LMU LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR DATENBANKSYSTEME
UND DATA MINING

Bachelor Thesis

in Computer Science

Trajectory Embedding Learning for Subspace Clustering

Oleg Tolochko

Aufgabensteller:	Prof. Dr. Thomas Seidl
Betreuer:	Zichong Xian
Abgabedatum:	27.08.2025

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

This paper was not previously presented to another examination board and has not been published.

München, 27.08.2025

.....

Oleg Tolochko

Abstract

Trajectory subspace clustering is a fundamental motion analysis task, where recently deep learning based approaches have shown promising results. However, their evaluation is problematic due to small and outdated benchmarks, such as the Hopkins155 [1] dataset, with no standardized evaluation protocols, leading to potentially overestimated results. This thesis conducts a deep dive into creating a reproducible and fair investigation for a state-of-the-art deep learning model with a focus on data augmentation and architectural modifications. We perform a faithful implementation and introduce a reproducible evaluation protocol for the Hopkins155 dataset, including a parent-wise splitting strategy and cross-validation. Within this framework, an extensive ablation study on data augmentation strategies, orthogonality regularization, and cross-trajectory attention in both supervised and unsupervised settings is performed. The results reveal that the choice of evaluation strategy is critical, as naive splits overestimate generalization performance. We find that augmentations are essential for the unsupervised learning objective and that small architectural modifications can significantly improve results. Furthermore, we find that the simplicity and size of existing benchmarks are primary bottlenecks for this field, which are not directly well-suited for the deep learning era.

Contents

1 Introduction

1.1 Motivation

For humans, the task of differentiating between different motions is effortless. For example, distinguishing between a moving car and a person walking in the opposite direction does not require us to take measurements of positions or velocities. However, for machines, this task of understanding motion is far from simple. In reality, tracked points extracted from videos are often noisy, partially occluded, and may involve multiple intersecting motions occurring simultaneously. Varying camera quality and camera movement additionally increase the complexity of the problem. Such factors make motion segmentation a significant computational challenge.

In this thesis, we will be assuming a simplified environmental setup. We are given a set of 2D trajectories tracked across F frames in a video sequence. Such video sequences may have hundreds, sometimes even thousands, of tracked points. An example of tracked points is displayed in Figure 1, with colors referring to the motion these points belong to.

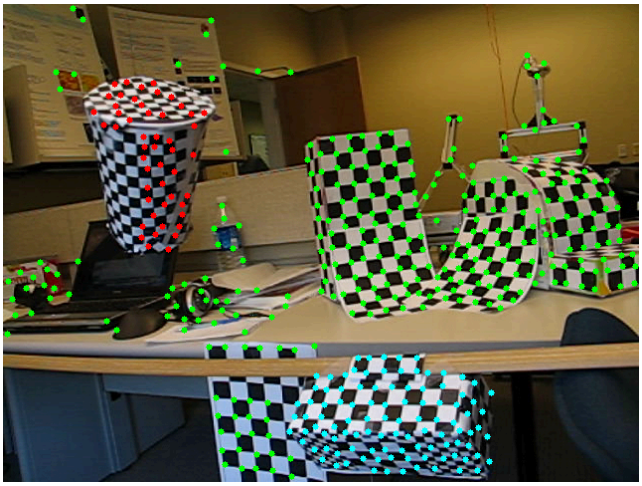


Figure 1: Hopkins155 Scene with tracked points. [1]

The essential objective is to partition the trajectories based on the motion they are following. This follows the assumption that all trajectories following a single motion can be represented by an unknown low-dimensional subspace.

This allows us to treat motion segmentation as a trajectory subspace clustering problem.

Traditional geometric and heuristic methods were the gold standard for this task. However, such methods struggle with complex and simultaneous motions, noise, and occlusion.

More recently, deep learning methods have gained increased popularity in the field of understanding videos and motion. Specifically, in the trajectory clustering domain, the paper "Learned Trajectory Embedding for Subspace Clustering" [2] has shown promising results. Though their evaluation leaves a lot of open questions about reproducibility and generalization.

This thesis will build upon the research presented in the aforementioned paper. A faithful re-implementation will be performed with a fully transparent and reproducible evaluation protocol. We investigate whether augmentations, orthogonality regularization, and cross-trajectory attention can improve learned embeddings in supervised and unsupervised settings.

1.2 Preliminaries

To create a common ground for the trajectory clustering problem, a set of prerequisites is necessary. Each trajectory is a set of 2D pixel coordinates over F frames:

$$x_j = \left[(x_{1j}, y_{1j}), (x_{2j}, y_{2j}), ..., (x_{Fj}, y_{Fj})\right] \in \mathbb{R}^{2\times F}$$

For consistency with subspace clustering literature, trajectories are flattened into $\mathbb{R}^{2\times F}$:

$$x_j = \left[x_{1j}, y_{1j}, ..., x_{Fj}, y_{Fj}\right]^T \in \mathbb{R}^{2F}$$

Accumulating P trajectories yields an observation matrix $M \in \mathbb{R}^{2F\times P}$, which describes P 2d point trajectories tracked over F frames:

$$M = [x_1, x_2, ..., x_P] \in \mathbb{R}^{2F\times P}$$

The objective is to partition trajectories into sub-matrices $S_i \subset M$, such that $x_j \in S_i$ only if the trajectory x_j is part of the motion S_i, so that the sub-matrices S_i represent different motions.

1.3 Benchmarks and Evaluation Protocol

To understand the performance of different approaches, a common evaluation setup is required. The most common dataset utilized for trajectory clustering objective is the Hopkins155 dataset [1]. It consists of 155 sequences with a varying number of trajectories and motions. It is assumed that it is known how many clusters we are targeting. This is a strict requirement by all to be mentioned methods, including the paper this thesis is building upon.

2 State of the Art

There are two main approaches to trajectory processing: heuristic and deep neural approaches.

2.1 Traditional and Heuristic Approaches

2.1.1 RANSAC Family

One of the most influential model families for geometric vision are RANSAC [3] based models. Such models can also be utilized in trajectory clustering objectives. Their core idea is to fit a model via random sampling of trajectories and inlier counting. A cycle of randomly selecting a set of samples, estimating model parameters, and counting inliers is repeated for a large number of steps. The model that has the most inliers is kept, and segmentation is performed sequentially. RANSAC based models are attractive due to their simplicity, but they rely on a greedy procedure that can fail if motions overlap strongly or the inlier ratio is low. Though such models are very sensitive to changes in inlier ratios and hyperparameters.

2.1.2 Subspace Clustering Methods

2.1.2.1 Generalized Principal Component Analysis (GPCA)

The Generalized Principal Component Analysis (GPCA) [4] fits a single polynomial to represent all subspaces at once. For c subspaces, it holds that:

$$\prod_{i=1}^{c}(\boldsymbol{b}_i^T \boldsymbol{x}_j) = 0$$

This means we are attempting to zero out the polynomial for all data points X. The approach directly attempts to find the subspaces from the

data, without knowing which point belongs to which subspace. In reality, we encounter noise, so points rarely ever lie exactly on the subspaces, which can make fitting the polynomial unstable in practice.

2.1.2.2 Local Subspace Affinity (LSA)

Local Subspace Affinity (LSA) [5] estimates a local subspace around each trajectory and then clusters with spectral clustering [6]. For each trajectory, it finds the k-nearest neighbors and fits a low-dimensional subspace U_i to it with Principal Component Analysis (PCA). Affinity between i, j for principal angles $\{\theta_l\}$ between U_i and U_j is defined as:

$$A_{ij} = \exp\left(-\sum_{l=1}^{d} \sin^2 \theta_l\right)$$

Finally spectral clustering on A is performed. Though LSA is sensitive to changes in the hyperparameters k, d, and noise. It is also slow to compute in comparison to other approaches [2].

2.1.2.3 Sparse Subspace Clustering (SSC)

Sparse Subspace Clustering (SSC) [7], on the other hand, doesn't attempt to fit a global polynomial but rather lets each point explain itself utilizing other points from the same subspace. The core idea is that any point x_i can be rewritten as a linear combination of other points from the same subspace. So for all trajectories $X = \left[x_1, x_2, ..., x_j\right]$ we solve:

$$\min \|c_i\|_1 \; s.t. \; x_i = X c_i, \; c_{ii} = 0$$

where $\|\cdot\|_1$ represents the L1-norm which motivates a sparse coefficient c_i. $c_{ii} = 0$ prevents the trivial solution of a point explaining itself. This produces the sparse coefficient matrix $C = \left[c_1, ..., c_j\right]$, which is utilized to construct the affinity matrix $W = |C| + |C|^T$, on which spectral clustering [6] is applied. SSC performs especially well in the trajectory subspace clustering domain. However, since we are solving an equation for each point x_i, it is computationally relatively expensive.

2.1.2.4 Low-Rank Representation (LRR)

Similarly to SSC, Low-Rank Representation (LRR) [8] attempts to let points from the same subspace explain each other. It tries to solve the problem:

$$\min_{Z,E} \|Z\|_* + \lambda \|E\|_{2,1}, \ s.t. \ X = XZ + E$$

It attempts to find the lowest-rank representation Z such that $X \approx XZ$. LRR separates errors into its own error matrix E, allowing it to handle outliers relatively well.

2.1.2.5 Robust Shape Interaction Matrix (RSIM)

Robust Shape Interaction Matrix (RSIM) [9] utilizes the Singular Value Decomposition of the data $X = U\Sigma V^T$. The r first right singular vectors V_r are selected, which are utilized for constructing an interaction matrix $V_r V_r^T$ on which spectral clustering [6] is performed. It is very fast [2] in comparison to previous methods and also significantly outperforms all other methods in the subspace clustering domain. Its only caveat is its susceptibility to the choice of rank r.

2.2 Deep Learning Methods

More recently, deep learning methods trying to solve the subspace clustering problem were proposed. The paper "Deep subspace clustering networks" [10] from 2017 was one of the first to utilize deep neural networks. It is structured as a deep autoencoder [11] with an additional self-expressive layer. This is a special layer that is supposed to enforce the self-expressiveness property directly in the network's latent space.

2.3 Position of Original Paper

The paper "Learned Trajectory Embedding for Subspace Clustering" [2] is the first paper to specifically challenge the domain of trajectory subspace clustering with a deep learning based method. It proposes state-of-the-art performance and surpasses other methods significantly in terms of computation speed.

The paper compares its methodology with other methods and provides an extensive performance comparison Table 1.

	Hopkins155					
	2 motions		**3 motions**		**All**	
Method	**Mean Error**	**Time**	**Mean Error**	**Time**	**Mean Error**	**Time**
GPCA [4]	4.59	324ms	28.66	738ms	10.34	417ms
LSA [5]	3.45	7.58s	9.73	15.96s	4.94	9.47s
LRR [8]	4.10	-	9.89	-	5.41	1.1s
SSC [7]	0.82	-	2.45	-	2.45	920ms
RSIM [9]	0.78	-	1.77	-	1.01	176ms
Original Paper	0.63	7ms	0.60	10ms	0.62	9ms

Table 1: Performance comparison on Hopkins155 dataset. [1], [2]

It claims to outperform all heuristic methods in terms of mean clustering error and speed for the Hopkins155 dataset [1].

3 Method of Original Paper

The paper "Learned Trajectory Embedding for Subspace Clustering" [2] proposes utilizing an autoencoder-like [11] architecture to extract motion information out of pure trajectories, such that these embeddings are self-expressive. Performing this extraction for each single trajectory in a given video sequence delivers a new subspace representation of the trajectories in the sequence.

Figure 2 gives a simplified overview of the model architecture. The first part of the model is the feature extractor, which takes a single trajectory and extracts a feature embedding f_i for it. This feature vector is supposed to represent the motion that the trajectory follows. The second part of the model is the subspace estimator, which is responsible for creating a mathematical representation B_i of the motion for a given feature embedding f_i.

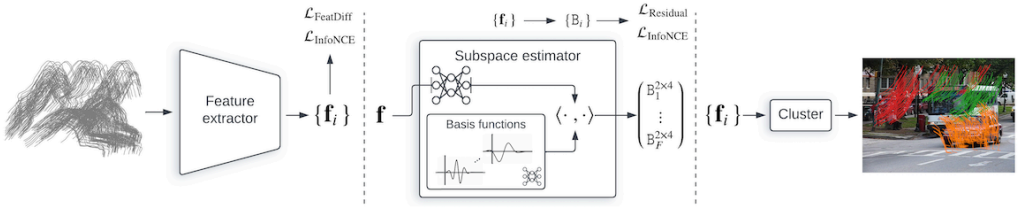


Figure 2: Model Architecture. [2]

3.1 Feature Extractor

The feature extractor takes a trajectory x_i of shape $(F, 2)$ and converts it into an embedding vector. This feature extraction is performed for all trajectories in a sequence, delivering the same amount of feature embeddings f_i as the original trajectories x_i. Trajectories belonging to the same motion should deliver identical feature embeddings, whereas feature embeddings from two separate motions should be mutually different. These feature vectors can be directly utilized for clustering with traditional clustering methods.

3.2 Subspace Estimator

The subspace estimator g_Φ acts as a decoder, generating a low-rank basis $B \in \mathbb{R}^{2F \times r}$ for a given feature embedding f. This basis is the mathematical description of the motion subspace. The number of basis functions is set to $N = 64$ and the rank is fixed at $r = 4$, in line with standard conventions for affine models of 3D motion [1], such as all heuristic models in Section 2.

The subspace estimator consists of two main parts:

1. A set of $N = 64$ learnable temporal basis functions, $\left\{h_j(t)\right\}_j^N = 1$, shared inside the model. It is defined as:

$$h^j_{\varphi(t)} = e^{-(\alpha_j(t-\mu_j))^2} \cos\left(\beta_j t + \gamma_j\right)$$

   where $(\alpha, \mu, \beta, \gamma)$ are learnable parameters defining the basis functions over normalized time $t \in [0, F-1]$.

2. A coefficient network Ω_ζ, which is a small Multi Layer Perceptron (MLP). It takes in a trajectory embedding f and predicts the coefficients needed to combine with shared basis functions to form the basis. The output is a matrix $\Omega = \Omega_\zeta(f) \in \mathbb{R}^{2N \times r}$.

The basis output can be rewritten as:

$$B = H_\psi(t)\Omega_\zeta(f_\theta(x)$$

where:

$$H_\psi(t) = \begin{pmatrix} h_\psi(t_1)^T & & \\ & h_\psi(t_1)^T & \\ h_\psi(t_F)^T & & \\ & h_\psi(t_F)^T & \end{pmatrix} \in \mathbb{R}^{2F \times 2N}$$

The basis B is utilized for two main tasks:

1. Trajectory reconstruction: $\tilde{x}_j = B(x_j)B(x_j)^\dagger x_j$, where $B^\dagger = (B^T B)^{-1} B^T$ represents the Moore-Penrose pseudo inverse [12]. This can create denoised versions of trajectories x_j.
2. Feature vector reconstruction: $v_i = \left(\text{flatten}(B_i)^T (f_i)^T\right)^T$, which can incorporate motion information into existing feature vectors.

3.3 Losses

Since the essential goal is that feature vectors coming from the same class should be the same $f_i \approx f_j$, where $\text{class}(x_i) = \text{class}(x_j)$, the original paper [2] introduces a modified version of the contrastive InfoNCE loss [13]. The proposed InfoNCE formulation in the paper suggests contrasting each positive pair (i, j) against a single randomly sampled pair (l, k). It is important to note that this is a major simplification in relation to other common InfoNCE loss variants, which generally sum over a set of negative pairs in the denominator, such as in the original InfoNCE paper [13] or the SimCLR paper [14].

Positive pairs p_{ij}, where the class is the same, should be large, and negative pairs p_{lk} should be extremely small, approaching 0. Given perfect separation p_{ij} is large and $p_{lk} = 0$ with $\frac{p_{ij}}{p_{ij}+p_{lk}} = 1$, thus $\log(1) = 0$.

$$L_{\text{InfoNCE}} = -\frac{1}{|D|} \sum_{(i,j,l,k) \in D} \log\left(\frac{p_{ij}}{p_{ij} + p_{lk}}\right)$$

$$p_{ij} = \exp\left(-\frac{\|f_i - f_j\|_2^2}{T}\right)$$

The second loss is the residual loss, which is a standard L2 loss that compares original trajectories to reconstructed versions:

$$L_{\text{Residual}} = \frac{1}{P}\sum_{j=1}^{P}\|x_j - \tilde{x}_j\|_2^2$$

Finally, a third feature difference loss is introduced, which is also a standard L2 loss, with feature vectors and reconstructed feature vectors:

$$L_{\text{FeatDiff}} = \frac{1}{P}\sum_{j=1}^{P}\|f_\theta(x_j) - f_\theta(\tilde{x}_j)\|_2^2$$

3.4 Training Setup

The original paper [2] proposes splitting up the training into two phases. The first phase solely trains the feature extractor f_θ utilizing the InfoNCE loss. The second phase trains the whole model (feature extractor + subspace estimator) $g_\Phi(f_\theta(x), t)$ with all 3 defined losses.

3.5 Issues

The original paper's results are promising, though several aspects limit the reproducibility and interpretability of the results.

3.5.1 Lack of Implementation Details

The first issue is the lack of a public implementation and incomplete training details. An additional supplementary document is provided with experiments and implementation details [2]. It specifies architectural decisions and some hyperparameters, but still leaves out a lot of crucial information, such as:

- Loss weights for multi-loss training
- The number of training epochs
- No splitting protocol and associated train/validation/test ratios

3.5.2 High Risk of Data Leakage

The original paper does not specify whether a train/validation/test split was performed, nor how sequences were allocated to each set. This is critical for small datasets such as Hopkins155 [1], where a large majority of the sequences in the dataset are derived from the same parent sequence. Without parent-wise splitting, data leakage is nearly unavoidable, which can artificially overestimate model performance.

3.5.3 Dataset Size

Hopkins155 consists of only 50 unique parent sequences [1], and KT3DMoSeg [15], another dataset utilized in their evaluation, only contains 22 sequences. The inclusion or exclusion of even a few sequences in the test set can significantly impact the reported performance. The paper fails to acknowledge such limitations, nor provides information about its evaluation proceedings. The authors mention "The proposed network was trained on the two motion segmentation datasets: Hopkins155 and KT3DMoSeg, yielding two different sets of weights" [2], which implies they only trained 2 models for evaluation.

3.5.4 Results Inconsistent with Prior Literature

The paper claims a mean clustering error of 0.63% for two-motion sequences and 0.60% for three-motion sequences on Hopkins155 [1]. These results are unusual, since three-motion sequences are generally more challenging. Moreover, the best heuristic baseline (RSIM, Section 2.1.2.5) claims a 1.77% mean clustering error for three-motion sequences, in comparison to a 0.78% mean clustering error on two-motion sequences. Without a clear evaluation protocol, understanding these results is challenging.

4 Research Questions

In consideration of the previously mentioned issues of the paper, the two main research questions that will be explored in this thesis are:

1. How important is a fair evaluation protocol for the generalizability of models?
2. How can we further improve the generalizability?
   - Can data augmentation techniques increase generalization performance?
   - Can orthogonalization improve the disentanglement of the learned motion bases as well as the generalization performance?
   - Does cross-trajectory attention improve generalization performance over single-trajectory encoders?

The evaluation is therefore performed on the Hopkins155 dataset [1] in a supervised and unsupervised setting.

5 Concepts

5.1 Fair Evaluation Protocol

Evaluation of deep neural approaches for trajectory subspace clustering comes with challenges, largely because of benchmarks such as Hopkins155 [1]. Additionally, deep learning methods are susceptible to overfitting. An evaluation protocol that can accurately measure a model's generalization performance is necessary.

This is especially critical in the Hopkins155 dataset. The dataset mainly consists of child sequences derived from parent sequences. A naive, random split could place a child sequence into the training set and another from the same parent into the validation set. In such a scenario, the model is not evaluated on truly unseen data. This leads to artificially inflated performance metrics and possibly a poor generalization estimate.

To address these issues, an evaluation protocol is constructed, focusing on three main principles:

1. Strict Parent-Wise Splitting: To prevent any data leakage, splits are not performed on an individual sequence level, but rather on the parent sequence level. All sequences belonging to a parent must be put into the same data partition. This ensures validation data represents a set of fully unseen sequences.
2. Robustness through Cross-Validation: For a small number of parent sequences (50 in Hopkins155), exchanging a single sequence between the train and validation set will impact performance. A single split might not correctly represent the model's average performance. Therefore, repeated random cross-validation is performed on a set of different splits. This allows reporting of mean, standard deviation, minimum, and maximum performance for a given configuration, providing a more reliable assessment of generalization performance. Additionally, in relevant cases, a paired samples t-test is performed to test the improvement significance of a given setup over a baseline.
3. Full Reproducibility: To ensure results are reproducible and fairly comparable across different studies, we use fixed random seeds for stochastic

processes, mainly for data splits. Furthermore, public access to the code and the exact splitting protocol is provided.

5.2 Data Augmentation

Deep neural networks require large, diverse datasets to mitigate overfitting and ensure good generalization performance. However, datasets in the domain of clustering on trajectories are very small. Consequently, this thesis explores methods for creating more artificial training samples, by means of augmentations, motivated by ideas presented in the SimCLR paper [14]. SimCLR utilizes different types of augmentations on image data for an unsupervised learning objective. Since in trajectory clustering, the data is frame-wise coordinate data, the two main compatible augmentations are point shifting and point occlusion.

This augmentation framework serves a dual purpose:
1. In the supervised setting, it acts as a regularizer to improve generalization performance.
2. In the unsupervised setting, it serves as the main learning signal for contrastive learning, where two trajectories from the same sequence are contrasted with each other.

5.2.1 Point Shifting

Point shifting can help with creating new trajectories for a motion. This can enforce the model to be less sensitive to specific numerical patterns in the data and focus more on actually representing a motion. There are three main ways to accomplish shifting with coordinate data. This can be formulated as a hierarchy of shifting:
1. Individual Point Shifting: Random noise application to a fraction of individual points across all trajectories in a sequence.
2. Point-wise Trajectory Shifting: Random noise application to all points in a subset of trajectories. This creates a set of entirely noisy trajectories.
3. Full Trajectory Shifting: Application of a random single offset to a subset of trajectories. This creates a set of shifted trajectories.
4. Full Sequence Shifting: Application of the same random offset to all trajectories in a sequence.

All represent viable ways of shifting and will be explored and tested.

5.2.2 Occlusion

Occlusion represents another way of augmenting data. Occlusions can effectively simulate information loss, such as objects getting hidden or tracking failing for a brief moment. This can add diversity and can also be thought of as a form of dropout.

There are two main occlusion methods. First, random point-wise occlusions, where individual points are randomly hidden. Secondly, grouped occlusions, where points are hidden in groups. This is arguably closer to reality, since we are rarely able to observe objects without them getting hidden or completely disappearing at some point.

An important assumption is that all points are visible at all times. Thus, all trajectories in a sequence are of the same length. This is a requirement for parallelized trajectory handling in the model and also needs to be taken into account when performing occlusion, since we can't just remove points. Setting the points to other values like -1 wouldn't work either, since we are working with coordinates, (0,0) is also a valid position. It would theoretically be possible to introduce another class for occluded points. This would make the training objective even more complex, since we now have an additional class.

Due to these limitations, occlusions will be performed by copying over the x and y coordinate values of the previous non-occluded point in the trajectory.

5.2.3 Unsupervised Learning

Building upon the already existing augmentation framework, the benefit of augmentations will be explored in the realm of unsupervised learning, such as in the SimCLR paper [14]. To make the architecture of the paper compatible with an unsupervised learning objective, we treat two different augmented versions of the same original trajectory as the same class, and all other trajectory pairs as negatives. The intuitive idea is that with meaningful augmentations, an augmented trajectory might represent another trajectory from the same class.

5.3 Orthogonality

The main part guiding the subspace estimator is the set of $N = 64$ basis functions $h_j(t)$. The basis functions serve as building blocks to construct complex motion patterns. To encourage the basis functions learning a disentangled motion representation, this thesis will explore a method of enforcing an orthogonality constraint. To accomplish this, a new loss is introduced. For a sequence length of F, each basis function is evaluated to form a vector $h_j \in \mathbb{R}^F$. These are stacked into a matrix $H = [h_1, ..., h_N] \in \mathbb{R}^{F \times N}$. The loss is defined as:

$$L_{\text{ortho}} = \|H^T H - I\|_F^2$$

Orthogonality is enforced along $\mathbb{R}^{\mathbb{F}}$. This means we strive for 64 mutually orthogonal basis vectors. Each basis channel and thus basis function is pushed to capture a different temporal pattern.

5.4 Cross-trajectory Attention

The original paper's architecture follows a relatively traditional autoencoder [11] architecture and is well designed for the domain of handling trajectories. However, it remains unclear if the architecture is optimal for this task. One main issue is that trajectories in a sequence never interact with each other. All trajectories are handled independently and are only ever considered as a group during clustering.

To combat this issue, this thesis will explore the potential of trajectories exchanging information between each other during feature extraction. Transformers [16] have enabled recent advances in large language models and generative models for images and videos. Attention is the main part of the transformer, which will enable trajectories to 'communicate' with each other and incorporate information from other trajectories into its own feature embedding. The inclusion of attention does increase computational load, which is $O(P^2)$, where P is the number of trajectories. However, for a dataset like Hopkins155 [1], where P is typically in the low hundreds, the cost remains manageable.

Specifically, the feature extractor will be extended with a transformer encoder, which is placed directly after the convolutional layers of the network, before the final feed-forward layer.

6 Experiments

All presented experiments were conducted by the author. The results in this section stem from a custom implementation of the described models and protocols [17].

6.1 Datasets

The paper utilizes 3 datasets in total to test the model's performance. This thesis will focus on one dataset, the Hopkins155 [1] dataset, which is the most commonly used and largest dataset available for trajectory clustering. It consists of 155 video sequences with 63-556 trajectories per sequence, an average of 296 trajectories per sequence, trajectory lengths between 15-100, with up to 3 separate motions per sequence. The dataset has 50 unique sequences in total. 35 of these sequences have 3 motions, and 15 have 2 motions. Figure 3 demonstrates this for an exemplary parent sequence, showing how simpler two-motion sequences are constructed from it.

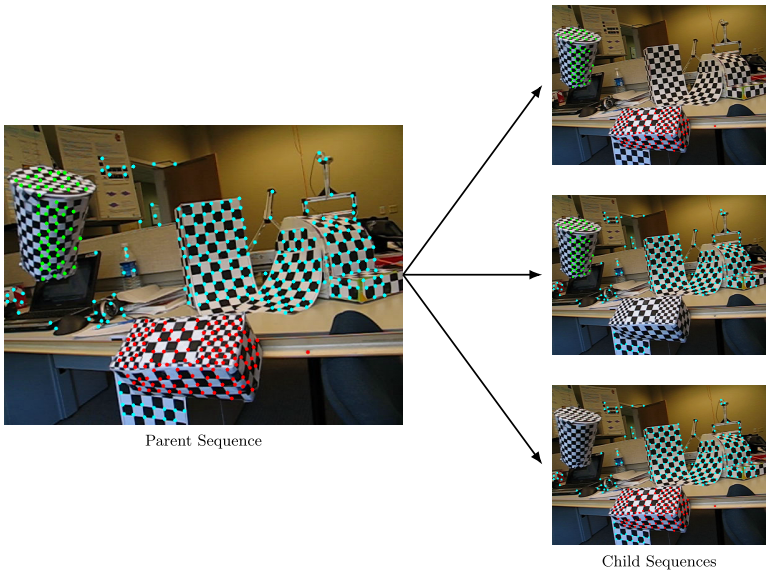


Figure 3: Example of a three-motion parent sequence and its three derived two-motion child sequences from Hopkins155 [1].

For every one of the 35 sequences with three motions, we include its three pairwise two-motion variants, resulting in $35 \times 3 = 105$ additional sequences.

This means that for a fair estimate of the performance on the dataset, with a deep learning model, we run a significant risk of data leakage. To ensure a fair and unbiased evaluation, all data splits must be strictly done at the parent sequence level.

Metric	**Value**
Total points	45,856
Stationary points (raw, 2.5% of img)	16.5%
Stationary points (camera-corrected, 1% of img)	25.0%
Stationary points (camera-corrected, 2.5% of img)	42.0%
Stationary points (camera-corrected, 5% of img)	64.0%
Mean point displacement	1.71% of image per frame
Average camera motion	1.40% $\pm$ 0.92% of image per frame
Camera-dominant sequences	107 out of 155

Table 2: Key statistics and characteristics of the Hopkins155 Dataset.

An additional trait of the dataset is that a large number of the points in the dataset are stationary. This is confirmed by the analysis in Table 2. 25% of points don't move more than 1% off their origin. The average movement of points is 1.71% of the image. Moreover, in 107 out of 155 sequences, the camera is the dominant movement. This means that the mean point displacement of an image per frame is smaller than the average camera motion per frame.

The dataset comes with both unnormalized coordinates and ones that are normalized in the range $[0, 1]$. During this thesis, only normalized coordinates will be utilized.

6.2 Original Paper Reconstruction

Before being able to perform any experiments, a fair re-implementation of the original paper is required. The implementation is performed in Python using PyTorch [18]. The architecture of the original paper is largely specified in the supplemental material [2].

6.2.1 Feature Extractor

The proposed architecture consists of three 1D convolutional layers, each followed by a ReLU activation function. The number of channels is expanded from 2 to 512 during this process. We apply max-pooling after the convolutional layers and flatten the embedding channel-wise, resulting in a 512-long vector, which is passed through 2 more fully connected layers, further reducing the spatial dimension to 128. The result is a 128-long embedding vector. This process is performed individually for each trajectory in a sequence to extract the motion information. To visualize the feature extractor, a simplified structure overview is provided in Figure 4.

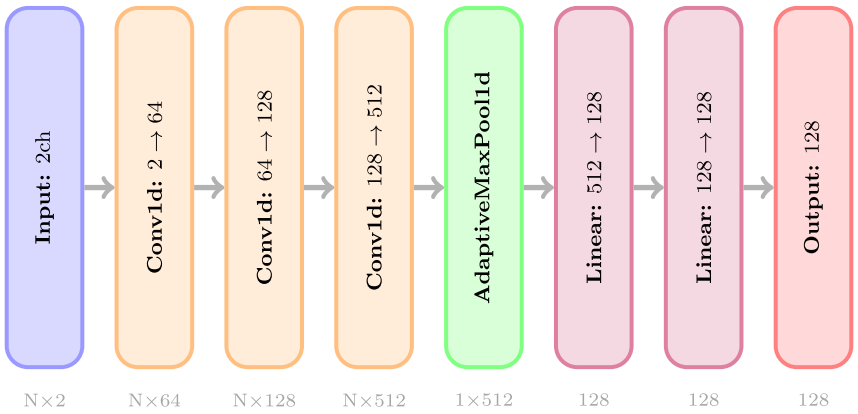


Figure 4: Feature Extractor Architecture.

6.2.2 Subspace Estimator

The subspace estimator generates a low-rank basis $\boldsymbol{B}$ for a given trajectory's motion based on its feature embedding $f \in \mathbb{R}^{128}$. It takes in two inputs: a feature embedding f and a normalized time vector t, where the elements are defined as $t_i = \frac{i}{F-1}$ for $i \in \{0, ..., F-1\}$.

As illustrated in Figure 5, the architecture has two main branches:

1. The Coefficient Branch: The feature embedding f is passed through three linear layers, producing a coefficient matrix $\boldsymbol{\Omega} \in \mathbb{R}^{128\times4}$.
2. Basis Branch: $N = 64$ learnable basis functions $h_j(t)$ are calculated at each of the F time steps t. This produces $\boldsymbol{H} \in \mathbb{R}^{F\times128}$.

The final basis $\boldsymbol{B} \in \mathbb{R}^{2F\times4}$ is constructed by the matrix multiplication ($\boldsymbol{H\Omega}$).

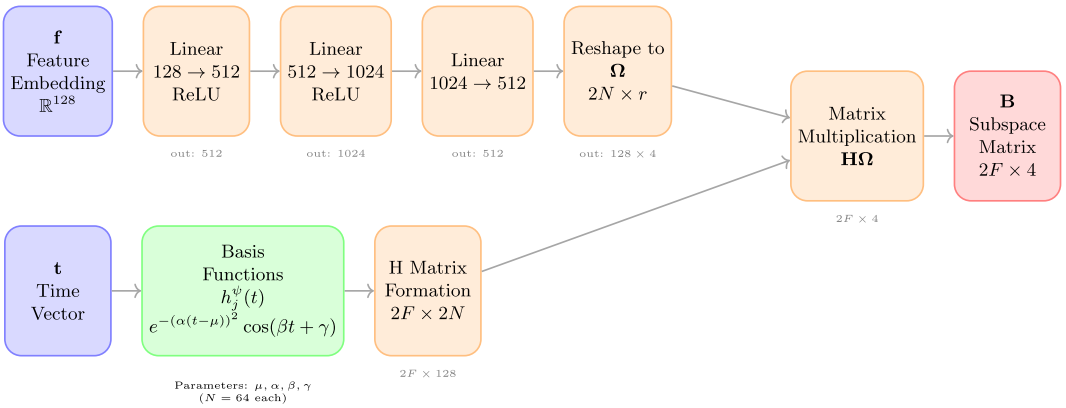


Figure 5: Subspace Estimator Architecture.

6.2.3 Clustering Algorithm

The paper [2] proposes utilizing hierarchical clustering as the clustering algorithm on extracted feature embeddings. Therefore, the 'AgglomerativeClustering' implementation given by scikit-learn [19] is used. The chosen linkage criterion is 'ward' [20], which minimizes the variance of clusters being merged. In agglomerative clustering, each data point starts in its own cluster. Clusters are iteratively merged together with their closest clusters based on which pair will result in the minimum increase in the total within-cluster variance. This procedure requires the number of target clusters to be known in advance, which is adopted for all experiments in this thesis. We repeat until the desired number of clusters is reached.

6.3 Experimental Setup and Evaluation Protocol

Due to the aforementioned evaluation issues with the original paper in Section 3.5, we want to ensure a reproducible and unbiased evaluation protocol. To accomplish this, fixed random seeds will be set, fair train-test splits to ensure no data leakage with respect to dataset constraints, and public code

and associated splits will be provided. The following protocol is used for all experiments.

To evaluate model performance, we use the Hopkins155 [1] dataset. Due to dataset size limitations, we do not perform fixed train-test-validation splits, since this would leave too few sequences for training and evaluation. We therefore utilize repeated random parent-wise cross-validation splits. In each run, the sequences are split into 80% training and 20% validation. The same 25 fixed splitting seeds are set and utilized for every model configuration, ensuring reproducibility and fair comparison. This choice is made to ensure that the variance between different model configurations is better quantified. The mean, standard deviation, minimum, and maximum error are reported for each model configuration.

For unsupervised evaluation, models are trained on the full dataset and validated on the full dataset. This follows the assumption that models are not utilized for further downstream tasks, which aligns with the setup of previous literature as seen in Section 2. Since we are not performing splits, we are less prone to randomness between two different models trained with the same setup. Thus, we choose to evaluate 10 trained unsupervised models per setup and report the mean, standard deviation, and minimum error.

Additionally, in relevant cases, a paired samples t-test [21] is performed to test the significance of a given setup over a baseline. For each of the splits, we calculate the difference in performance between the two models to compare. The test evaluates the null hypothesis H_0 that there is no difference between models, against the alternative hypothesis H_a that there is a difference:

$$H_0 : \mu_{\text{difference}} = 0$$
$$H_a : \mu_{\text{difference}} \neq 0$$

The resulting p-value quantifies the probability of measuring the difference as large as the measured one if the null hypothesis were true. If the p-value is lower than the chosen significance level of 0.05, the result is considered to be statistically significant.

The main evaluation metric will be the mean clustering error. For the calculation, the Hungarian (Munkres) matching algorithm [22] is utilized

for cluster assignments. This is done because integer class labels don't hold semantic meaning. For simplicity, the validation will always be performed on reconstructed feature vectors $\boldsymbol{v}_i = \left(\text{flatten}(B_i)^T \boldsymbol{f}_i^T\right)^T$.

To ensure results are reproducible and are easy to manage, a good configuration and monitoring setup is necessary. We utilize the 'Weights and Biases' (wandb) framework [23]. To determine a fair base set of parameters, Bayesian optimization [24] was performed. Thus, the following hyperparameters are fixed for all future experiments:

Hyperparameter	**Value**
Pretraining Epochs	50
Full Training Epochs	50
Learning Rate	0.00025
Weight Decay	1e-5
Scheduler Gamma	0.999
InfoNCE Loss Weight	0.1
Residual Loss Weight	1.0
Feature Difference Loss Weight	1.0

Table 3: Fixed hyperparameters used for all experiments, determined via Bayesian optimization.

6.4 Base Performance

6.4.1 Supervised Baselines

To test supervised performance on the base model, the mean clustering error is analyzed for strict train-validation splits, non-strict train-validation splits, and no train-test split at all, where the training data is the validation data.

Setup	**avg**	**min**	**max**	**std**
Hopkins155 strict train-val split	15.05%	9.55%	23.28%	3.51%
Hopkins155 non-strict train-val split	4.16%	1.04%	9.14%	1.81%
Hopkins155 no train-val split	1.08%	0.87%	1.43%	0.20%

Table 4: Analysis of mean clustering error for Hopkins155 [1] with 25 train-test splits on base model.

Table 4 displays the critical importance of strictly splitting based on parent sequences. By naively splitting, the model performance is severely overestimated, which is apparent by the comparison of the average of 15.05% for strict splits and the minimum of 1.04% for non-strict splits. Furthermore, the minimum of 1.04% for a naive split nearly matches that of a model evaluated on its training data (1.08%), strongly indicating data leakage. Even with strict splitting based on parent sequences, each model performs differently, which is quantified by the standard deviation of 3.51%. This underlines the importance of 25 random train-val split seeds per training setup.

This raises the question of why the model generalizes so poorly to unseen data. A possible explanation could be that most points don't move much between frames, quantified by the analysis in Table 2. This, combined with the high clustering accuracy on pure sequences, could mean that a significant amount of points are well-separated, not requiring the model to learn much about actual motions. To get a deeper understanding, multiple frames from a random sequence are chosen. This issue is demonstrated in Figure 6, which shows six frames from a representative sequence.

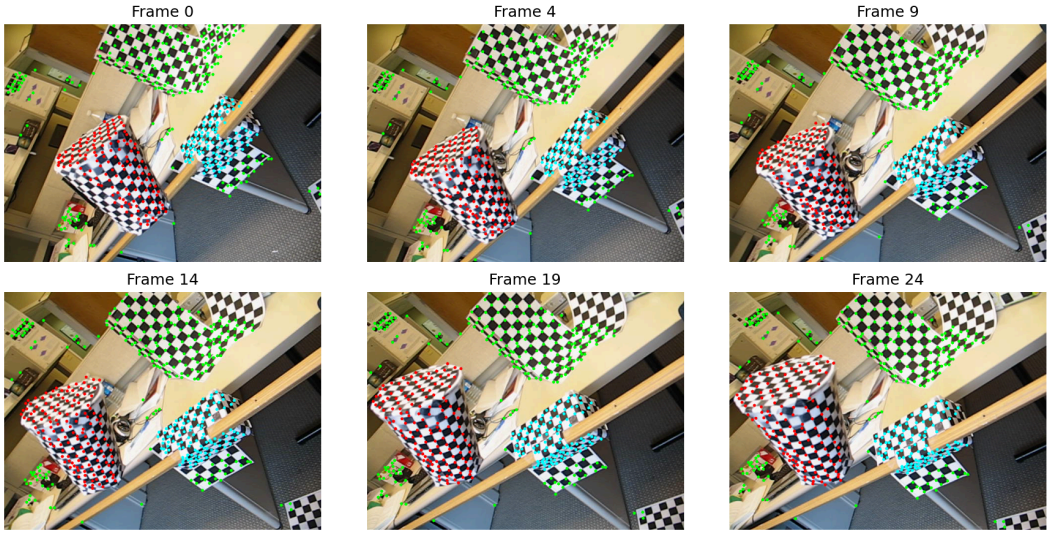


Figure 6: Six frames spanning the whole video from a Hopkins155 [1] sequence.

It is apparent that trajectories from different objects show little motion in comparison to each other. Most of the movement occurs through camera

movement in this sequence, aligning with the results in Table 2. For such sequences, the clustering task is simplified to spatial separation between groups of points, where a deep understanding of motion is not required.

6.4.2 Unsupervised Baselines

To create context for the unsupervised case, we establish three main baselines, summarized in Table 5. Each serves a different purpose for evaluating our approach.

1. Simple Unsupervised Baseline: This is the performance of the hierarchical clustering [20] on raw, unprocessed trajectories. This yields a mean clustering error of 20.05%. This is the primary target to beat, which any unsupervised model should be capable of outperforming.
2. No Augmentation Control: We train 10 unsupervised models, with no augmentation at all (positive pairs are identical copies of the same trajectory). The model performs very poorly with a mean clustering error of 31.07%. This clearly displays that augmentations are required for the contrastive learning process to work correctly.
3. RSIM, Heuristic State-of-the-Art: RSIM [9] represents the ceiling for this task with a 1.73% error.

Baseline Method	Description	Mean Error
Hierarchical Clustering	Simple baseline on raw trajectories	20.05%
Our Model (No Aug.) + Hierarchical Clustering	Control experiment to test learning framework	31.07% ($\pm$2.16%)
RSIM [9]	Heuristic State-of-the-Art	1.73%

Table 5: Performance benchmarks for the unsupervised learning task.

The results in Table 5 frame the goal for the unsupervised domain. The first primary goal is to develop an augmentation strategy that is capable of beating performance on raw trajectories. Getting an error of under 20.05%, with the ultimate goal of approaching the heuristic state-of-the-art.

6.5 Augmentations

The essential goal with augmentations is to motivate the model to better understand motions. To address the lack of data and to mitigate overfitting,

we explore augmentations to create more artificial trajectories during training, with the hope of increased generalization performance.

To ensure randomness instead of fixing augmentation setups, a maximum for all variables is set. This means that if, for example, the maximum shift percentage is set to 30%, for each processed sequence, anything between 0% and 30% is possible. This introduces more variance into augmentations, and in the unsupervised case, two augmentations are more likely to be mutually different. The two types of augmentations that will be tested are shifting and occlusions. With shifts, we mean shifting the x and y coordinates of 2D points in our trajectories. Such shifts may be performed individually for points, for full trajectories, or for whole sequences. Due to points being normalized between 0 and 1, points that exceed 0 or 1 through shifting are clamped to 0 or 1 accordingly.

Secondly, occlusions are tested, which means hiding points. In this specific setup, the trajectory lengths need to stay the same, so we can't remove points from trajectories. Due to this limitation, occlusions are performed by setting the coordinates of the points to occlude to the coordinates of the previous non-occluded point in the same trajectory. This is in line with the occlusion setup, utilized to derive additional sequences from the Hopkins155 dataset [1], often referred to as Hopkins12 in related literature [2].

6.5.1 Supervised Performance

6.5.1.1 Shifting

We investigate four types of shifting augmentations to test the model's sensitivity to noise, as well as whether a generalization performance increase can be achieved: random point-wise noise, trajectory-wise noise, full trajectory shifting, and global sequence shifting.

6.5.1.1.1 Individual Point Shifting

We test whether the addition of random noise to individual points across all trajectories can increase generalization performance.

	10% points shifted	20% points shifted	30% points shifted
±5% shift distance	$20.45\% \pm 2.48\%$	$20.86\% \pm 2.69\%$	$20.55\% \pm 1.83\%$
±10% shift distance	$20.54\% \pm 2.74\%$	$20.72\% \pm 2.52\%$	$20.13\% \pm 2.64\%$
±15% shift distance	$20.25\% \pm 2.68\%$	$20.42\% \pm 2.19\%$	$21.41 \pm 2.41\%$

Table 6: Average mean clustering errors ± std for 25 strict train-val Hopkins155 splits with point-wise shifting augmentation.

This type of augmentation significantly reduces model performance, suggesting that the model is sensitive to outliers in trajectories. This can be seen in Table 6, where the performance for all configurations is worse than the 15.05% baseline. The best attempt with 20.13% at a max of 30% points shifted and a max of ±10% shift distance, still was over 5% worse than the baseline. The feature extractor likely interprets the outliers as uninformative outliers, which hence corrupts learned embeddings. Thus, this approach is discarded in further supervised testing.

6.5.1.1.2 Trajectory-wise Point Shifting

Further, noise application to a subset of all trajectories is tested. This effectively creates $x\%$ noisy trajectories and $1 - x\%$ untouched trajectories.

	10% points shifted	20% points shifted	30% points shifted
±5% shift distance	$15.12\% \pm 3.25\%$	$14.30\% \pm 3.08\%$	$14.95\% \pm 3.12\%$
±10% shift distance	$14.62\% \pm 3.23\%$	$14.12\% \pm 3.63\%$	$14.98\% \pm 3.20\%$
±15% shift distance	$14.63\% \pm 2.96\%$	$14.85\% \pm 3.31\%$	$14.97 \pm 3.26\%$

Table 7: Average mean clustering errors ± std for 25 strict train-val Hopkins155 splits with trajectory point-wise shifting augmentation.

This method provides consistent and significantly better results than individual point shifts in Section 6.5.1.1.1. The baseline of 15.05%% was improved by nearly all configurations. The best configuration scored 14.12%, with a maximum 20% of points shifted and a maximum shift distance of 10%. To verify the improvement, a paired t-test [21] was performed comparing this configuration against the baseline across the 25 splits. The result yielded a p-value of $0.0068 < 0.05$, which means the observed improvement is statistically significant. The corruption of entire trajectories simulates variations within

motion classes, which possibly forces the model to learn more robust motion embeddings and thus explains the improved performance.

6.5.1.1.3 Trajectory-wise Shifting

Next, full trajectory shifting is tested. $x\%$ trajectories are chosen, and for each trajectory, all points are shifted by the same amount.

	10% traj. shifted	20% traj. shifted	30% traj. shifted
$\pm$5% shift distance	$14.66\% \pm 3.31\%$	$15.03\% \pm 3.21$	$15.03\% \pm 3.17\%$
$\pm$10% shift distance	$14.86\% \pm 3.08\%$	$15.11\% \pm 3.43\%$	$14.97\% \pm 3.13\%$
$\pm$15% shift distance	$14.88\% \pm 3.37\%$	$14.67\% \pm 3.19\%$	$14.86\% \pm 3.27$

Table 8: Average mean clustering errors $\pm$ std for 25 strict train-val Hopkins155 splits with trajectory-wise shifting augmentation.

Trajectory-wise shifting also performed well consistently with most configurations. Table 8 shows that the baseline of $15.05\% \pm 3.63\%$ is beaten out with almost all tested configurations. The best performing configuration improved the baseline slightly to 14.66%. However, a paired t-test comparing the setup with the baseline yields a p-value of $0.1445 > 0.05$, meaning we fail to reject the null hypothesis. This indicates the improvement cannot be seen as statistically significant. The model seems to only slightly benefit from seeing trajectories at different locations, forcing the model to learn representations that are invariant to absolute position.

6.5.1.1.4 Full Sequence Shifting

The final type of shifting to be tested is full sequence shifting. This augmentation shifts all trajectories in a sequence by a set distance.

10% shift distance	20% shift distance	30% shift distance
$20.80\% \pm 2.40\%$	$21.12\% \pm 3.29\%$	$20.84\% \pm 2.37\%$

Table 9: Average mean clustering errors $\pm$ std for 25 strict train-val Hopkins155 splits with full shifting augmentation.

This type of augmentation significantly degraded the model performance. As seen in Table 9, full shifting fell below the baseline of 15.05%. The best version with a shift distance of 10% is still more than 5 percentage points worse than the baseline. The poor performance can possibly be explained

by information loss due to coordinate clamping at the [0, 1] boundary. This approach is thus also discarded in further supervised testing.

6.5.1.2 Occlusion

We investigate two types of occlusion augmentations to see whether generalization performance can be increased: random point-wise occlusions and chunk-wise trajectory occlusion.

6.5.1.2.1 Point-wise Occlusion

First, random point-wise occlusions are tested. Inside a sequence, 2D points are selected at random and occluded.

10% of points occluded	20% of points occluded	30% of points occluded
$15.18\% \pm 3.14\%$	$14.91\% \pm 3.11\%$	$14.76\% \pm 2.90\%$

Table 10: Average mean clustering errors $\pm$ std for 25 strict train-val Hopkins155 splits with point-wise occlusion augmentation.

This method performed well relatively consistently. It only slightly beat out the set baseline of 15.05% as seen in Table 10 in almost all configurations. Occluding 30% of all points worked best with a 14.76% mean clustering error, a slight improvement of 0.29 percentage points over the baseline. However, this small improvement was not found to be statistically significant, as a paired t-test [21] granted a p-value of $0.4576 > 0.05$. Introducing random occlusion can be seen as a different way to introduce noise, which, compared to individual point shifting in Table 6, proved to be a non-destructive form of regularization.

6.5.1.2.2 Chunk-wise Trajectory Occlusion

Secondly, we test chunk-wise occlusion. For each trajectory, a total maximum of $x\%$ points are occluded. This occlusion is split into a maximum of N_c separate chunks, where the starting points of the chunks are randomly sampled along the trajectory's length.

	10% points occluded	20% points occluded	30% points occluded
1 chunk	$14.72\% \pm 3.26\%$	$14.82\% \pm 3.21$	$14.30\% \pm 3.04\%$
2 chunks	$15.67\% \pm 2.91\%$	$14.74\% \pm 2.94\%$	$14.60\% \pm 3.33\%$
3 chunks	$14.86\% \pm 3.21\%$	$14.88\% \pm 2.88\%$	$14.55\% \pm 2.59$
4 chunks	$15.01\% \pm 3.03\%$	$14.65\% \pm 3.43\%$	$14.74\% \pm 3.24$

Table 11: Average mean clustering errors $\pm$ std for 25 strict train-val Hopkins155 splits with chunk-wise trajectory occlusion augmentation.

Chunk-wise trajectory occlusions also performed relatively consistently. It beat out the baseline in most cases, as visible in Table 11, with 30% of points occluded and 1 chunk performing best with a 14.30% error, 0.75 percentage points better than the baseline. Nevertheless, a paired t-test [21] comparing the best setup with the baseline granted a p-value of $0.1105 > 0.05$. We thus also here can't reject the null hypothesis and can't consider the improvement to be statistically significant. The average performance improved consistently, albeit only by a small margin. The consistent improvement across multiple configurations could suggest a positive trend, but further investigation is required to statistically confirm its benefit. Chunk-wise occlusions may force the model to focus on embedding the motion as a whole, instead of focusing on point-wise changes in trajectories.

6.5.1.3 Augmentation Combinations

To see how combinations of augmentations work together, we greedily choose the individually best-performing version of each augmentation and combine these with each other. This is done due to the large number of possible combinations and the computation time required.

We test a variety of different combinations and provide an overview of the best-performing configuration, from a set of greedily chosen configurations, for each listed combination:

Augmentation Combination	Best Mean Clustering Error	Baseline Difference
Chunk-wise occlusion + Trajectory-wise point shifting	$14.58\% \pm 3.06\%$	-0.47%
Point-wise occlusion + Trajectory-wise point shifting	$15.00\% \pm 3.31\%$	-0.05%
Chunk-wise occlusion + Trajectory-wise shifting	$14.92\% \pm 3.15\%$	-0.13%
Point-wise occlusion + Trajectory-wise shifting	$14.51\% \pm 3.01\%$	-0.54%

Table 12: Average mean clustering errors $\pm$ std for 25 strict train-val Hopkins155 splits with greedily chosen augmentation combinations.

Interestingly, combining individually beneficial augmentations did not yield further improvements, sometimes even degrading the performance relative to the best single augmentations. The models do not seem to benefit from a larger variety of augmentations. This underlines the idea that modifying the original data too much can disrupt meaningful feature extraction. If, for instance, an occlusion is performed and then a random shift is applied to the same point, it could create artifacts that the model can't handle. It's thus important to find a balance in augmentation selection. Additionally, it is possible that the optimal hyperparameters discussed in Section 6.3 for combined augmentations differ from the best for the baseline.

6.5.2 Unsupervised Performance

In unsupervised learning, the model learns by contrasting differently augmented versions of the same trajectory (positive pairs) against versions of other trajectories (negative pairs). Performance thus heavily depends on meaningful augmentations. The essential goal is to find a set of augmentations strong enough to force the model to learn meaningful motion embeddings.

6.5.2.1 Insufficiency of Single Augmentation Strategies

Same as in the supervised case, we utilize the same augmentation setup. The evaluation setup follows the setup discussed in Section 6.3. We first evaluate each augmentation strategy individually and summarize the best result from each category in Table 13.

Single Augmentation Type	Best Mean Error (± std)	Best Min Mean Error
Individual Point Shifting	$28.30\% \pm 0.81\%$	25.95%
Trajectory-wise Point Shifting	$29.60\% \pm 1.70\%$	26.87%
Trajectory-wise Shifting	$29.16\% \pm 0.48\%$	25.68%
Full Sequence Shifting	$26.95\% \pm 0.85\%$	25.42%
Point-wise Occlusion	$28.82\% \pm 1.42\%$	25.68%
Chunk-wise Occlusion	$27.38\% \pm 1.71\%$	24.97%

Table 13: Summary of the best performance for each single augmentation strategy in the unsupervised setting with 10 models per configuration.

Table 13 clearly shows that not one of the single augmentation strategies is sufficient to produce a useful model. The best performing strategy, chunk-wise occlusion, achieved a mean error of 27.38%. The single best run across all experiments reached 24.97%, still significantly worse than the 20% baseline. Interestingly, during single augmentation strategy runs, stronger augmentations (e.g., larger shift distances) seemed to perform best, possibly suggesting that models benefit from more substantial augmentations.

6.5.2.2 Combined Augmentations

Since single augmentations seem to fail in producing useful models, combinations are tested. The hypothesis is that the application of multiple different augmentations for positive pairs (e.g., one trajectory is shifted, the other is occluded) creates a much harder task for the model, which forces it to learn better motion features.

First, we combine each shift version with each occlusion version, totalling 8 different evaluated pairs:

Augmentation Combination	Best Mean Error (± std)	Best Min Mean Error
Chunk-wise Occlusion + Individual Point Shifting	$26.72\% \pm 1.15\%$	24.45%
Chunk-wise Occlusion + Trajectory-wise Point Shifting	$29.00\% \pm 1.31\%$	27.11%
Chunk-wise Occlusion + Trajectory-wise Shifting	$27.95\% \pm 0.94\%$	26.28%
Chunk-wise Occlusion + Full Sequence Shifting	$25.86\% \pm 0.81\%$	24.66%
Point-wise Occlusion + Individual Point Shifting	$27.47\% \pm 1.33\%$	26.06%
Point-wise Occlusion + Trajectory-wise Point Shifting	$28.29\% \pm 1.20\%$	26.46%
Point-wise Occlusion + Trajectory-wise Shifting	$27.96\% \pm 0.94\%$	26.28%
Point-wise Occlusion + Full Sequence Shifting	$26.14\% \pm 1.21\%$	24.15%

Table 14: Summary of the best performance for each augmentation pair strategy in the unsupervised setting with 10 models per configuration.

Different from combinations in the supervised setting, combining augmentations does improve results over single augmentation strategies in most cases, albeit only marginally. As can be seen in Table 14, the baseline of 20.05% is still not beaten, with the best augmentation combination of chunk-wise occlusion with full sequence shifting scoring a mean error of 25.86%. The single best model scored an error of 24.15%, approaching the baseline of 20.05%. The results show that augmentations are effective in unsupervised learning, though they have yet to produce a meaningful model.

Since manually selecting pairs of augmentations only yielded marginal gains with the number of possible combinations being incredibly large, we thus conduct an automated hyperparameter optimization study, utilizing Bayesian optimization [24] with the Weights and Biases sweep framework [23]. We therefore perform a run with all augmentations enabled, totalling 200 trained models. We plot the evolution of the mean clustering error over the runs.

Figure 7: Development of mean clustering error for 200 model unsupervised hyperparameter tuning runs with all augmentations allowed over time.

As seen in Figure 7, the Bayesian optimization [24] progressively found better configurations. The single best model scored an error of 20.22%. This is a substantial improvement over manual combinations, bringing us closer to the baseline of 20.05%. The optimal strategy consistently utilized a combination of full sequence shifting (20-30% distance) and chunk-wise occlusion (4/3 chunks and 20-30% occlusion percent).

This investigation clearly displays the importance of contrastive learning for the unsupervised domain. Singular, simple augmentations are insufficient to learn meaningful features, whereas the combination of augmentations showed significant performance improvement. The fact that the exhaustive hyperparameter tuning run could not beat the hierarchical clustering [20] baseline of 20.05% suggests that the issue does not lie in the augmentation strategy, but rather in the model's architecture itself. Trajectories are processed in isolation, meaning trajectories can never exchange information with each other during inference. This is the direct motivation for the exploration of a cross-trajectory attention mechanism in the following sections.

6.6 Orthogonality Regularization

The subspace estimator depends on a set of $N = 64$ basis functions $\{h_j(t)\}_{j=1}^{64}$ to construct motion representations. These should capture unique and independent components of the motion. To encourage the separation of concerns, we introduce an orthogonality loss penalizing redundant basis functions.

For a sequence length of F, each basis function $h_j(t)$ can be represented by a vector $h_j \in \mathbb{R}^F$. These can be stacked together into a matrix $H = [h_1, ..., h_N] \in \mathbb{R}^{F \times N}$. To enforce orthogonality on these, we penalize the off-diagonal elements of the Gram matrix [25] $H^T H$ utilizing the Frobenius norm:

$$L_{\text{ortho}} = \|H^T H - I\|_F^2$$

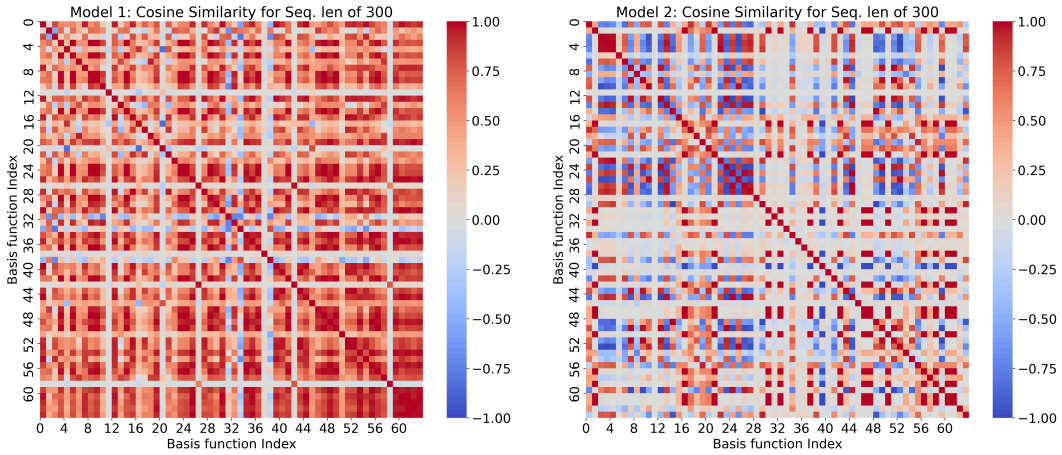
where I is the $N \times N$ identity matrix. The loss is added to the main training loss with a weight of λ_{ortho}.

Before testing out the performance, we verify whether the orthogonality constraint effectively enforces orthogonality. Subsequently, we evaluate the impact of the loss. We therefore perform a two-part experiment:

1. We test the effect on the base model
2. We test the interaction with our best-performing augmentation configurations

6.6.1 Verification of Orthogonality

To confirm that the loss function is working as intended, we visualize the cosine similarity matrix of the 64 basis functions for a model trained with and without the loss L_{ortho}.



Without Orthogonality Loss

Mean off-diagonal cosine similarity: 0.5164

With Orthogonality Loss

Mean off-diagonal cosine similarity: 0.3378

Figure 8: Cosine similarity matrices of 64 basis functions showing the effectiveness of orthogonality regularization for 2 models trained with an identical setup, besides the orthogonality loss.

Figure 8 verifies the effectiveness of the loss function. Without the loss, the basis functions show correlation between almost all basis functions (bright off-diagonal entries), quantified by a mean off-diagonal cosine similarity of 0.5164. In contrast, the model with the loss got a significantly smaller mean off-diagonal cosine similarity of 0.3378. Models trained with an orthogonality loss can thus produce a set of basis functions, where more are mutually different from each other.

6.6.2 Effect on Base Model

First, we test whether the orthogonality constraint can improve the base model performance. We therefore test out different weight losses:

$\lambda_{\text{ortho}} = 0.01$	$\lambda_{\text{ortho}} = 0.005$	$\lambda_{\text{ortho}} = 0.001$
$14.78\% \pm 2.93\%$	$14.80\% \pm 3.03\%$	$14.64\% \pm 2.81\%$

Table 15: Average mean clustering errors $\pm$ std for 25 strict train-val Hopkins155 splits with orthogonality constraint.

As shown in Table 15, enforcing orthogonality seems to improve model performance over the baseline of 15.05%. The best result of $14.64\% \pm 2.81$ achieved a mean improvement of 0.41 percentage points over the baseline. Under a paired samples t-test [21], the best configuration with a loss weight of 0.001 cannot be seen as statistically significant compared to the baseline, with a p-value of $0.2556 > 0.05$. The results suggest that regularizing basis functions via an orthogonality constraint could be useful, although not to a large degree.

6.6.3 Combination with Augmentations Supervised

To see how an orthogonality constraint combines with augmentations, we select the individual best configurations for all presented combinations in supervised evaluation.

	Mean Clustering Error (no orthogonality loss)	Mean Clustering Error (with orthogonality loss)
Chunk-wise occlusion + Trajectory-wise point shifting	$14.58\% \pm 3.06\%$	$15.29\% \pm 2.78\%$
Point-wise occlusion + Trajectory-wise point shifting	$15.00\% \pm 3.31\%$	$15.26\% \pm 2.95\%$
Chunk-wise occlusion + Trajectory-wise shifting	$14.92\% \pm 3.15\%$	$14.88\% \pm 3.50\%$
Point-wise occlusion + Trajectory-wise shifting	$14.51\% \pm 3.01\%$	$15.06\% \pm 3.37\%$

Table 16: Average mean clustering errors $\pm$ std for 25 strict train-val Hopkins155 splits with previously best augmentation configurations with an orthogonality constraint.

The results in Table 16 are relatively surprising and display a possibly interesting interaction. Adding an orthogonalization constraint to augmented models seemed to partially harm the performance. For example, with chunk-wise occlusion and trajectory-wise point shifting, the performance goes down from 14.58% to 15.29% when the orthogonality loss is added.

This may suggest that over-regularization is performed. Data augmentation on its own acts as a type of regularizer. The same goes for the orthogonality loss. When added together, these might be over-constraining the model,

preventing it from learning more complex patterns effectively. This also falls in line with the results in the augmentation combinations in Section 6.5.1.3, where the combination of too many augmentations would degrade model performance. This highlights a critical trade-off: more is not always better, with possible non-additivity between regularization techniques.

6.6.4 Combination with Augmentations Unsupervised

Further, we test whether an orthogonality constraint helps in the unsupervised case. We therefore perform an evaluation on all previously tested unsupervised augmentation pairs with an additional orthogonality constraint.

Augmentation Combination	Mean Error (no ortho loss)	Mean Error (w. ortho loss)	Error Difference
Chunk-wise Occlusion + Individual Point Shifting	$26.72\% \pm 1.15\%$	$28.20\% \pm 1.18\%$	+1.48%
Chunk-wise Occlusion + Trajectory-wise Point Shifting	$29.00\% \pm 1.31\%$	$30.99\% \pm 2.22\%$	+1.99%
Chunk-wise Occlusion + Trajectory-wise Shifting	$27.95\% \pm 0.94\%$	$29.52\% \pm 0.96\%$	+1.57%
Chunk-wise Occlusion + Full Sequence Shifting	$25.86\% \pm 0.81\%$	$26.75\% \pm 1.60\%$	+0.89%
Point-wise Occlusion + Individual Point Shifting	$27.47\% \pm 1.33\%$	$28.11\% \pm 1.63\%$	+0.64%
Point-wise Occlusion + Trajectory-wise Point Shifting	$28.29\% \pm 1.20\%$	$29.44\% \pm 1.28\%$	+1.15%
Point-wise Occlusion + Trajectory-wise Shifting	$27.96\% \pm 0.94\%$	$29.65\% \pm 0.22\%$	+1.69%
Point-wise Occlusion + Full Sequence Shifting	$26.14\% \pm 1.21\%$	$26.99\% \pm 0.52\%$	+0.85%

Table 17: Summary of performance for each augmentation pair strategy in the unsupervised setting with 10 models per configuration, comparing models with and without orthogonality constraint.

As apparent from Table 17, the orthogonality constraint degraded performance in all tested combinations. In all cases, the performance drop ranged from 0.64 to 1.99 percentage points, with an average degradation of 1.28 percentage points. The results align with the supervised case in Table 16,

where utilizing an orthogonality constraint in addition to augmentation would harm performance noticeably. This further supports the idea that more regularization techniques are non-additive. For such self-supervised tasks that solely rely on data augmentation, adding further constraints like orthogonality seems to be counterproductive, obstructing the model's ability to learn from the data itself

6.7 Cross-trajectory attention

One of the largest issues with the current architecture is that trajectories are all handled independently during inference. More often than not, a single trajectory on its own cannot hold enough information to fully describe a motion. This is especially an issue in the unsupervised case since cross relations between trajectories may hold valuable information. To combat this issue, the feature extractor is extended with Transformer encoder layers. The motivation behind this addition is that during the attention process, similar trajectories should deliver high Query-Key dot product values QK^T and thus high softmax scores. Thus, trajectories can understand how other trajectories move in relation to themselves and further enhance their own embedding of the motion. This would also presumably make the model less prone to noise, since single noisy trajectories may absorb enough information to fully discard possibly influencing noise.

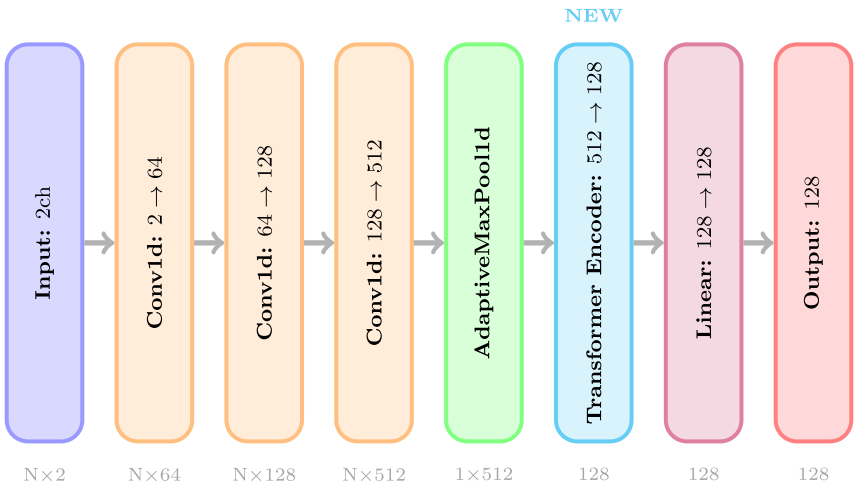


Figure 9: Feature Extractor Architecture with Transformer Encoder.

As seen in Figure 9, we replaced the first linear layer after the MaxPool layer with a Transformer Encoder. The number of Transformer Encoder layers is a hyperparameter that will be tested.

We utilize the same evaluation setup as previously.

Setup	avg	min	max	std
Base Architecture	15.05%	9.55%	23.28%	3.51%
Cross-Attention Architecture w. 1 Transformer Encoder Layers	10.74%	5.66%	14.45%	2.57%
Cross-Attention Architecture w. 2 Transformer Encoder Layers	12.08%	5.83%	17.78%	3.00%
Cross-Attention Architecture w. 4 Transformer Encoder Layers	16.55%	10.04%	23.74%	3.29%

Table 18: Average mean clustering errors ± std for 25 strict train-val Hopkins155 splits comparing the Base Architecture with a Cross-Attention Architecture.

The Table 18 shows that we are getting a significant improvement in the supervised domain. The best setup yielded an error of 10.74% with one transformer encoder layer, more than a 4% improvement in relation to the proposed architecture. To further quantify the improvement, a paired samples t-test [21] is performed comparing the base architecture performance with the new architecture. The test yielded a p-value of $p < 0.0001$, confirming that this architectural improvement is highly statistically significant. Interestingly, with the addition of more transformer encoder layers, the error goes up, indicating that the model is overfitting. We will thus, for further experiments, continue with only one transformer encoder layer.

6.7.1 Supervised Cross-trajectory Attention with Augmentations

We investigate the interaction of the previously best-performing augmentation strategies with cross-trajectory attention models. The goal is to find out whether the benefits, if any, are additive and can further push performance. The baseline for this experiment is the mean error of 10.74% for models trained without any augmentations. Similar to Section 6.6.3, we take the

individual best configurations from supervised evaluation in Section 6.5.1.3 on the base model architecture.

Augmentation Combination	Best Mean Clustering Error	Baseline Difference
Trajectory-wise point shifting + Chunk-wise occlusion	$15.26\% \pm 2.95\%$	$+4.52\%$
Trajectory-wise point shifting + Point-wise occlusion	$14.27\% \pm 2.54\%$	$+3.47\%$
Trajectory-wise shifting + Chunk-wise occlusion	$10.49\% \pm 2.33\%$	-0.25%
Trajectory-wise shifting + Point-wise occlusion	$10.63\% \pm 1.91\%$	-0.11%

Table 19: Average mean clustering errors $\pm$ std for 25 strict train-val Hopkins155 splits for models with one Transformer Encoder layer with greedily chosen augmentation combinations.

The results shown in Table 19 display a surprising interaction between the attention mechanism and augmentation. Trajectory-wise point shifting seemed to significantly degrade performance, whereas trajectory-wise shifting yielded good, consistent results. Trajectory-wise shifting with chunk-wise occlusion scored an error of 10.49%, slightly better than the cross-trajectory attention baseline of 10.74%. This improvement cannot be seen as statistically significant, though, with a paired t-test [21] yielding a p-value of $0.2922 > 0.05$. The previously best augmentation for the base model, on the other hand, increased the error by over 4.5 percentage points. Trajectory-wise point shifting introduces high-frequency noise that fully changes a trajectory's shape. This likely corrupts the features the attention mechanism relies on, causing it to propagate noisy information across embeddings. This experiment shows how an architectural change can change the effectiveness of data augmentation strategies.

6.7.2 Unsupervised Cross-trajectory Attention with Augmentations

Due to trajectories being able to exchange information with each other, the hope is that the model can learn better, more robust features from augmented

data alone. We therefore, similarly to Section 6.5.2.2, perform a Bayesian hyperparameter tuning run [24] with 200 trained models:

Figure 10: Development of mean clustering error for 200 cross-trajectory attention model unsupervised hyperparameter tuning runs with all augmentations allowed over time.

As displayed by Figure 10, the unsupervised performance increased significantly, with the best model getting 11.85% error, clearly beating out the baseline of 20.05%. The previously best unsupervised model had an error of 20.22%, 8.37 percentage points higher than the new best model. Most models even outperformed the mean error of 15.05% for the supervised base model. Interestingly, the best augmentation combination found in the hyperparameter search was exclusively a combination of chunk-wise occlusion with point-wise occlusion, with no shifting at all. This is different from unsupervised learning with the base model, where the best augmentation combinations included some version of shifting. Occlusion seems to be a more effective regularizer for this model architecture, where the model is forced to reconstruct motion from partial data in a global context, than the alteration of information via shifting.

This confirms the effectiveness of adding the attention mechanism to construct a global context for each trajectory. In the base model, trajectories

could only rely on their own limited information. With attention, it can absorb information from its neighbors and decide how to utilize the information.

7 Conclusion

This thesis conducted a reproducible and thorough investigation into deep learning for trajectory subspace clustering, examining the impact of data augmentation, regularization, and architectural modifications on the Hopkins155 benchmark.

The primary contribution was to provide a fair and transparent evaluation protocol that addresses data sparsity and its sensitivity to splitting strategies. We introduced a parent-wise splitting strategy to ensure data leakage between training and validation is prevented. Naive splits, without such a splitting strategy, severely overestimated the model's performance on unseen data. We additionally employed a random cross-validation strategy to ensure we are not prone to split randomness.

Within the framework, different data augmentation strategies were tested. Augmentations simulate possible motion variations and reality, where objects may get hidden in between frames or noise may creep in. The two main types were shifts and occlusions. The augmentations showed consistent performance gains in supervised learning, albeit minimal. For unsupervised learning, on the other hand, augmentations were crucial, enabling the model to learn meaningful features. Additionally, we found an over-regularization effect: an orthogonalization constraint was tested, which, on its own, increased performance, though if combined with augmentations, would often degrade performance. This highlights that the combination of different regularization strategies can have non-additive interactions.

Cross-trajectory attention was employed by adding a Transformer to the feature extractor, which yielded the largest gains. The gains were especially apparent in the unsupervised domain, where the best models would significantly beat out set baselines, with the exception of the state-of-the-art baseline set by RSIM [9].

However, the investigation also very clearly showed significant issues in the field of trajectory clustering. Relying on a small dataset such as Hopkins155 can be seen as a double-edged sword. While the simplicity of it allows for rapid and high variety in testing, the dataset is full of easy cases (e.g., stationary points), which can create the illusion of an effective model, even though it never learned meaningful motion representations for more complex motions. This clearly suggests that current benchmarks in the field of trajectory clustering are designed for an era of heuristic models, being insufficient for the modern deep learning models. The field lacks larger and more challenging datasets with standardized protocols. This increases the difficulty in comparing results and ensuring that models actually generalize.

It is questionable if findings based solely on the Hopkins155 dataset are extensible to complex, real-world scenarios with a larger amount of motions and non-fully visible objects. We rely on pre-computed fixed-length trajectories requiring a tailored setup, which is also the reason for the sparsity of datasets for this objective. It is not practical in reality to first extract point clouds and then order them, since this already requires knowledge of the motion itself.

For further research, multiple possible research directions emerge. Due to the significant gains from the Transformer addition, further research could involve exploring more complex Transformer-based models. Additionally, we found that the type augmentation is crucial. Thus, new augmentation techniques may be explored, like trajectory reversal or mirroring. However, based on the findings in this thesis, for further research, larger, more diverse, and more challenging trajectory clustering datasets are the primary requirement. These should have more diverse motions, complex camera paths, and non-stationary objects to allow for more sophisticated research. Furthermore, developing models that can handle more raw forms of input, such as unordered point clouds, like the Waymo Motion Dataset with Lidar-tracked point clouds [26], reduces reliance on perfect pre-processed data.

Conclusively, deep learning methods demonstrate a promising research direction for trajectory subspace clustering. Though their true potential is currently held back by outdated benchmarks. This thesis has shown how architectural adjustments and regularization measures can further push per-

formance, though a fundamental change in evaluation is needed. The future of trajectory clustering, therefore, depends not just on developing better models but on constructing better benchmarks to measure them.

Bibliography

[1] R. Tron and R. Vidal, "A Benchmark for the Comparison of 3-D Motion Segmentation Algorithms," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8. doi: 10.1109/CVPR.2007.382974.

[2] Y. Lochman, C. Olsson, and C. Zach, "Learned Trajectory Embedding for Subspace Clustering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 19092–19102.

[3] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, doi: 10.1145/358669.358692.

[4] R. Vidal, Y. Ma, and S. Sastry, "Generalized Principal Component Analysis (GPCA)." [Online]. Available: https://arxiv.org/abs/1202.4002

[5] J. Yan and M. Pollefeys, "A General Framework for Motion Segmentation: Independent, Articulated, Rigid, Non-rigid, Degenerate and Non-degenerate," 2006, p. . doi: 10.1007/11744085_8.

[6] A. Ng, M. Jordan, and Y. Weiss, "On Spectral Clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., MIT Press, 2001, p. . [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2001/file/801272ee79cfde7fa5960571fee36b9b-Paper.pdf

[7] E. Elhamifar and R. Vidal, "Sparse Subspace Clustering: Algorithm, Theory, and Applications." [Online]. Available: https://arxiv.org/abs/1203.1005

[8] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma, "Robust Recovery of Subspace Structures by Low-Rank Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 171–184, Jan. 2013, doi: 10.1109/tpami.2012.88.

[9] P. Ji, M. Salzmann, and H. Li, "Shape Interaction Matrix Revisited and Robustified: Efficient Subspace Clustering with Corrupted and Incomplete Data." [Online]. Available: https://arxiv.org/abs/1509.02649

[10] P. Ji, T. Zhang, H. Li, M. Salzmann, and I. Reid, "Deep Subspace Clustering Networks." [Online]. Available: https://arxiv.org/abs/1709.02508

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[12] A. Ben-Israel and T. N. E. Greville, "Generalized inverses: theory and applications," 1974. [Online]. Available: https://api.semanticscholar.org/CorpusID:121529370

[13] A. van den Oord, Y. Li, and O. Vinyals, "Representation Learning with Contrastive Predictive Coding." [Online]. Available: https://arxiv.org/abs/1807.03748

[14] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, "A Simple Framework for Contrastive Learning of Visual Representations," 2020. [Online]. Available: https://arxiv.org/abs/2002.05709

[15] Z. L. Xun Xu Loong-Fah Cheong, "Motion Segmentation by Exploiting Complementary Geometric Models," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[16] A. Vaswani *et al.*, "Attention is All You Need," 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf

[17] O. Tolochko, "Source Code for Trajectory Subspace Clustering Thesis." [Online]. Available: https://github.com/OlegTolochko/trajectory-subspace-clustering

[18] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035, 2019.

[Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[19] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] J. H. W. Jr., "Hierarchical Grouping to Optimize an Objective Function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963, doi: 10.1080/01621459.1963.10500845.

[21] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural computation*, vol. 10, no. 7, pp. 1895–1923, 1998.

[22] J. Munkres, "Algorithms for the Assignment and Transportation Problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957, Accessed: Aug. 25, 2025. [Online]. Available: http://www.jstor.org/stable/2098689

[23] L. Biewald, "Experiment Tracking with Weights and Biases." [Online]. Available: https://www.wandb.com/

[24] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., Curran Associates, Inc., 2012, p. . [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf

[25] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[26] S. Ettinger *et al.*, "Large Scale Interactive Motion Forecasting for Autonomous Driving: The Waymo Open Motion Dataset," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 9710–9719.