}, c, \alpha_f, \delta$) **:**
- # $\mathbf{ps}$: list of principal directions and combinations
- # c, $\boldsymbol{\mu}$: class index and class prototypical mean
- # α_f, δ: forward learning rate and margin
- $\mathbf{ys} \leftarrow [\,]$ # initialize feature list for inversion
- **for** $i \leftarrow 1 :$ *len*($\mathbf{ps}$) **do**
  - $\mathbf{f} \leftarrow \boldsymbol{\mu}$
  - **for** $k \leftarrow 1$: *iter* **do**
    - $\mathbf{f} \leftarrow \mathbf{f} + \alpha_f\ \mathbf{ps}[\mathbf{i}]$
    - **if** *Classify*($\mathbf{f}$) $\neq c$ **then**
      - $\mathbf{f} \leftarrow (1 - \delta)\,\mathbf{f} + \delta\,\boldsymbol{\mu}$
      - `ASSERT Classify` ($\mathbf{f}$) $== c$
    - **end**
  - **end**
  - $\mathbf{ys}$.`append`($\mathbf{f}$)
- **end**
- **return** $\mathbf{ys}$

End Function

Function `ClampNumSamples` ($\mathbf{ps}$, n) **:**
- # $\mathbf{ps}$: list of samples to clamp
- # n: maximum # of samples allowed
- **if** $n >$ *len*($\mathbf{ps}$) **then**
  - **return** `FALSE` # not clamped
- **end**
- # randomly select n samples in-place
- $\mathbf{ps} \leftarrow$ `RandomSelect`($\mathbf{ps}$)
- **return** `TRUE` # clamped

End Function

Table 1: Results with BatchNorm (average of 3 runs with different class order) for DFCIL over six tasks in SHREC-2017.

Method	Task 0	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6		Mean (Task 1 → 6)	
Oracle (DGSTA-BN)							90.3								
		G↑	IFM ↓	G↑	IFM↓	G↑	IFM ↓	G↑	IFM ↓	G↑	IFM ↓	G↑	IFM↓	G↑	IFM↓
DeepInversion-BN [6]	92.8	81.4	7.0	62.6	18.2	52.9	29.1	46.3	35.6	34.8	48.1	31.5	51.3	51.6	31.6
ABD-BN [4]		81.7	1.9	72.2	2.0	65.9	10.7	57.3	14.8	51.7	16.3	42.7	25.2	61.9	11.8
R-DFCIL-BN [3]		72.0	5.1	61.5	7.3	53.2	9.6	50.7	5.1	46.6	8.9	39.9	0.6	54.0	6.1
BOAT-MI-BN (Ours)		86.5	7.5	82.2	9.3	74.7	1.2	72.6	2.6	65.5	8.7	62.3	1.7	**74.0**	**5.2**

Table 2: Results with BatchNorm (average of 3 runs with different class order) for DFCIL over six task in EgoGesture3D.

Method	Task 0	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6		Mean (Task 1 → 6)	
Oracle (DGSTA-BN)							75.0								
		G↑	IFM ↓	G↑	IFM↓	G↑	IFM ↓	G↑	IFM ↓	G↑	IFM ↓	G↑	IFM↓	G↑	IFM↓
DeepInversion-BN [6]	77.4	66.4	16.1	52.3	24.6	35.0	46.5	24.5	59.7	20.5	65.0	15.1	72.8	35.6	47.5
ABD-BN [4]		65.7	17.4	58.1	19.4	49.5	31.1	45.1	35.2	42.7	37.2	39.5	40.4	50.1	30.1
R-DFCIL-BN [3]		68.3	5.9	56.0	11.9	49.0	17.9	44.4	20.1	39.9	29.4	36.8	30.6	49.1	19.3
BOAT-MI-BN (Ours)		73.0	4.6	66.9	7.1	60.8	15.9	57.9	16.3	52.1	22.2	49.0	27.6	**60.0**	**15.6**

1. Experimental Details

We follow the original implementations of all DFCIL methods that we used for benchmarking in this work. However, these methods were originally designed for the image classification problem, as we discussed in Section 4.4 of the main paper. Therefore, we had to adjust them to our keypoint-based gesture recognition domain and tune their hyperparameters individually for our setup. All hyperparameter values and configuration files will be made available along with the codebase.

Base model training: We use the original DG-STA [1] architecture as our backbone to process the 3D keypoint sequences. We train the Oracle (upper bound, all classes in one task) and Task 0 models (base classes for each setup) with Adam optimizer (learning rate of 1e-3, $\beta_1 = 0.9$ and $\beta_2 = 0.999$) for 150 epochs.

Learning rate for incremental tasks: All the methods start from the same Task 0 weights for Task 1. We modify the initial learning rate for the incremental tasks depending on the method since a stronger regularization requires a higher learning rate to learn the new classes. In this regard, we select the optimal learning rate for each method with a grid search in the range $\{$1e-5 : 1e-3$\}$.

Synthetic data generation: Following ABD [4], for a fair comparison among the model inversion-based approaches (DeepInversion [6], ABD [4] and R-DFCIL [3]), we use the same model inversion strategy for the synthetic data generation. However, we optimize the randomly initialized inputs directly instead of training a model to generate the samples. After a grid search we find $\{\alpha_{lr}, \alpha_{con}, \alpha_{stat}, \alpha_{temp}\}$ as $\{$1e-1, 1, 2e2, 1e1$\}$.

R-DFCIL hyperparameters: R-DFCIL [3] includes three hyperparameters to weight each of the loss terms, the **local classification loss** (λ_{lce}), the **hard knowledge distillation loss** (λ_{hkd}) and the **relational knowledge distillation loss** (λ_{rkd}). For our experiments, we found $\{\lambda_{lce}, \lambda_{hkd}, \lambda_{rkd}\}$ as $\{$1, 1.5e-1, 7.5e-1$\}$ as the optimal values for SHREC-2017 and $\{$1, 1e-1, 1e-1$\}$ for EgoGesture3D.

BOAT-MI implementation: The complete `Python`-style pseudocode is provided in Algorithm 1 and 2. In Algorithm 1 Line 1, `Function Main` is the driver for our proposed model inversion mechanism for a single class c. Similar to the other baselines, we employ DG-STA [1] backbone as our feature extractor $\mathcal{F}$ here. We use the radial basis function as the kernel for our SVM classifier ($\mathcal{H}$) here. To generate proto-SVs for model inversion, it uses the principal directions of the prototypical variance $\mathbf{\Sigma}$ (Line 29, Algorithm 1). Ray casting from the class-prototypical mean $\boldsymbol{\mu}$ is done iteratively with a learning rate α_f (Line 29-36, Algorithm 2) until the ray hits the boundary. Also, a margin δ, normalized with respect to the distance between the mean $\boldsymbol{\mu}$ and the boundary vector, is used to avoid noise inherent in the boundary estimation process (Figure 2 main paper, Line 33, Algorithm 2). In other words, the proto-SV feature is taken to be the on the margin, which is δ inside the boundary.

Note that the number of principal directions (PDs) is pretty low compared to the dimensionality of the feature vector. Along with these raw PDs, we consider upto third-order interactions, i.e. simple unweighted linear combination of PDs ($p[i]+p[j]$ and $p[i]+p[j]+p[k]$, Line 37-40, Algorithm 1, $p[\cdot]$ is the list of PDs) ignoring duplicates, which we found to be empirically sufficient for all our studies. Thus, our augmentations are deterministic (not random), conditioned on the set of PDs.

The feature inversion procedure (feature → input, Line 1-22, Algorithm 2) deviates from the vanilla implementation [2] in three aspects. First, we initialize the input tensor with

the inverted class-prototypical mean (Line 9, Algorithm 2). Second, we employ the normalized $\mathcal{L}_2$ function as the distance metric (Line 14, Algorithm 2). Third, we replace the vanilla gradient descent with its momentum-based counterpart [5] (Line 18-19, Algorithm 2). All these modifications are empirically found to expedite the convergence.

The forward learning rate (α_f), momentum (m), and reverse learning rate (α) are set to 0.05, 0.9, and 1.0 respectively. We use the value of 0.2 as our normalized margin (δ) based upon the ablation study presented in Table 4 of the main paper. The rest of the parameters are set similarly to the baselines. All these configurations will be open-sourced with the codebase.

2. Additional Experiments

The effect of stat alignment loss during model inversion: Model inversion-based methods like DeepInversion, ABD and R-DFCIL use the BatchNorm (BN) statistics to compute a regularization loss that aligns the synthetic and real data distributions during model inversion. This loss is based on the KL divergence between the synthetic data distribution and the BatchNorm distribution of the previous task model, which reflects the real data statistics from the previous training. However, our backbone architecture, DG-STA [1], does not have BatchNorm layers and thus this loss term is not included. To investigate how the **stat alignment loss** affects model inversion for these baselines, we replace the LayerNorm layers with BatchNorm layers in DG-STA and run additional experiments.

Table 1 and 2 show the results for SHREC-2017 and EgoGesture3D setups, respectively. As can be seen, the incorporation of the BN layer does not change the overall ranking of the methods. With BN, our proposed approach achieves significantly higher global accuracy in each stage there with 12.1% and 9.9% improvements on average for SHREC-2017 (Table 1) and EgoGesture3D (Table 2), respectively, over the next best methods. Such improvements are also accompanied by the lower mean instantaneous forgetting in general – 0.9% and 3.7% lower IFM than the second best method for SHREC-2017 (Table 1) and EgoGesture3D (Table 2), respectively. Therefore, regardless of the choice of normalization layers, BOAT-MI excels the SOTA methods by a large margin.

Performance degradation/comparison to when all data is available, with large number of incremental stages: We first clarify that the upper bound of performance when all data is available – 75.8% – is reported on the "Oracle" row of Table 3 in our paper. To study this further with large number of incremental stages, we experimented with an extreme incremental learning setting on EgoGesture3D, where we added one class at a time to a model trained on the 59 base classes. Concretely, instead of adding 4 classes at a time over 6 incremental stages (as in paper), we added 1 new class at a time over 24 incremental stages. These results are shown in Figure 2 for both global accuracy and IFM. Our (BOAT-MI's) performance drops by about half after 24 incremental stages – much lower compared to SOTA methods.

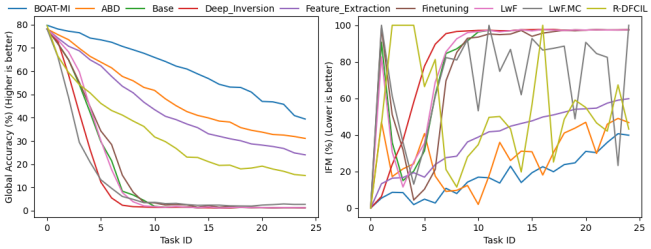

Figure 2: Global accuracy (%) and IFM (%) for $59 + 1 \times 24$ setup on EgoGesture3D dataset. Like other setups, our BOAT-MI excels the SOTA approaches here as well. Best viewed in digital format.

3. EgoGesture3D Processing

The new 3D skeleton dataset used in this paper is the derivative of the EgoGesture [8] comprising RGB-D images. Following the nomenclature of the parent dataset, we call it EgoGesture3D. Thanks to the MediaPipe [7] API that we use for 3D skeleton extraction from the images. The temporal association is turned on during the detection process with minimum detection and tracking confidences of 0.8 and 0.5, respectively. These numbers are chosen empirically for better extraction results.

However, merely using the extracted output from the temporally segmented gesture sequences is problematic for several reasons. First, the MediaPipe API fails to detect all the keypoints in all frames successfully. Next, it detects additional spurious hands (more than 2) in some of the frames. Also, in a few cases, for two hands detection, we find the handedness prediction to be the same (both left or both right hands). These issues are resolved with proper heuristics for each frame independently following the temporal detection with MediaPipe. Moreover, the baseline DG-STA model provides the optimal performance with 8 frames generally equally distant along the time dimension. Therefore, we save the samples with at least 8 frames of successful detections. In this regard, we experimented with interpolating with the less number of original frames. We find the oracle accuracy comparatively much lower that prevents us from saving samples with less than 8 frames. For the single-handed gestures, the values for the keypoints representing the absent hand are set to zeros. Very occasionally hand swapping occurs for just a few frames (i.e. 1-2 frames detect right hands with all others left). We keep these detections unchanged as some gestures may contain one hand mostly with the other in a cameo. Figure 3 and 4 show sample gestures in each column with the 2D projection of the skeleton on top of the corresponding RGB images.

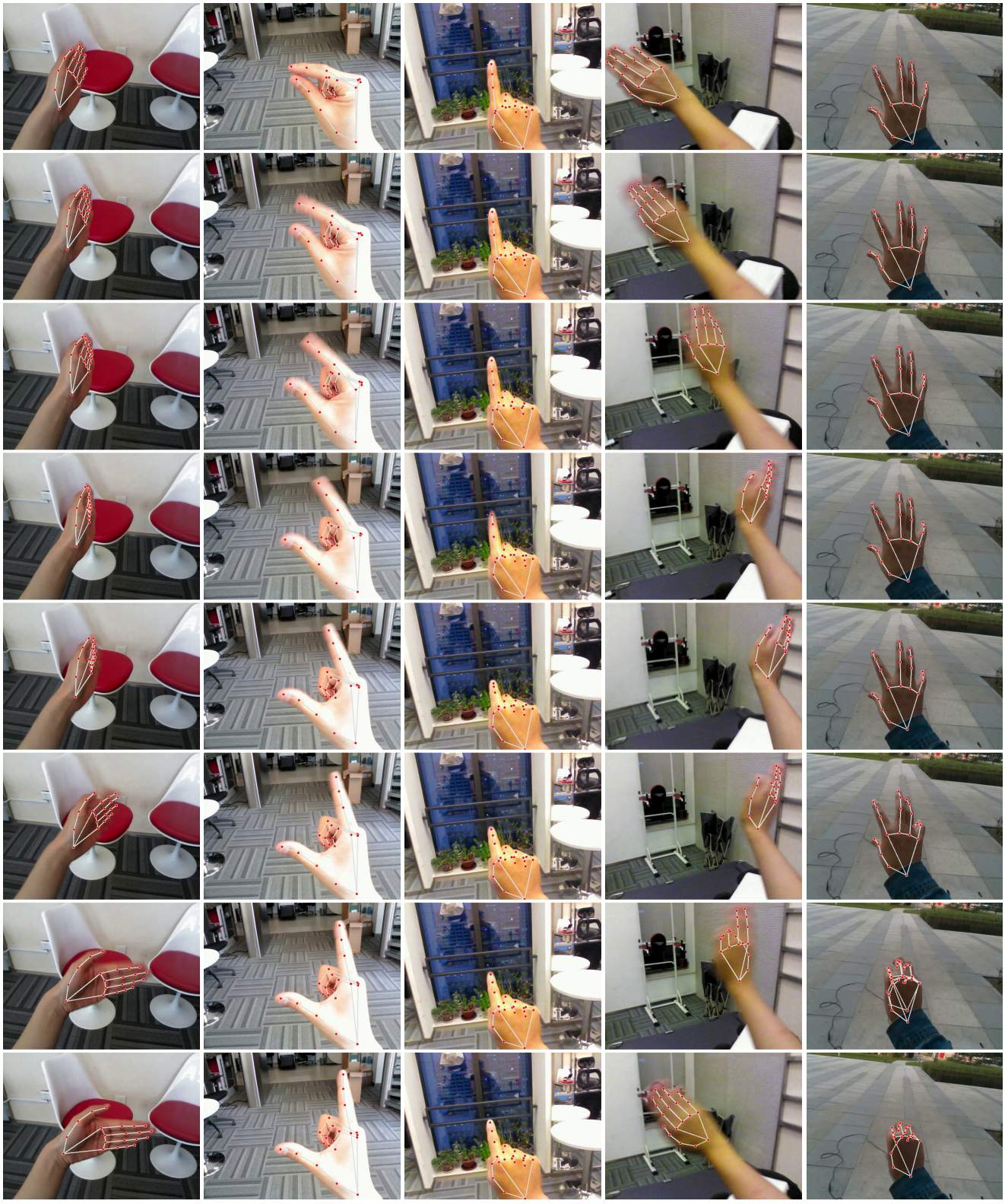

Figure 3: 2D projection of the EgoGesture3D skeletons on EgoGesture RGB images. (Top to bottom) Each column shows the temporal sequence for a single gesture (8 successfully detected frames selected in temporal order at random). (Left to right) [1] wave palm towards right; [12] zoom in with two fingers; [16] click with index finger; [39] wave hand; [48] grab.

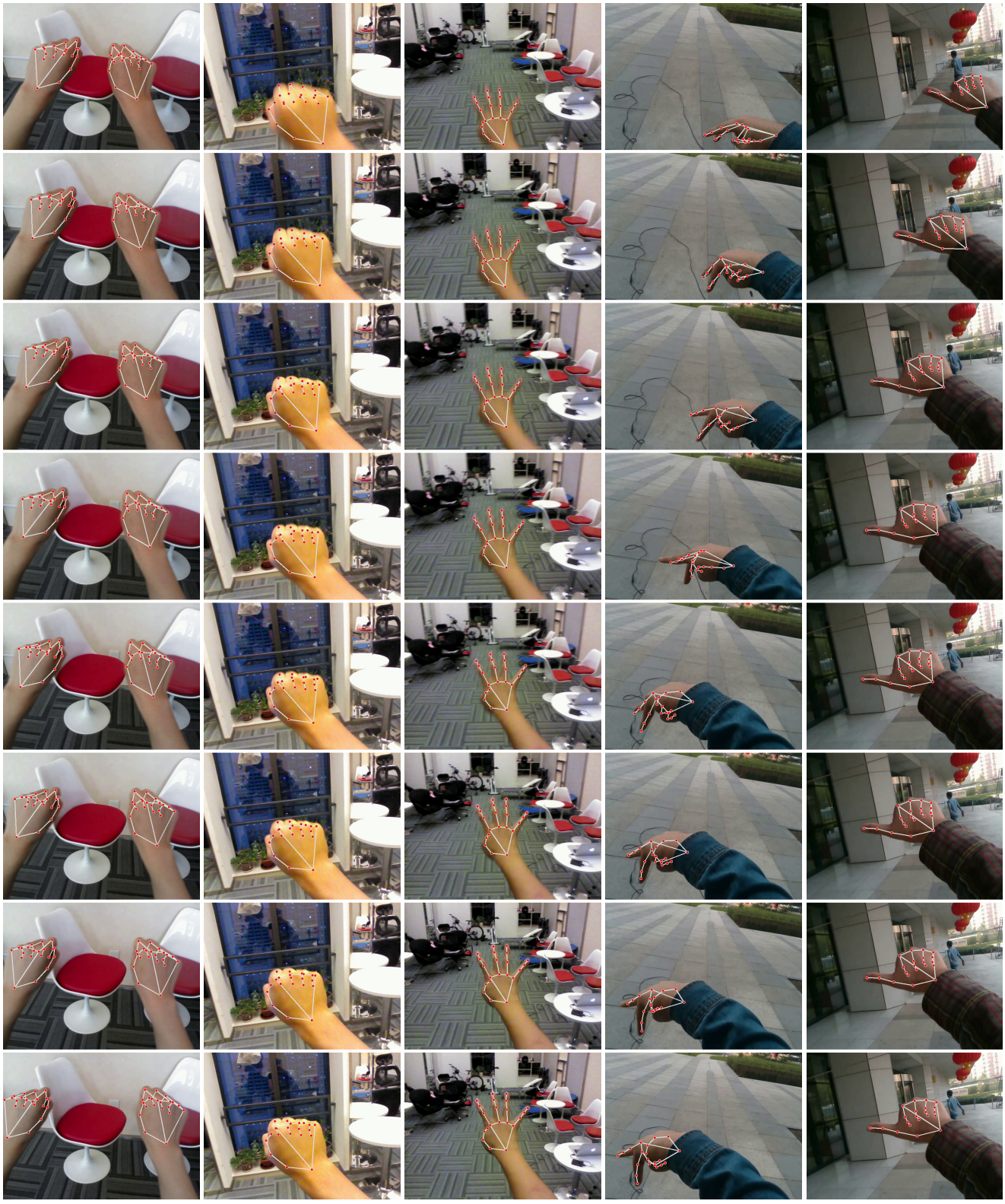

Figure 4: 2D projection of the EgoGesture3D skeletons on EgoGesture RGB images. (Top to bottom) Each column shows the temporal sequence for a single gesture (8 successfully detected frames selected in temporal order at random). (Left to right) [8] zoom in with two fingers; [21] static fist; [28] number 4; [49] walk; [66] thumb towards left.

References

[1] Yuxiao Chen, Long Zhao, Xi Peng, Jianbo Yuan, and Dimitris Metaxas. Construct dynamic graphs for hand gesture recognition via spatial-temporal attention. In *BMVC*, 2019.

[2] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM Conference on Computer and Communications Security*, 2015.

[3] Qiankun Gao, Chen Zhao, Bernard Ghanem, and Jian Zhang. R-dfcil: Relation-guided representation learning for data-free class incremental learning. In *ECCV*. Springer, 2022.

[4] James Smith, Yen-Chang Hsu, Jonathan Balloch, Yilin Shen, Hongxia Jin, and Zsolt Kira. Always be dreaming: A new approach for data-free class-incremental learning. In *ICCV*, 2021.

[5] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[6] Hongxu Yin, Pavlo Molchanov, Jose M. Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K. Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *CVPR*, 2020.

[7] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.

[8] Yifan Zhang, Congqi Cao, Jian Cheng, and Hanqing Lu. Egogesture: A new dataset and benchmark for egocentric hand gesture recognition. *IEEE TMM*, 2018.