DISSERTATION

SCALABLE LEARNING OF ACTIONS FROM UNLABELED VIDEOS

Submitted by

Stephen O'Hara

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2013

ABSTRACT

SCALABLE LEARNING OF ACTIONS FROM UNLABELED VIDEOS

Emerging applications in human-computer interfaces, security, and robotics have a need for understanding human behavior from video data. Much of the research in the field of action recognition evaluates methods using relatively small data sets, under controlled conditions, and with a small set of allowable action labels. There are significant challenges in trying to adapt existing action recognition models to less structured and larger-scale data sets. Those challenges include: the recognition of a large vocabulary of actions, the scalability to learn from a large corpus of video data, the need for real-time recognition on streaming video, and the requirement to operate in settings with uncontrolled lighting, a variety of camera angles, dynamic backgrounds, and multiple actors.

This thesis focuses on scalable methods for classifying and clustering actions with minimal human supervision. Unsupervised methods are emphasized in order to learn from a massive amount of unlabeled data, and for the potential to retrain models with minimal human intervention when adapting to new settings or applications. Because many applications of action recognition require real-time performance, and training data sets can be large, scalable methods for both learning and detection are beneficial.

The specific contributions from this dissertation include a novel method for Approximate Nearest Neighbor (ANN) indexing of general metric spaces and the application of this structure to a manifold-based action representation. With this structure, nearest-neighbor action recognition is demonstrated to be comparable or superior to existing methods, while also being fast and scalable. Leveraging the same metric space indexing mechanism, a novel clustering method is introduced for discovering action exemplars in data.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Bruce A. Draper, for his guidance, encouragement, and for the opportunity to be involved in exciting research. His insight and feedback was invaluable for helping me to grow as a scientist. My thanks as well to Dr. Ross Beveridge, who provided explanations of many computer vision topics and encourages us all to be better at presenting our research. I am appreciative of my thesis committee members: Dr. Chris Peterson for numerous lectures on manifold geometry, Dr. Charles Anderson for introducing me to the field of machine learning, and Dr. Adele Howe for always encouraging precision of thought and expression.

I also am grateful for the collaboration, support, and advice given to me by my friends and colleagues in the department. Yui Man Lui, David Bolme, and Nayeem Teli were always ready to listen, providing helpful and challenging comments, and, at times, commiseration. The pizza was great, the company better.

I am deeply thankful to the support of my family. My parents promoted education, personal integrity, and achievement, while instilling confidence, love, and a healthy dose of self-reliance. My siblings are sources of inspiration and pride. My wife has taught me much about decisiveness, determination, and perseverance, and my daughter reminds me that life should be fun, too!

Finally, my sincere gratitude to the Computer Science systems administrators. Theirs is often a thankless task, yet I know how hard they work to keep the machines alive while our data processing jobs hammer on their clusters.

DEDICATION

This thesis is dedicated to my wife, Jodi. She is the love of my life and I’m proud to be her husband.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Beyond Forced-Choice Action Classification

1.1 Human Behavior Recognition

Emerging applications in human-computer interfaces, security, and robotics have a need for understanding human behavior from video data. Accurate and efficient human behavior recognition in streaming video is a challenge facing those desiring to automatically annotate a movie, to build systems that detect suspicious behavior in subway stations, or to build household robotics to assist the elderly. Behavior recognition requires the ability to detect subjects and objects of interest in a video, track their locations frame-to-frame, label what they are, what they are doing, and in what context. These are all open problems in the field of computer vision.

There has been notable success at using computer vision techniques for specific, domain-constrained tasks. Face recognition, under controlled illumination and pose, has been deployed by law enforcement and border security agencies. Contemporary video game systems have human computer interfaces that employ computer vision techniques in concert with special sensors (Microsoft Kinect) or illuminated markers (Sony PlayStation Move). Recognizing single-actor actions in controlled benchmark video data sets has shown notable performance gains in the past several years. For example, action recognition on the KTH Actions data set (see Figure 2.1 in §2.2) has risen from below 80% [SLC04] in 2004 to above 96% in 2010 [LBK10, GIK10].

Given the high recognition rates cited in action recognition benchmarks, are fielded applications of human behavior recognition from standard monocular video cameras far away? Perhaps motivated by a similar analysis, in 2010, the Defense Advanced Research Projects Agency (DARPA), funded a large program called "Mind's Eye" [Gel11]. The goal of the Mind's Eye program is to develop smarter surveillance cameras that can recognize important events and behaviors in video and alert operators with concise summaries. Much of the research presented in this thesis was motivated by, and performed as part of, a Mind's Eye grant awarded to Colorado State University.

Significant challenges remain in trying to adapt action recognition techniques developed using controlled benchmark data to less structured and larger-scale data sets, such as that provided by the Mind's Eye program. Those challenges include: the recognition of a large vocabulary of actions, the scalability to learn from a large corpus of video data, the need for real-time recognition on streaming video, and the requirement to operate in settings with uncontrolled lighting, a variety of camera angles, dynamic backgrounds, and multiple actors.

In an attempt to address these challenges, this thesis focuses on scalable methods for classifying and clustering actions with minimal human supervision. Unsupervised methods are emphasized in order to learn from a massive amount of unlabeled data, and for the potential to retrain models with minimal human intervention when adapting to new settings or applications. Because many applications of action recognition require real-time performance, and training data sets can be large, scalable methods for both learning and detection are beneficial.

This thesis presents a set of contributions, as summarized later in this chapter, that leads to the development of a scalable unsupervised method for action recognition. The ultimate goal is to develop a system that learns and recognizes actions, in real-time, from streaming video sources with minimal human guidance. We are not there yet. But the methods proposed in this thesis may get us closer to the goal. In order to reduce confounding factors, evaluations of the specific contributions are performed using controlled benchmark data sets. We later tie together the individual contributions by providing an overview of the impact made to the Mind's Eye program, which features many of the challenges discussed earlier.

1.2 Motivation for Unsupervised Learning

High-definition video consists of millions of pixels per video frame, 30 frames per second, leading to gigabytes of data for just minutes of observation. Applying supervised training methods to video data has significant logistical challenges, notably in terms of generating a broad set of labeled data. Video data annotation is very time consuming.

Consider generating annotations for a set of surveillance videos for the purposes of recognizing just four verbs: "chase," "dig," "give," and "turn." It is not sufficient to simply have a user click one of four buttons whenever they observe the associated verb. The video could show subjects performing these and other non-relevant behaviors simultaneously. A single subject could be exhibiting more than one behavior at once, such as turn and chase. One must annotate which subjects in the video were involved in a given behavior, where they were, and when. This requires the tedious drawing of bounding boxes around subjects to allow for recording which verbs are applicable to each subject at each time step. While tools can help mitigate the tedium of this task (i.e., [Uni12]), the level of human effort involved is still quite high.

Another challenge is label consistency. There can be ambiguity in determining exactly when a behavior occurs. Consider the verb "give." Does the "give" start when the object held by the first person begins moving towards the second? When does it end? Perhaps "give" is defined as the very instant of time when the first person lets go and only the second person is holding the object? But constrained only to that moment in time, is there enough context to even recognize the action by a human observer? There are ambiguities in the semantics of verbs as well. Does "dig" actually require the moving of soil, or would it be considered digging if a person clears a superficial amount of dirt using a trowel? What if a dog is observed digging? What about if the digging is being done by a backhoe operated by a person? Is the person digging or is it the machine?

To deal with the time-consuming nature of detailed image and video annotation, some researchers enlist large groups of people, via a crowd-sourcing service such as Amazon Mechanical Turk (as in [JKJ$^+$11]). However, label consistency drops when a large group of people is involved. Even with only a few people that deeply understand how label quality and consistency affects machine learning, there can be label drift when the annotation process takes many days.

Consider what happens even when this approach yields positive results. Assume the time and effort is expended to annotate training videos, and some method of supervised learning is employed that successfully detects or recognizes the actions in new videos presented to the system. What happens when the need arises to detect a new set of verbs? Or when the same set of verbs needs to be recognized, but in a different context where the current model fails? In the first case, new annotations are required over the training set, and the system must be retrained. In the second case, a whole new data set is required to be collected, and then the developer must start from scratch with the annotation and training process.

If learning from video can be accomplished with a greatly reduced need for annotations, then it may be possible to more rapidly extend trained models to cover additional actions and to adapt methods for a variety of applications and domains. This thesis emphasizes the role of unsupervised learning for action recognition. Unsupervised learning mitigates both the annotation overhead as well as the label consistency problem. Instead of deciding a-priori what labels to learn in a data set, a clustering analysis of the data generates the vocabulary of actions which can be supported by the data. There are many challenges with unsupervised action learning, some of which are discussed further below and addressed in the technical contributions of this thesis.

1.3 Speed and Scalability

In moving beyond action classification benchmarks, one challenge that must be faced is scalability. A method which has a high computational complexity may perform well on a small enough data set, but may be infeasible for use with real-time streaming video sources.

Speed and scalability are important constraints for most applications. For video surveillance, human-computer interaction, and robotics applications, real-time processing is required. There are scalability concerns for both training and run-time aspects of a method. If the learning process has a high computational complexity, it can be too time consuming to learn from large amounts of data. Generally, data clustering works best when there are many data samples. Being unable to process large-scale corpi may prevent unsupervised methods from discovering those actions that appear less frequently in the data.

If run-time recognition speed has a dependency on data set size, or another attribute such as the number of entities in each frame of video, then such a method may appear fast when tested on a single-actor short-duration data set like KTH Actions, but slow when applied to streaming surveillance video.

1.4 Scope of Work

This thesis makes contributions towards moving the field of action recognition beyond the dominant paradigm of forced-choice action classification methods, the distinction of which is discussed further in the Background chapter. The following is a high-level summary of the technical body of this work. Significant contributions are enumerated afterwards.

1.4.1 How well do actions cluster with existing representations?

This thesis first seeks to understand how contemporary action representations lend themselves to unsupervised learning, where the set of possible action labels is learned from the data. Specifically, a relatively new manifold-based representation [LBK10] is compared with a Bag of Features representation based on the sampling of localized space-time features [DRCB05]. Different labelling choices of benchmark data are evaluated to measure how well clusters formed using these representations align to the label choices.

The following is a key consideration for unsupervised learning. Without supervision, grouping of videos will be determined solely by the biases of the representation and associated distance metric.

As a simple example, consider three video clips, as illustrated in Figure 1.1. The first depicts a person walking from right to left. The second depicts the same action, but in a different direction, diagonally across the screen. The third depicts the same person jogging in the same diagonal direction. When the goal is to group by action label, ideally the first and second video clips would be grouped separately from the third (walking vs. jogging). The second and third videos have the actor moving in the same direction, so it would be understandable, but irrelevant to the goal, if they were clustered along this aspect of similarity (motion direction).

In considering this example, it seems reasonable to predict poor alignment between cluster membership and a set of desired class labels selected a-priori. The clusters could form just as easily based on distinctions which are irrelevant to the desired labels. Furthermore, a given linguistic label could present itself visually in a multitude of ways.

To understand how representation bias impacts action clustering, Chapter 3 of this thesis ex-

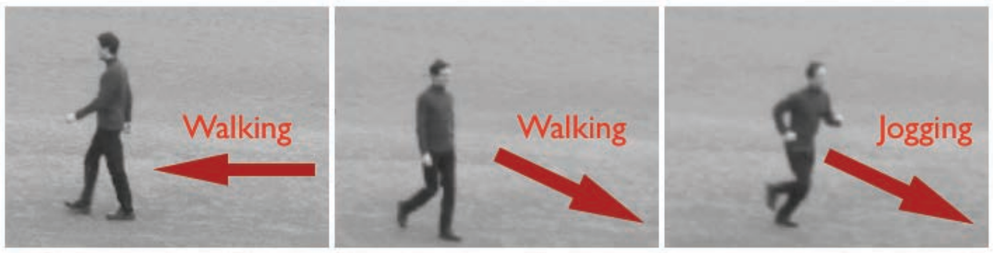


Figure 1.1: Three examples from KTH Actions data set [SLC04]. From left to right, the first two show the same actor walking, but in different directions. The last two show the same actor moving in the same direction, but performing different actions. Without supervision, would clustering these samples cause them to group together based on direction of motion or the action being performed? This question is addressed in Chapter 3.

plores how alternative labelings of benchmark data align with a given aspect of similarity among the video clips. The results from this study suggest that a manifold-based representation developed by Lui et al. [LBK10], called the Product Manifold (PM) representation, may be more amenable to clustering along the desired semantics than a popular approach based on histograms of localized space-time features [DRCB05]. The PM representation is demonstrated to cluster data from the popular KTH Actions benchmark (Figure 2.1, middle) that are over 90% aligned with the class labels.

1.4.2 Unsupervised action recognition on streaming video

The challenge of learning and recognizing actions from continuous video streams is addressed in Chapter 4. This chapter presents an approach for streaming video action recognition based on detecting and tracking the actor in the video and using temporal windows to select a few seconds of video localized around the person for classification. Unlike with pre-segmented benchmark data, classifying the tracked video segments includes errors due to inaccuracies involved in detection and tracking, which remain open problems in computer vision.

With our approach, one first discovers the set of observed actions based on sampling a set of unlabeled training videos. Detection and tracking algorithms are applied to the training videos to extract "tracklets," which are defined as short-duration segments of video localized around a tracked subject. The set of tracklets are clustered using the PM representation.

Each cluster is assigned an appropriate action label by a human annotator. The clusters are formed without supervision, and they define the actions learned from the video, but for the sake of output, a human-assigned label has to be provided at some stage. That is, unless users would be content with the machine employing cluster numbers as labels, such as “action 25 occurred from frames 250–310.” Generally speaking, the effort required to label K clusters is much less than the effort required to label N samples, where $K \ll N$.

To recognize actions in a new video, the detection and tracking algorithms are applied as before to yield new tracklets. Each tracklet is matched to the K-nearest clusters and assigned a label based on the labels of the matching clusters. This proposed method has advantages over competing approaches in being unsupervised, tolerating minor detection and tracking errors, and allowing for multiple labels per tracklet.

A challenge left unaddressed by Chapter 4 is scalability, as the number of tracklets being clustered is less than 300. Clustering scalability using a manifold-based action representation is addressed in later chapters, after first introducing a novel data structure used for nearest-neighbor action classification. This structure is then used as the underpinnings for scalable unsupervised action recognition.

1.4.3 Action classification via Subspace Forests

Using the PM representation, action classification is done via nearest-neighbor selection. In small benchmark data sets, this poses no problem, but when applied to large-scale applications with many thousands of samples, nearest-neighbors may not allow for real-time performance, especially when the distance function requires 10-100’s of milliseconds per comparison, as is the case with the PM representation.

To speed up similarity queries, one can employ an Approximate Nearest Neighbor (ANN) algorithm that identifies neighbors in logarithmic time complexity. The most popular ANN algorithms used in computer vision applications, such as those implemented in the Fast Library for Approximate Nearest Neighbors (FLANN) [ML09], are predicated upon a vector-space data representation. However, for non-Euclidean data equipped with a distance metric, ANN indexing can be accomplished using structures designed for partitioning general metric spaces [Uhl91, Yia93].

A question which arises is whether ANN indexing accomplished using general metric-space structures works as well as the more popular methods employed in contemporary computer vision applications. If so, then would not the more general-purpose method be preferred over methods restricted to vector data?

In Chapter 5, an ANN indexing structure is introduced which can be used on any metric data. This structure is called a Proximity Forest. The Proximity Forest is compared to leading ANN methods on well-known vector data representations. The results strongly suggest that not only is the Proximity Forest a more general-purpose data structure, but that it produces significantly more accurate results on data representations of wide interest to the computer vision community.

Chapter 6 presents a special-purpose implementation of a Proximity Forest for action recognition called a Subspace Forest. Inspired the PM representation, the Subspace Forest can index actions as represented as points on Grassmann manifolds. This is significant because most methods for ANN indexing assume the data lives in a vector space, and thus can not be directly applied to non-Euclidean manifold representations. The Subspace Forest has the scalability advantage of the Proximity Forest, allowing for real-time action recognition, while matching or besting the accuracy of competing methods on benchmark data.

1.4.4 Scalable clustering of arbitrary metric data and unsupervised action recognition

In Chapters 3 and 4, using the PM representation for clustering involves first computing an all-pairs distance matrix, followed by a hierarchical clustering process using the distance matrix as input. The all-pairs comparison was used because of the non-Euclidean nature of the representation. (More details on the Product Manifold representation follow in Chapter 2.) Construction of the all-pairs distance matrix is an $O(N^2)$ operation, where N is the number of tracklets, and each distance computation requires about 100ms.

Computing the distance matrix on a relatively small data set consisting of 2,500 tracklets takes about 3 days on high-end commodity computers available in 2011. To put that in perspective, consider a hypothetical 10-minute surveillance video recorded at 30 frames per second which averages two tracked people per frame. Depending on implementation details described later, this 10-minute video might produce 2400 tracklets. At this rate, a few hours of surveillance video used for training would generate tens of thousands of tracklets. Being able to cluster samples from long-duration video without having to throw away the majority of the data is infeasible given $O(N^2)$ distance computations, each taking 100 ms. Typically with clustering, the more samples the better, so one would prefer to not discard a large portion of the training samples due to scalability concerns.

The Subspace Forest is built without supervision, but it is not directly a method for clustering actions. It is, however, a structure for finding the closest neighbors to a given tracklet. As such, one might expect that there is a way to take advantage of the scalability of a Subspace Forest for clustering.

Chapter 7 describes such a method, called Latent Configuration Clustering (LCC). LCC uses a connectivity graph generated from the leaves of a Proximity Forest for clustering. It is called Latent Configuration Clustering because the configuration space of the data samples is unknown. LCC requires only a distance function which can be applied pair-wise between samples, and thus can efficiently cluster data in general metric spaces.

Chapter 7 describes the LCC algorithm in general as well as its application for unsupervised action recognition when coupled with a Subspace Forest. Using a Subspace Forest and the Grassmann manifold representation of actions, LCC is shown to have superior performance than the best known method of unsupervised action recognition on the KTH Actions data [NWF08]. In terms of scalability, LCC allows the clustering of a large corpus of tracklets in $O(N \log N)$ time complexity. As with many tree construction algorithms, under certain pathological conditions, the tree may degenerate to a list. Time complexity is reported under non-pathological conditions, which can be reasonably assured by randomly shuffling the order of the input data. LCC reduces the clustering time of the aforementioned 2,500 tracklets from 3 days to about 20 minutes.

1.5 Summary of Thesis Contributions

Necessity is the mother of invention. The DARPA Mind's Eye data set consisted of over three thousand sample videos from which to learn 48 verbs. Many of the target verbs are not what we consider actions, but instead require higher level representations that may be built using actions as a component. The distinction between the following terms: action, interaction, activity, behavior, and event, are discussed in Chapter 2. The action recognition component of our Mind's Eye system learned 66 unique action labels from the unlabeled training corpus (see Chapter 8 for additional discussion).

When faced with the challenge of learning actions from such a large data set, scalability improvements were a necessity. Without the accuracy and scalability advances of the methods described in this thesis, it would have been much harder to process the data required by the program. The action recognition techniques described herein were sufficient to recognize a subset of these verbs, and played an important role in the higher-level system that was created to recognize the rest.

Of even broader impact is the potential of the Proximity Forest to improve the accuracy of many computer vision algorithms that employ approximate nearest neighbor indexing. One popular application is large-scale image retrieval, where an image is presented to the system, and similar images from a huge data set are displayed to the user. Contemporary approaches to image retrieval [NS06, PCI+07] use ANN indexing structures which are shown in Chapter 5 to be inferior in accuracy to the Proximity Forest. The implication is that a black-box replacement of the ANN index may be possible, and as a result, that similarity queries may be 15 to 20% more accurate. The Proximity Forest has the same essential time complexity as existing ANN methods, but the current implementation will require optimization before a fair speed comparison can be performed. No claim is made that the current python-based serial implementation is anywhere near as fast as the current multithreaded and vector-optimized ANN implementations.

The specific contributions from this dissertation relate to addressing the challenges of developing unsupervised learning methods for the scalable recognition of actions in streaming videos.

Contributions include the following:

- An evaluation of how well existing action representations support unsupervised learning. Results suggest that the PM representation is more amenable to action clustering than methods based on space-time feature sampling.
- A method of unsupervised learning and recognition of actions in streaming video.
- A novel structure called a Proximity Forest for Approximate Nearest Neighbor indexing of general metric spaces.
- An evaluation showing superior performance of the Proximity Forest to popular ANN methods used in contemporary computer vision applications. This suggests that existing methods for large-scale image retrieval and nearest-neighbor classification could be improved by a simple substitution of a Proximity Forest for existing ANN indexing methods.
- A structure based on the Proximity Forest, called a Subspace Forest, for ANN indexing of actions represented as points on Grassmann manifolds, similar to the PM representation. The Subspace Forest provides highly accurate action classification on popular benchmarks while also allowing for real-time recognition.
- A clustering method called Latent Configuration Clustering (LCC) that can be applied to general metric data. The power of LCC is in transforming the clustering problem from using expensive pair-wise distance computations into clustering a sparsely connected graph by comparing scalar edge weights.
- When combined with a Subspace Forest, LCC can be used for clustering actions. LCC outperforms existing published methods of unsupervised action recognition.

Chapter 2

Background

This chapter provides background information on terminology, the limitations of existing forced-choice benchmarks, an overview of existing approaches to action recognition, and a description of the data sets used for evaluation purposes in this thesis. Additional emphasis is placed on the Product Manifold representation [LBK10], which is the model for manifold-based representations used extensively in this thesis.

2.1 Behavior Terminology

This section provides the definition of common terms used in the computer vision community relating to the general topic of human behavior recognition in video. We define our usage of the terms: action, interaction, gesture, activity, event, and behavior. Usage of these terms is not always consistent in the literature, so our intended meanings are provided below.

For the purposes of this thesis, an *action* is a characteristic motion that can be recognized with a second or two of observation of a single subject. Examples include bending, walking, jumping, and waving. An *interaction* is an action between two or more people, such as when two people hug, or a person gives an object to another. As with actions, we consider interactions to be short in duration, lasting a few seconds. A *gesture* is a characteristic motion of a body part, often the hand, considered in isolation from the rest of the body. An *activity* is longer in duration than an action or interaction, and may require a context for effective recognition. Examples include playing soccer, jogging, and loitering near a secured door. Note that the jogging activity can be recognized by the repeated observation of the jogging action over a relatively long duration of time, whereas playing soccer is more complex. Observing a single person kicking a ball is not sufficient. An *event* in video is a time when something important happens, such as when a person tailgates through a secured door or a goal is scored in the soccer match. Finally, the term *behavior* is used to generically represent actions, interactions, activities, and events.

Actions and interactions may be considered somewhat atomic, and can be used as building blocks to the recognition of higher level behaviors. Activities may include events, and events may include activities. For example, the activity of playing soccer involves actions (running, jumping, kicking), interactions (passing, slide tackling), and events (offsides, goals). An "unauthorized access" event may require the recognition of the loitering activity, followed by the loiterer tailgating another person through a secured door. Thus, the relationship between gestures, actions, interactions, activities, and events is not a simple taxonomy, and the divisions between these terms may be subject to interpretation.

A recent review on human activity recognition by Aggarwal and Ryoo [AR11] provides an overview of contemporary approaches to behavior recognition in video, including gestures, actions, interactions, and higher-level group activities. Our definitions are similar to those used in Aggarwal and Ryoo, except that we use "behavior" as a general term, and Aggarwal uses "activity" for this same purpose. Aggarwal does not present an equivalent term for our definition of activity.

This thesis focuses on the application of methods to action recognition. Yet methods used for action recognition may also be applicable to interactions, gestures, and other visually distinctive motions of tracked entities/objects in video. Of the data sets employed for evaluation purposes in this thesis (described later), one is a gesture recognition data set, which helps illustrate that the representation we employ may be useful beyond the strict definition of action recognition.

Finding ways to advance the state-of-the-art in recognizing actions, interactions, and gestures, may lead to concomitant improvements in higher level human behavior characterization in video.

2.2 Forced-Choice Action Classification

There is a great deal of research relating to the recognition of human actions in video, yet much of the research to date has focused on the problem of classifying short video segments (a few seconds long) according to a small, fixed set of labels. The term *forced-choice* is used to differentiate this from a classification task where it is assumed that the recognizable actions are a small subset of the possible actions that a subject could exhibit. A forced-choice benchmark data set is one in which each sample shows exactly one action from a closed set of labels.

Many popular benchmark data sets [SLC04, GBS$^+$07, LJD09, KWC07] consist of spatially and temporally segmented video clips that show a single actor executing a single behavior. Examples of three representative data sets, showing actions, gestures, and interactions, are shown in Figure 2.1. While the forced-choice classification paradigm has led to notable performance gains on benchmark data, it leaves many questions unanswered regarding the larger challenge of detecting and recognizing human actions in less structured contexts and in continuous streams of input.

Pre-segmented video clip classification is popular because no significant effort is required to annotate the data for training or performance evaluation. Each sample video clip demonstrates only a single label, and there is no requirement for a method to localize where and when the action occurs (in other words, action *detection* is not required).

Bag of Features (BOF) methods are popular choices for benchmark action recognition. In a BOF representation, an entire sample video clip is represented as a histogram of features. Features are commonly derived from intensity gradients in small space-time regions sampled from the video, as in [SLC04, Lap05, DRCB05, KG10] and others. The limitation of BOF methods is that the feature representation, a simple histogram of the detected features, does little to preserve the structure of the video. The locations from whence the features were sampled are discarded. A normalized histogram of two people walking might be extremely similar to that of a single person walking. When it is known that the benchmark contains only a single subject performing a single action, the significant structural limitations of this approach does not negatively impact performance.

In fact, because BOF methods perform no localization of the action, they benefit when benchmarks have semantically meaningless but strongly co-varying background information. The UCF Sports data set [RAS08], for example, has actions like golfing, diving, and weight-lifting. One could predict golfing by the green grass, diving by the blue water, and weight-lifting by a common backdrop evident in all the samples. See Figure 2.2 for illustration. As pointed out by Pinto et al. [PDC09], a sufficiently powerful machine learning mechanism, when applied to localized feature representations, can yield results that are inflated due to artifacts in the data that co-vary with the class labels. While this phenomenon is not limited to BOF methods, they are particularly suscep-

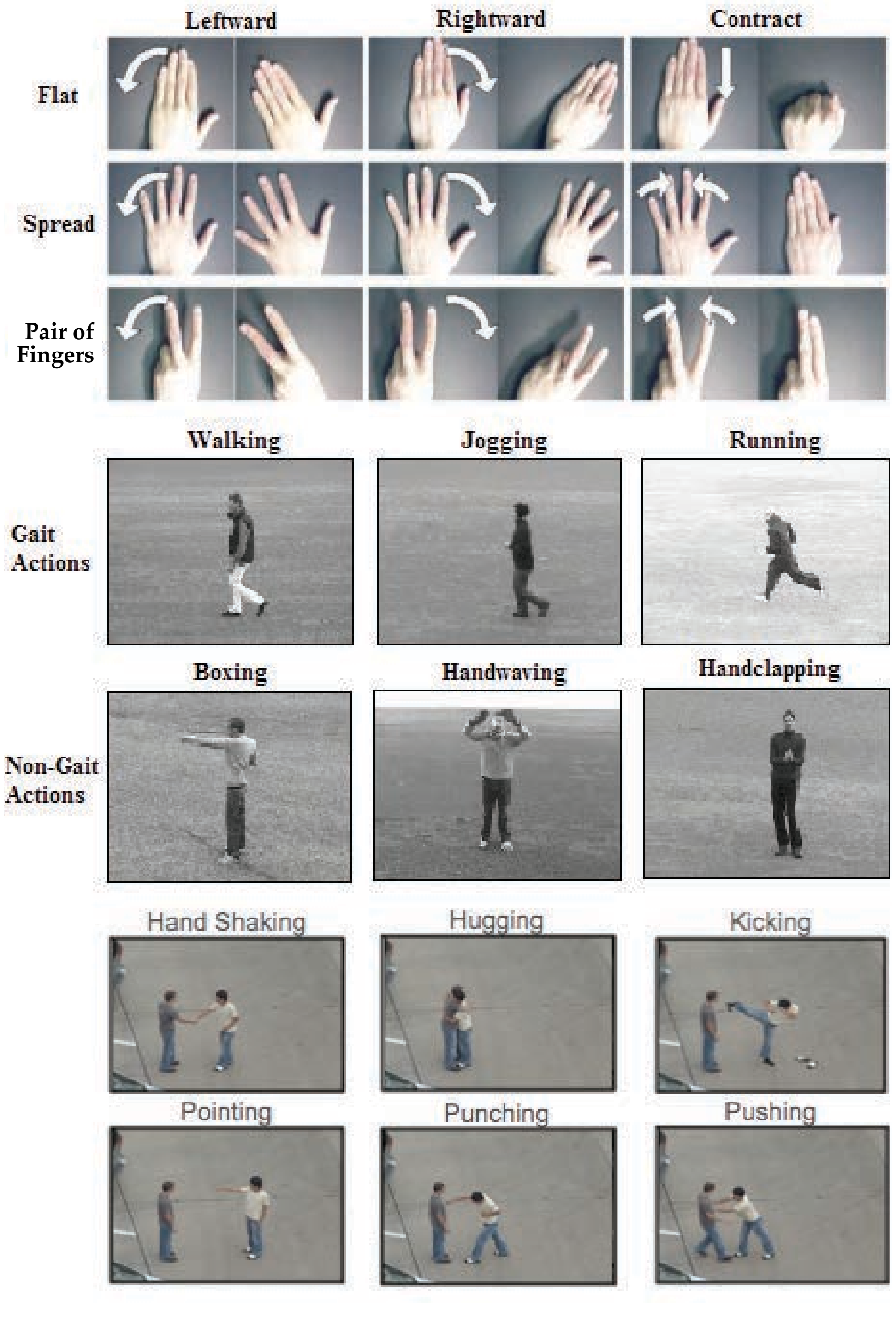


Figure 2.1: Examples of existing gesture, action, and interaction data sets. From top to bottom: Cambridge Gestures [KWC07], KTH Actions [SLC04], and UT Interactions [RCAR10] data sets.

tible because there typically is no attempt to control for sampling only features from the actor (or other region) of interest.

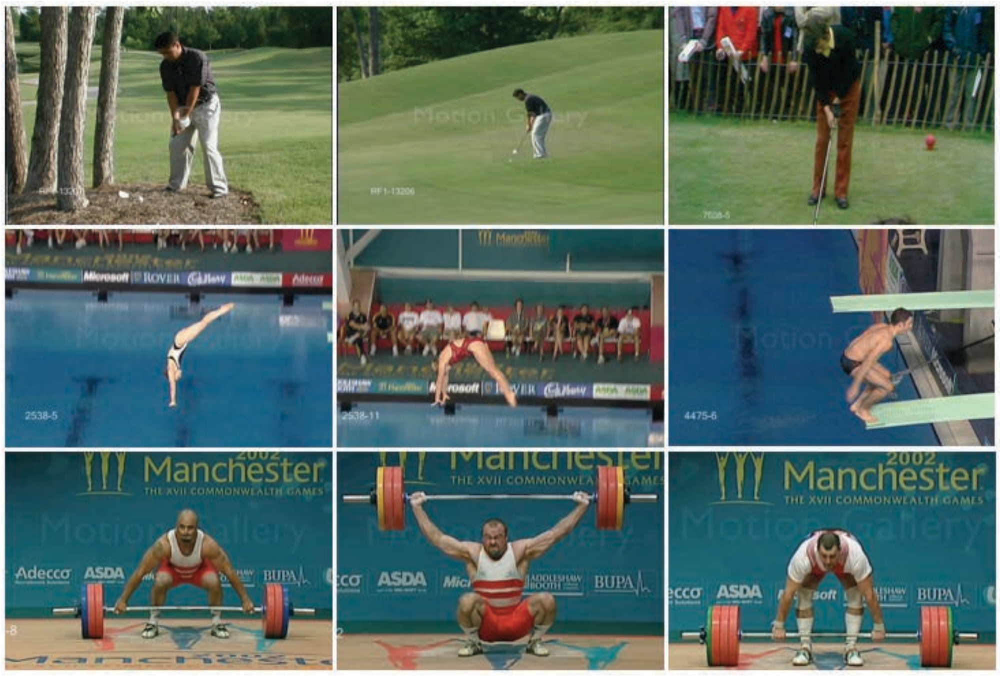


Figure 2.2: Samples from the UCF Sports data set [RAS08]. Many of the actions, such as golfing (top row), diving (middle row), and weight-lifting (bottom row), have strongly co-varying background features.

Action recognition is hard, and it is reasonable to attempt to simplify the problem using controlled data sets. However, in deference to the no-free-lunch theorem [Wol01], the techniques used to push performance to the highest levels on existing classification benchmarks may not yield the desired gains in addressing the more general challenges relating to action recognition from less controlled, streaming data sources. Recent results from the Contest on Semantic Description of Human Activities (SDHA Challenge) [RCAR10] indicate that current feature-based approaches perform well on pre-segmented video clip classification, yet fail to perform well in detecting and localizing actions in continuous videos.

The representations put forth in this thesis require that the subjects are first localized, and then action recognition is performed to describe what they are doing. In this way, successful recognition is attributed only to the pixel data that comprises the action, independent of context or background.

2.3 Existing Action Representations

Because the choice of representation can be critically important to the performance of pattern recognition algorithms, we discuss existing action recognition techniques based on their fundamental representational choices. Representations common in the literature include: localized space-time features (i.e., Bag of Features), silhouettes, body-part tracking, and 3D arrays (tensors) built from tracked regions in the video.

2.3.1 Bag of Features

The Bag of Features approach has become one of the most popular methods for human action recognition in short video clips [SLC04, DRCB05, KSH05, Lap05, GBS$^+$07, SAS07, NWF08, RAK09, KG10]. As adapted from similar methods of image classification and retrieval, BOF approaches represent video clips as unordered sets of local space-time features. Features are quantized into discrete vocabularies, or codebooks. The space-time features in a video are assigned to their nearest neighbors in the codebook. The BOF representation is a normalized histogram of frequency counts of the codebook entries detected in videos. Activity classification is often done by applying Support Vector Machines with appropriate kernels (χ^2 is common) to the BOF representation.

There are many choices involved when implementing a BOF approach. One must decide how to sample the video to extract localized features. Possible sampling strategies include interest point operators [MTS$^+$05, Lap05, DRCB05], grids/pyramids [RAK09, KG10], or random sampling [JT05]. Each strategy comes with parameters including space and temporal scales, overlap, and other settings. From the sampled regions, an appropriate descriptor must be chosen [DRCB05, SAS07, LP07] to provide a balance between discrimination, robustness to small photometric and geometric perturbations, and compactness of representation. Wang et al. provide an evaluation of popular space-time interest point detectors and features [WUK$^+$09], yet there is no conclusive result indicating which combination of detector and descriptor is best. The results are data-set dependent. Beyond feature detection and extraction, other design choices include codebook size, quantization method, and distance function to be used in nearest-neighbor assignments.

These and other considerations of the Bag of Features method are discussed in the context of image classification and retrieval in [OD11a].

An attractive aspect of the BOF approach is the lack of any requirement to pre-process the videos to perform segmentation or track moving objects. This comes, however, with a significant difficulty in knowing precisely why two videos are considered similar, as there is little semantic meaning in the representation. As stated earlier, it is possible to correctly classify videos due to co-varying, but semantically irrelevant, background artifacts in the data set.

In terms of scalability and recognition speed, there have been efforts to improve the computational efficiency of bag-of-features action recognition using hierarchical structures. Taking cues from the vocabulary trees used in object recognition and image retrieval (e.g., [MTJ06]), trees can be effectively applied to codebooks of spatio-temporal features. Yu et al. [YKC10] propose the use of a forest of randomized trees to build a feature codebook directly from the pixels surrounding space-time interest points. Yu reports recognition speeds from 10 to 20 frames per second, and vocabulary generation time is 500 times faster than K-Means (no absolute training time is reported.) Yao et al. [YGV10] use randomized forests to map densely-sampled features into a Hough voting space. Unlike Yu et al., Yao's use of a randomized forest is not for speeding up codebook construction, but rather as a core action-recognition mechanism that uses features to vote for the most likely class label. Gilbert et al. [GIB09] use a hierarchical structure inspired by data mining techniques to perform rapid action recognition using localized features. Gilbert reports recognition speeds of 24 frames per second on KTH data, and training throughput of a multi-stage process ranging from 21 to 640 frames per second per stage, although it is unclear what total training time is required.

2.3.2 Silhouettes and Parts

Departing from the BOF works are silhouette motion methods such as those from Lin et al. and Nater et al. [LJD09, NGV10]. Lin et al. employ joint likelihood maximization between the current observation and learned shape-motion prototypes. Nater et al. presents an unsupervised method for learning human behaviors by clustering silhouettes and motion patterns.

The main challenge with silhouette approaches is the sensitivity to silhouette segmentation, which is an open challenge in non-trivial settings with complex and dynamic backgrounds. For real-world usage, one must solve the segmentation problem to yield the silhouette before a silhouette representation for action recognition can be applied. Segmentation is a different and more difficult challenge than simple bounding-box localization.

Similar to silhouette methods, which model how silhouettes change over time, is the idea of tracking body parts for action recognition via changes in pose. The segmentation challenge is similar in both cases. Accurately localizing body parts such as the upper arm, lower arm, waist, thigh, calf, foot, hand, etc., is a significant challenge with many points of failure. Recent publications on pose detection suggest that even the best methods are not yet reliable enough for action recognition in monocular video. Andriluka et al. [ARS09] present a leading human pose detection algorithm with accuracy around 55% correct part localization. While it may be possible to improve part localization performance by integrating the temporal information in video data, we know of no competitive published results on benchmark data sets using a body-part localization representation.

We do not further investigate silhouette and body part/pose representations due to the segmentation challenge when applied to monocular video. However it is worth noting that in some domains, such as with a sensor that provides depth information like the Microsoft Kinect, these methods can work well because segmentation becomes much easier [SFC$^+$11].

2.3.3 Tensors

A tensor is a mathematical formalism for an n-dimensional array. For action recognition, we can construct tensors from the (x,y,t) coordinates of every pixel in a video segment, where the time dimension, t, is represented by a frame number. The 3D array of pixel data (i.e., a 3-mode tensor) may also be referred to as a data cube.

Tensor representations are pixel-based representations. They may be used for action recognition using 3D correlation [RAS08, SC12], or by performing some sort of subspace analysis after factoring the tensor into a set of matrices [KWC07, LBK10]. An advantage of tensor representations is that the majority of pixels involved in the action is used, unlike feature-based methods

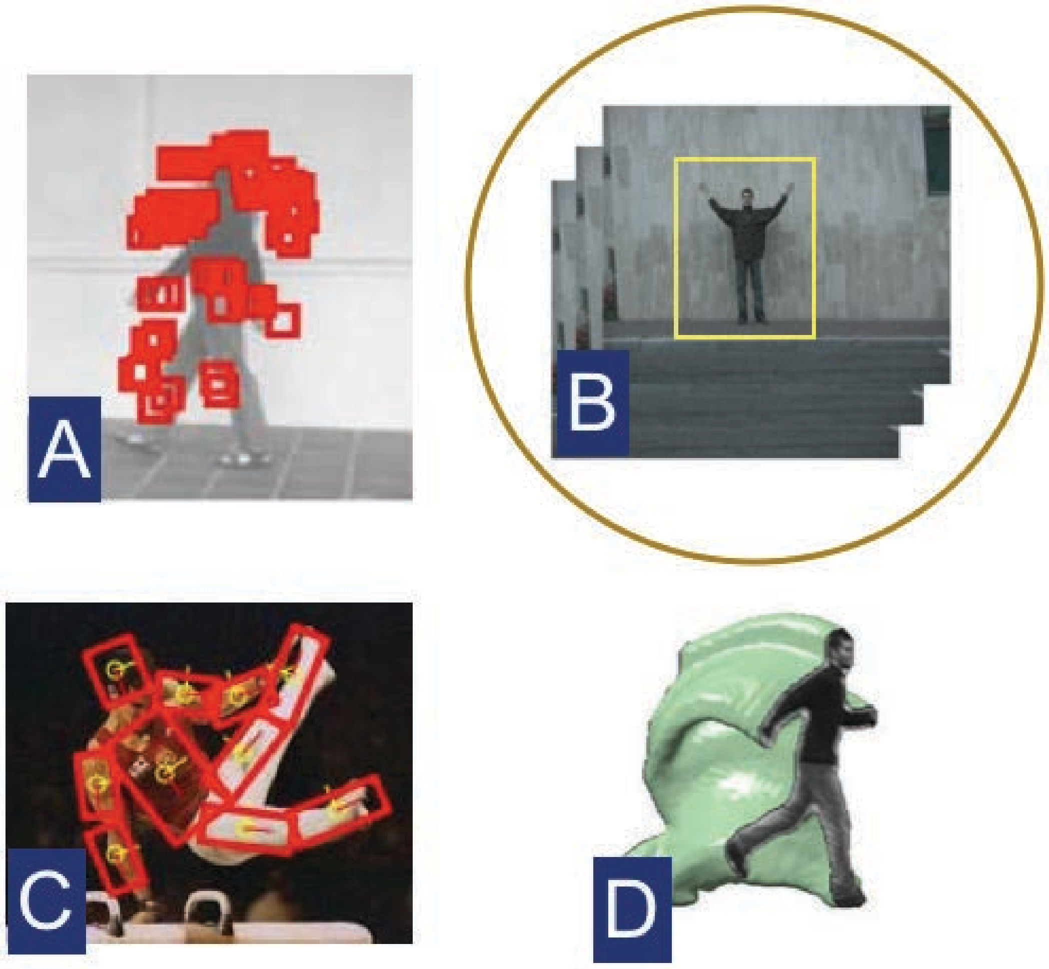


Figure 2.3: Four approaches to representing actions. A) Space-time interest points, illustration from Niebles et al. [NWF08]. B) Tracked region, as used in this thesis. C) Parts-based pose representation, illustration from Andriluka et al. [ARS09]. D) Silhouette tunnel, illustration from Gorelick et al. [GBS$^+$07].

which sample small space-time patches, and thereby discard the majority of pixels in the process. A challenge in keeping all the pixels is the high dimensional nature of the representation.

One approach to addressing the high-dimensional nature of the representation is to map the tensor into a lower dimensional space with some regular structure. This is the approach taken the PM representation, which maps tensors into a product of Grassmann manifolds (as defined in the following section). The general idea is that video segments are mapped to points on a manifold structure, and distances between video segments are measured using a geodesic distance defined on the manifold. Assuming the geodesic distance can be efficiently computed or approximated, it can be used to classify or cluster the corresponding videos.

Although it may be difficult to conceptualize the mapping of tensor data to a manifold surface, the computation is fairly straight forward, requiring techniques from basic linear algebra, as we show below. The advantages are that the representation requires many fewer design choices and parameter settings in comparison to space-time feature sampling methods, requires only bounding-box style localization in comparison to silhouette or parts-based representations, and does not require additional temporal sequence modeling because the dynamics are captured directly from the pixel information. This representation has been shown to be highly accurate in a nearest neighbor action classification. For these reasons, and others that we demonstrate in Chapter 3, the mapping of actions (as data cubes) to manifolds is our representational choice for the methods described in this thesis.

2.4 Product Manifold Representation

In this section, we provide an overview of the PM representation [LBK10] because it is referenced throughout this thesis. The high-level idea is that a short video segment can be represented as a point on a non-Euclidean manifold surface. By measuring the distance between points on the manifold, one can determine the similarity of two cropped video segments. Action classification is performed by cropping a salient region of an input video and finding the nearest neighbor on the manifold.

Figure 2.4 illustrates the process of computing the PM distance between a pair of video segments. We use the term *tracklet* to denote a sequence of cropped images of a subject tracked over a few seconds time (see sub-figure B of Figure 2.3). A tracklet is represented as a data cube with axes of width, height, and frame number, (x, y, f). All tracklets are scaled to be the same size. Tracklets tensors can be flattened into matrices by unfolding along any of the three axes.

Let T_i represent tracklet i. Let T_i^k be the matrix representing the tensor unfolding of T_i along axis k. Then, using QR Factorization, we have the following.

$$T_i^k = Q_i^k R_i^k \tag{2.1}$$

Q_i^k is an orthogonal basis for T_i^k. The distance between two tracklets, T_1 and T_2, can be computed with respect to unfolding k, using the principal angles between the subspaces spanned by Q_1^k and Q_2^k.

$$\begin{aligned} {Q_1^k}^T Q_2^k &= U \Sigma V^T \\ \cos\Theta &= diag(\Sigma) \end{aligned} \tag{2.2}$$

In Eq. (2.2), the Singular Value Decomposition is used to compute the principal angles between the subspaces. The singular values, given by the diagonal entries of Σ, are the cosines of the principal angles.

A Grassmann Manifold, $Gr(r, n)$, is the set of all r-dimensional linear subspaces of the n-dimensional vector space V. Points on a Grassmann Manifold are subspaces, and assuming the underlying field is $\Re$, can be identified by orthogonal matrices. By computing the orthogonal decomposition of the flattened matrix, one can identify the point on the Grassmann associated with the flattened data cube. In this manner, we map tracklets to points on Grassmann Manifolds, each tracklet being represented by the three subspaces arising from the three tensor unfoldings.

The chordal distance can be used to represent the distance between points on a Grassmann Manifold. This is a pair-wise comparison between two tracklets using the principal angles between the subspaces. The chordal distance, $d(T_1^k, T_2^k)$ between tracklets T_1 and T_2 along axis k is the L_2 norm of the component-wise sine function applied to the entries of Θ.

$$d(T_1^k, T_2^k) = \|\sin\Theta\|_2 \tag{2.3}$$

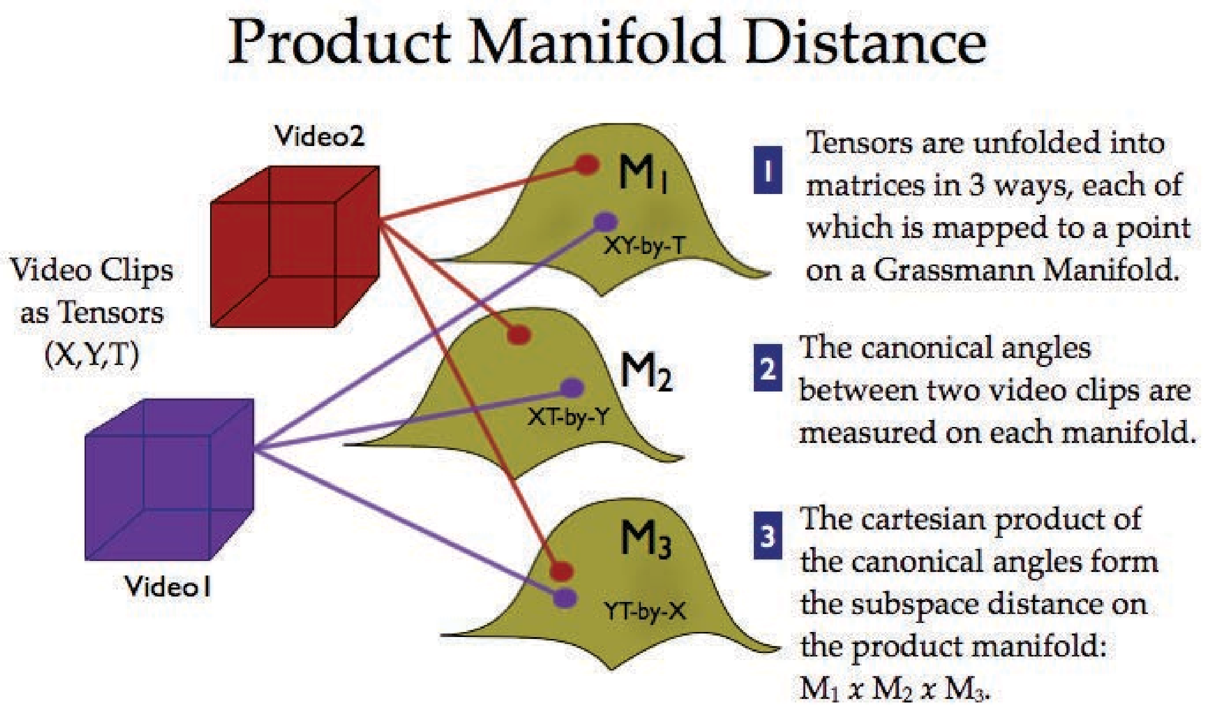

Product Manifold Distance
Video2
Video Clips as Tensors (X,Y,T)
Video1
M1
XY-by-T
M2
XT-by-Y
M3
YT-by-X
1 Tensors are unfolded into matrices in 3 ways, each of which is mapped to a point on a Grassmann Manifold.
2 The canonical angles between two video clips are measured on each manifold.
3 The cartesian product of the canonical angles form the subspace distance on the product manifold: M1 x M2 x M3.

Figure 2.4: Illustration of the Product Manifold Distance, adapted from [OLD12].

There exists a product manifold which is the product of the three Grassmann manifolds. Each video is a point in the product manifold structure. The distance between video clips is the distance on the product manifold, which can be computed using the Cartesian product of the canonical angles between the points on the factor manifolds. The chordal distance on the Cartesian product of the three sets of canonical angles is the PM Distance.

The advantages of the Product Manifold approach include the relatively small number of design choices, the lack of any training or codebook generation process, and its performance on nearest-neighbor action classification. The disadvantage of this method is the requirement to use fixed-size cubes in the representation. The video clips from the data sets must be cropped or scaled to a uniform-sized cube, which we discuss how to do in later chapters.

One significant challenge left unaddressed by the Product Manifold representation is scalability. A single comparison between two samples can take approximately 100 ms. While a linear search to find the nearest neighbor is acceptable for small scale applications, it will not allow for real-time recognition when using a large data set.

Other challenges arise due to the non-Euclidean nature of the representation. For example, computing the mean of a set of samples (for use in K-means clustering, e.g.) has no closed form solution, and even the definition of a mean can be unclear. To understand why, imagine two points on opposite ends of a sphere. Constrained to the surface of a sphere, which is a simple manifold, there is no unique shortest path, and thus preferred direction along which to sample the mean. Would the mean location of the two poles of a globe be the set of points along the equator? Which point should you choose when computing the distance from the "mean" to some other point on the surface?

For this and other reasons explored in later chapters, efficiently clustering points on Grassmann manifolds is not as simple as with vector data. The Product Manifold distance is a metric, however, and thus clustering can be achieved by directly computing all-pairs distances to generate a distance matrix, which can in turn be used by a number of common clustering methods such as hierarchical agglomerative clustering. However with 100ms per comparison, the $O(N^2)$ comparisons required to construct the distance matrix becomes a limiting factor for large data sets.

A significant contribution of this thesis is in addressing the scalability concerns relating to both clustering and nearest neighbor selection when actions are represented as points on Grassmann manifolds.

2.5 Overview of Data Sets

Several data sets are referenced throughout this dissertation. This section provides a consolidated overview of the key aspects of each, which are further summarized in Table 2.1. Two of the data sets, Cambridge Gestures and Facial Expressions, do not represent full-body actions, which is the main application focus of this dissertation. However, it is informative to include different types of human subject-based video data to illustrate that the representations and methods presented in this dissertation could be applied to a broader range of potential applications.

Of note, we use the terms "subject" and "actor" synonymously to indicate the person of interest in full-body action videos. However, the term "subject" can also refer to a close-up of a hand (for gestures) or face (for expressions).

2.5.1 KTH Actions

KTH Actions [SLC04] (KTH) presents full-body actions. KTH has been used extensively in the action recognition literature. At the time of this writing, it has been cited nearly 1,000 times according to Google Scholar.

The KTH data set consists of six classes (walking, jogging, running, boxing, handwaving, handclapping), demonstrated by 25 subjects, each in four different scenes. See the middle panels of Figure 2.1. Of note, one of the 600 variations is missing, so there are actually 599 samples provided in the data. The first three scenes are taken outdoors, with a fairly uniform background. The fourth scene is taken indoors, also with a uniform background. Scene 2 varies the scale or angle from Scene 1. Scene 3 varies the clothing of the subject.

Three of the classes (walking, jogging, running) involve a human gait, while the other three involve actions where the actor remains in place (boxing, handwaving, and handclapping). We call the latter three the "stationary" actions in KTH, although clearly the actions involve motion – stationary meaning the actor stays in the same place. The actor varies direction of travel (for the gait classes), and is not always well-centered in the stationary actions.

2.5.2 Cambridge Gestures

Cambridge Gestures [KWC07] (Gestures) is a popular gesture data set, cited over 100 times since 2007. The data set consists of nine classes, repeated in five sets of varying lighting, with 20 samples per class per set, for a total of 900 videos. Each sample is a close-up of a single hand on a uniform background performing one gesture. The nine classes are divided into three hand shapes (flat, spread, pair-of-fingers) combined with three motions (leftward, rightward, contracting), as illustrated at the top of Figure 2.1.

2.5.3 Facial Expressions

Facial Expressions [DRCB05] (Expressions) has short video clips of a subject exhibiting one of six different emotions. The Facial Expression data set differs from the others in that the motions involved are deformations of the facial features and have less translational motion of body parts.

The Expressions data consists of six classes (anger, disgust, fear, joy, sadness, surprise), repeated in four sets. The four sets are comprised of two subjects under two different lighting conditions performing eight repetitions of all expressions, for a total of 192 videos. Each video starts with the subject in a neutral expression, then transitions into one of the expressions, and then back to neutral.

2.5.4 ETHZ Living Room

The ETHZ Living Room data set [NGV10] (ETHZ) consists of videos of a single person moving about a living room set performing various actions such as sitting down, bending over, walking around the room, and so on. This data set differs from the others in that it does not contain pre-segmented video samples and does not strictly pre-define the set of possible actions.

We selected this data set for use in Chapter 4 because it represents the continuous surveillance problem better than many of the more popular action recognition benchmarks. The ETHZ data set provides three video sequences. The first, over 7,000 frames long, is a continuous recording of a person moving about a room and performing a few selected behaviors (walking, sitting, bending down). The second two videos are similar, but shorter, and contain novel actions not observed in the first video, such as falling down, jumping, and panicking.

2.5.5 UCF Sports

UCF Sports is a relatively small data set, with 150 video clips representing ten actions: diving, swinging a golf club, kicking, weight lifting, horseback riding, running, skateboarding, swinging on a pommel horse, swinging on high bars, and walking. UCF Sports is challenging due to unequal set sizes, significant intra-class variability, and complex backgrounds.

2.5.6 UT Tower

UT Tower [CRA10] (Tower) presents a set of full-body human actions captured from a camera perched atop a bell tower. This set features low-resolution data with few pixels on target. Compared to KTH, the Tower data set has fewer samples but more classes. Specifically, UT Tower has 108 samples of nine classes: standing, pointing, 2-handed waving, 1-handed waving, jumping, digging, walking, carrying, and running.

Table 2.1: Overview of Data Sets. This table summarizes key characteristics of the data sets used for evaluations in this dissertation. *Short Name* is the abbreviation used to refer to the set, *Classes* indicates the number of classes in the data, *# per Class* indicates the number of samples per class, and *Sample Length* is the approximate duration, "seconds" or "minutes", describing how long each sample video clip lasts.

Name	**Short Name**	**Classes**	**# per Class**	**Sample Length**
KTH Actions	KTH	6	100	Seconds
Cambridge Gestures	Gestures	9	100	Seconds
Facial Expressions	Expressions	6	32	Seconds
ETHZ Living Room	ETHZ	N/A	N/A	Minutes
UCF Sports	UCF Sports	10	Varies	Seconds
UT Tower	Tower	9	12	Seconds

Chapter 3
Action Clustering

Our first step towards advancing unsupervised action learning from videos is to select an appropriate representation of the data. Without supervision, grouping of videos will be determined by the biases of the representation and associated distance metric. This chapter investigates the differences in clustering on three data sets between a standard Bag of Features (BOF) representation based on localized spatio-temporal feature sampling and the Product Manifold (PM) representation. We undertake this evaluation with the goal of gaining insight into how these very different representations affect cluster purity.[1]

For this evaluation, we use the Expressions, Gestures, and KTH data sets described in Chapter 2. We consider each to be a controlled data set because there is only a single subject performing a single behavior with a nearly uniform background. Each of these data sets was designed for a forced-choice classification task, so each has a small set of defined class labels. In addition to the class labels provided by the data set, which we call the "nominal" labels, we apply other labels to the data that correspond with various aspects of similarity, such as direction of motion. Clustering is evaluated in terms of cluster label purity based on each set of labels. We are primarily interested in the cluster purity of the nominal labels, but by using other labels, such as direction of motion or subject identity, we can learn something about why the two methods perform differently.

We show that PM yields superior results to BOF when measuring the alignment between the generated clusters and the nominal class labeling of Gestures and KTH, and similar results on Expressions. We find that gross motions were easily clustered by both methods, but that the lack of structural information inherent to the BOF representation leads to limitations that are not easily overcome without supervised training.

[1]Material presented in this chapter was previously published by the author in [OLD11, OLD12].

3.1 Method for Comparing BOF and PM Representations

Our method for comparing the two representations is to generate a pair-wise distance matrix using both representations from the training samples of a given data set. We apply a well-known hierarchical agglomerative clustering routine to the distance matrices to produce dendrograms (tree structures) of the similarity between the samples. Dendrograms can be cut at varying levels in their hierarchy to produce different numbers of clusters, from coarser to finer-grained grouping. We vary the number of clusters, K, over a range of values and observe how well the unsupervised grouping of the video samples compares to the selected labels. While we use labels to *evaluate* the clustering, the formation of the distance matrices and subsequent hierarchical clustering is entirely unsupervised. More details of each of these aspects can be found below.

We selected Piotr Dollár's BOF implementation [DRCB05], popularly known as the "Cuboids" algorithm, because the well-documented code is readily available upon request from the author, and generates competitive classification results in an independent study of BOF feature detectors and descriptors for action recognition [WUK$^+$09]. We used Yui Man Lui's MATLAB implementation of the Product Manifold algorithm because it has been shown to provide strong results when used for nearest-neighbor classification of actions and gestures [LBK10].

3.1.1 Data Sets

We selected three data sets for this study: KTH Actions, Cambridge Gestures, and Facial Expressions. The data sets are described in Chapter 2, and illustrated in Figure 3.1. In addition to full-body actions from KTH, we include hand gestures and facial expressions so that we can better understand how our conclusions from this study apply to a slightly broader scope of application data.

All three data sets were designed to evaluate forced-choice classification algorithms. For the sake of familiarity within the action and gesture recognition community, we elected to use these same data sets, but in an evaluation scheme that measures unsupervised clustering and how the clusters align with different potential labelings of the data (direction of motion, subject identity, and others as later enumerated). Video samples may be similar along different aspects than the

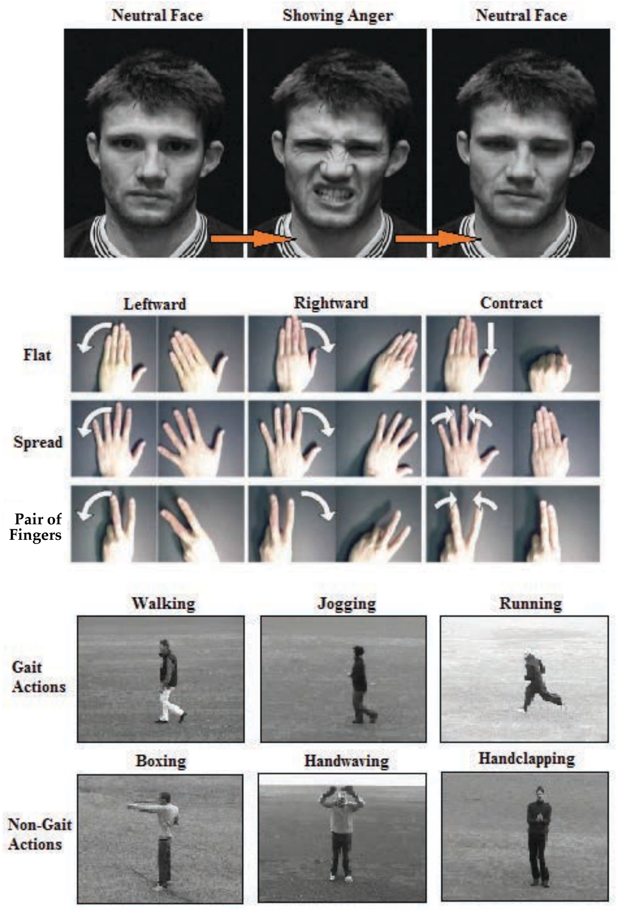


Figure 3.1: Data sets used in this evaluation. From top to bottom: Expressions [DRCB05], Gestures [KWC07], and KTH [SLC04].

externally applied class label, and our evaluation helps illustrate which aspects the two representations are sensitive to.

3.1.2 Bag of Features

When employing BOF, two key design choices are the selection of the feature detector, used to determine where to sample features, and the feature descriptor, used to represent a sampled feature. The choice of detector and descriptor is data-set dependent, and is also sensitive to parameters such as the spatial and temporal scales (denoted as σ and τ, respectively in this chapter) used in detection and feature description.

For the Expressions data, we used the settings provided in Dollár's source code for application to the Expressions data, which he also created. The code employs the Cuboids detector (separable linear filters, as described in [DRCB05]) coupled with the Cuboids descriptor, which is a flattened vector of gradients reduced via PCA to 100 dimensions.

For the Gestures and KTH data sets, we employ the Cuboids detector coupled with Histogram of Oriented Flow (HoF) features. HoF features were computed using Dollár's source code. We found this combination to yield superior results when applied to the Gestures and KTH data sets compared to what was used for Expressions. This combination was also shown to generate good classification accuracy on KTH Actions by Wang et al.'s evaluation of space-time features [WUK$^+$09]. The HoF descriptor has 440 dimensions, which we employ with no dimensionality reduction because we found applying PCA to this descriptor reduced performance. For the Cuboids detector, we set the spatial scale $\sigma = 2$ and the temporal scale $\tau = 4$ for KTH, which are the same settings in Wang et al.'s evaluation. For Gestures, we use $\sigma = 2$ and $\tau = 3$, as a result of a parameter selection search to optimize performance.

We use a vocabulary of size 150 for all evaluations, selected empirically as having the best performance among sizes ranging from 50 to 1000. Each data set requires separate vocabulary creation using the features extracted from training data. For each, the vocabulary was generated by K-Means (where K is the vocabulary size) over a random sample of 10% of all the features extracted from the data set. We chose to use 10% due to scalability issues in generating the vocabulary from the features. Due to the randomness inherent in the vocabulary creation, we repeated the process 20 times and chose the vocabulary that generated the best results.

The BOF representation was formed for each video by sampling features and mapping each to the closest entry in the vocabulary (also called a codebook). The BOF representation for a video is the normalized histogram of frequency counts of the vocabulary terms. In other words, with a vocabulary of size 150, each video becomes a 150-element histogram, called the feature vector. A distance matrix was created by computing the distances between all pairs of feature vectors in a data set. We used the χ^2 histogram distance function, which is a common choice in the BOF literature. The distance function is shown in Eq. 3.1, where i is the bin index of the histogram.

$$d(H_1, H_2) = \sum_i \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i)} \tag{3.1}$$

For the remainder of this chapter, this approach will be labeled “BOF.”

3.1.3 Product Manifold

We used the code provided by Lui with no modifications beyond those required to generate pair-wise distance matrices on different data sets. Tracklets are extracted from the samples in the data, as discussed below. The tracklet tensors are mapped onto the product manifold, as presented in Chapter 2, and then the pair-wise distances are computed. For the remainder of this chapter, this approach will be labeled “PM.”

For all three data sets, we extract tracklets having 32 frames where each frame is 20x20 pixels in resolution. For Gestures and Expressions, the videos provided by the respective data sets are already cropped spatially and temporally. Extracting a tracklet from either of these two data sets is done by selecting the 32 frames temporally centered on the sample, and down-sampling each frame using bilinear interpolation to be 20x20 pixels. To extract tracklets from KTH, we select a fixed position space-time bounding box, based on annotations provided by Lui, available from him upon request. As with Gestures and Expressions, we down-sample the spatial extents to 20x20 pixels and use the middle 32 frames from each sample. However, some actions do not have 32 consecutive frames of the subject on-screen. In that case, we use a smaller temporal window and repeat frames as required to fill 32 frames.

3.1.4 Cluster Purity

We define *cluster purity* as the fraction of tracklets that were of the majority label in their respective clusters. The minimum score always occurs when $K = 1$, in which case the cluster purity is the ratio of the number of tracklets in the largest class to the total number in the data set, N. At the other extreme, when $K = N$, the cluster purity will be 1.0, as all samples will be assigned unique clusters and thus there will be no cluster “impurity.” The computation is shown formally in Eq. 3.2, where C is the set of K clusters, x_i are the tracklets being clustered, and $|\cdot|$ indicates set cardinality.

$$
\begin{gathered}
C = \{C^1, C^2, \ldots, C^K\} \\
X_L^K = \{x_i | x_i \in C^K \wedge Label(x_i) = L\} \\
\text{Cluster Purity} = \frac{1}{N} \sum_{k=1}^{K} \max_{L \in Labels} |X_L^k|
\end{gathered}
\tag{3.2}
$$

3.1.5 Hierarchical Clustering

We use agglomerative hierarchical clustering [FHT09] to group similar video samples. Hierarchical clustering is used because it produces deterministic results and the number of clusters can be varied without re-computation. In a completely unsupervised learning environment, the number of class labels is unknown, so having the different levels of similarity/generalization provided by hierarchical clustering is appropriate.

The non-Euclidean PM representation is not easily clustered via K-means (or hierarchical K-means), because computing the mean of points on the manifold is relatively difficult. The Karcher Mean [Kar77] is one option for computing the mean of points on Grassmann manifolds, but this is an iterative method that is most accurate when the samples being clustered are somewhat close together. In certain cases, the computation may not converge. Instead, we selected a clustering method that could be applied to both representations in exactly the same way by accepting a pre-computed distance matrix as input.

Prior to performing the main evaluations presented in this Chapter, we performed a pilot study to determine which of several possible linkage functions to use with agglomerative hierarchical clustering. A linkage function computes the distance between two clusters and is used to determine which two clusters should be merged at each stage of the agglomerative process.

We evaluated the following linkage functions: Single [FHT09] (nearest neighbor between cluster members), Complete [FHT09] (furthest neighbor), Average [FHT09] (average distance), McQuitty's [McQ66] (weighted average based on recursive agglomerations), and Ward's [War63] (minimum incremental increase in inter-cluster variance).

Figure 3.2 shows a comparison of the linkage methods when employing the BOF representation for measuring the similarity of Gestures. In this figure, we plot the cluster purity of the Gesture class label against the number of clusters, K, which was varied from 1 to 30. We perform a single

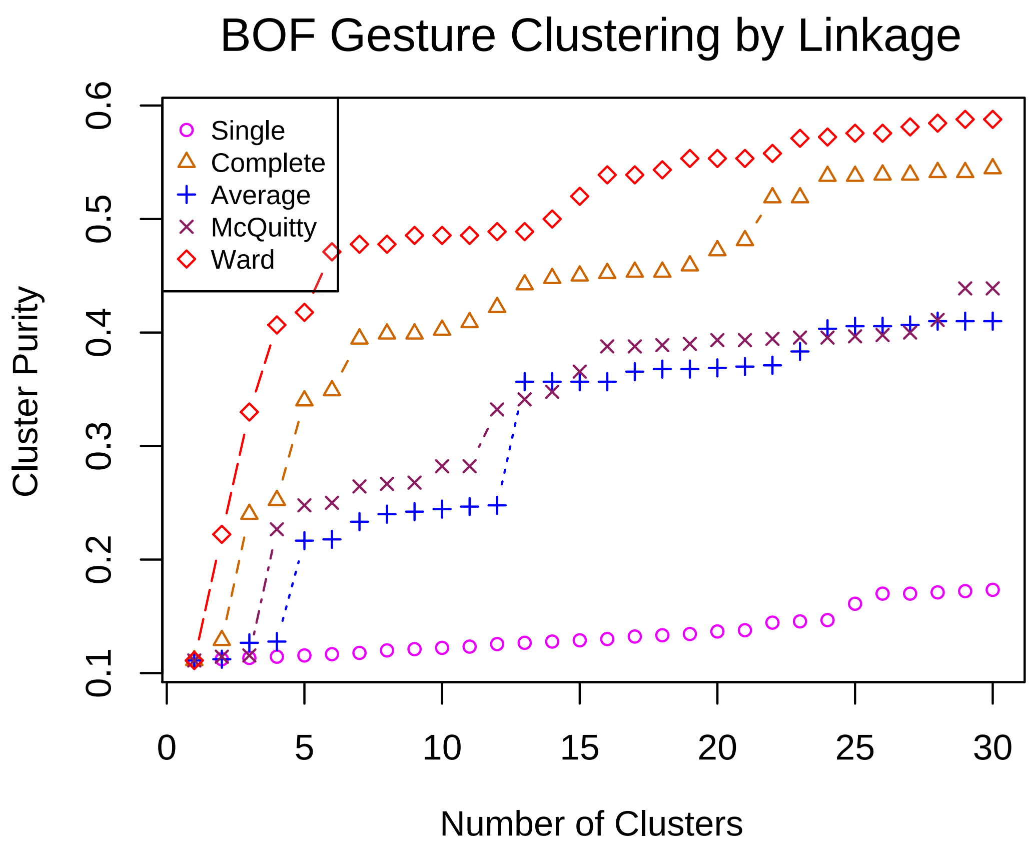


Figure 3.2: Comparison of hierarchical clustering linkage methods. This graph is formed using the Bag of Features method on the Gestures data set. This graph is representative of the linkage performance over all data sets in that Ward's linkage was superior in all cases.

hierarchical clustering per curve, and cut the resulting dendrogram at the appropriate level to yield K clusters. When the ideal number of clusters is unknown, the full curve may be more indicative than any single point in measuring performance.

We performed this comparison for both BOF and PM over all three data sets and determined that Ward's linkage generated superior results in all cases. The conclusion of this pilot study was the use of Ward's linkage for the subsequent analysis described in this chapter.

3.2 Results

We compared BOF and PM for clustering Expressions, Gestures, and KTH actions. Each evaluation is described below. A summary comparison of the relative performance of BOF and PM is illustrated in Figure 3.3. This figure presents the performance curve when the generated

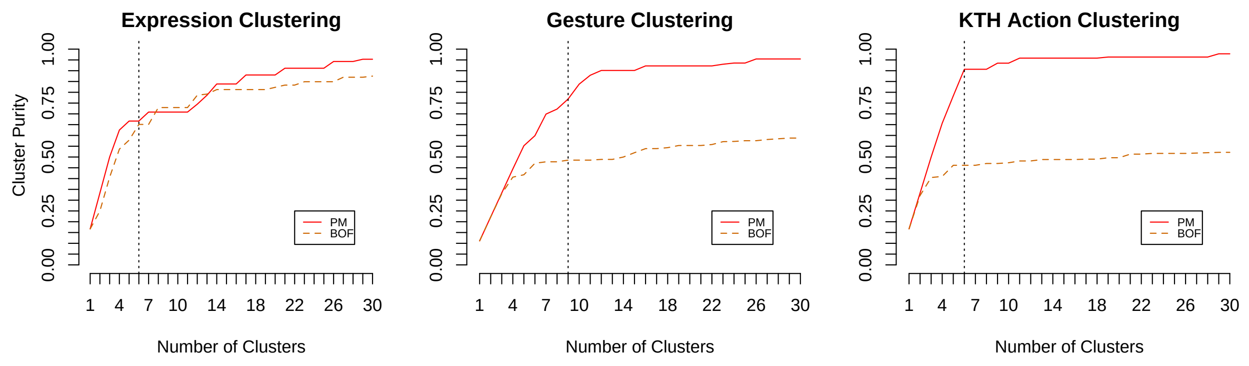


Figure 3.3: PM versus BOF methods compared against the nominal class labels on all three data sets. Vertical line indicates number of nominal classes in the data set (6,9,6, respectively). PM cluster purity on KTH Actions is 90.7% at K=6.

clusters are compared against the nominal class labels provided by the data set. There are 6 classes in KTH Actions and Expressions, and 9 classes for Gestures, indicated by the vertical dotted black line. The solid red curve shows the cluster purity of PM over all K, the amber dotted curve shows BOF.

From this figure, PM strongly outperforms BOF on the KTH Actions and Gestures data, while yielding comparable results on Expressions. This may be in part because Dollár developed both the Expressions data set and the BOF implementation we adapted for this study, and thus the implementation may have a level of tuning for this data set not present in the others. However, we believe other factors are involved, which we present later.

A key result shown in Figure 3.3 is the performance of PM on KTH Actions. At $K = 6$, the cluster purity is 90.7%, suggesting that the KTH data set is intrinsically separable along the class labels using PM, and that one could *discover* the classes if they were not known a-priori. In [LBK10], Lui et al. report nearest-neighbor classification results on KTH Actions using the PM representation scoring 96-97%, depending on the classification protocol (fixed partitioning of the data versus leave-one-out). Other features-based approaches also have classification accuracies on KTH in the mid-to-upper 90's, but they employ strong supervised classifiers. For example Kovashka and Grauman [KG10] scores 94.5% using hierarchical features and multiple kernel learning.

As with KTH Actions, the cluster purity on Gestures shows a big gap between PM and BOF.

We show in more detailed analysis, below, that the ability for BOF and PM to discriminate hand shape is a limiting factor in cluster purity. Since PM is better able to distinguish between hand shapes undergoing the same motion, it produces purer clusters.

In our work we desire representations for unsupervised learning, so the difference between the PM representation and the BOF representation, in terms of how much supervision is required to separate the data along the desired partitions, is important. Our results suggest that BOF requires more supervision for high classification accuracy because clustering alone does not effectively separate the data. We explore these and other aspects in more detail, presented according to data set, below.

For each data set, we explore the purity in comparison to alternative labelings, such as subject, direction of motion, or lighting. We select the alternative labels based on the information available about the data set. For example, there are 25 subjects in the KTH data and the data is provided in such a way that this label is easy to determine, thus subject identity is one alternative labeling we employ for KTH. The reason we explore alternative labelings is so that we can gain insight into why the performance differs between the representations.

3.2.1 Expressions

We compared the clusters generated on the Expressions data to four labelings: the nominal Expression label from the data set (6 classes), the Set label (4 sets), and labels for Subject (2) and Lighting (2). Figure 3.4 shows the results. Although the performance of the two methods is similar for Expression, Set, and Lighting labels, BOF clustering is much more closely aligned to subject identity than PM, as evidenced by the significantly higher curve.

With BOF, the Subject labeling generates less cluster impurity than Expressions. While the higher curve is indicative of the fact that there are only two subjects as opposed to six expressions, it is also true that with PM, the Subject labeling does not behave the same way. The subject identity is seemingly less useful to PM when grouping the expression video samples than it is with BOF. This leads to the speculation that if this small data set were expanded to include many more subjects, the sensitivity to subject identity evidenced by BOF may lead to decreased cluster purity

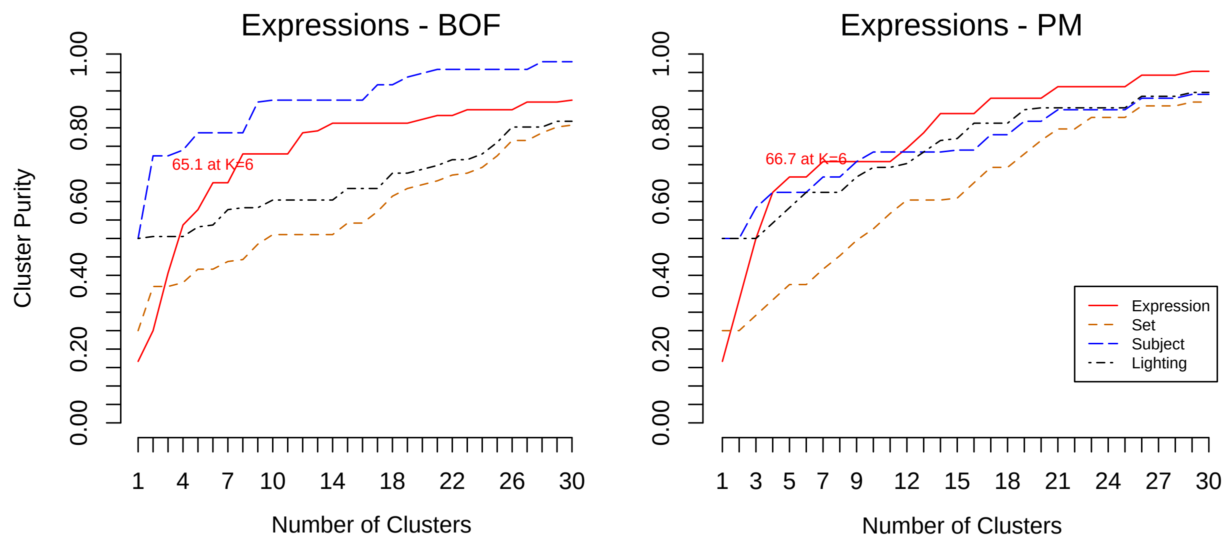


Figure 3.4: Clustering of Expressions data. BOF clusters are more closely aligned to the Subject labeling, but both methods perform similarly on other labels.

when labeling by Expression, while PM performance might be less affected.

3.2.2 Gestures

We evaluated BOF and PM clustering against the following labels applied to the Cambridge Gestures data set: Gesture (the nominal class label, 9 classes), Set (5 sets with varying lighting conditions), Direction of motion (3 motions as per Figure 2.1), and Shape (Flat, Spread, and Pair of fingers). Results are shown in Figure 3.5.

One immediately obvious aspect of Figure 3.5 is that both methods generate clusters that are nearly completely separable along direction of motion (98% purity at all K ranges for BOF and 100% for PM). At the same time, Gesture class labeling is nearly identical in performance to Shape labeling for both methods.

From this, we infer that the hierarchical clustering groups the data first by motion direction, and later by shape. Further, because the overall performance of BOF is much lower than PM, it may be that PM is doing a much better job differentiating shape, while BOF struggles in this regard. This would not be surprising because BOF discards locations of features in the representation. As such, the histogram of space-time features located near the fingertips of the spread hand and flat

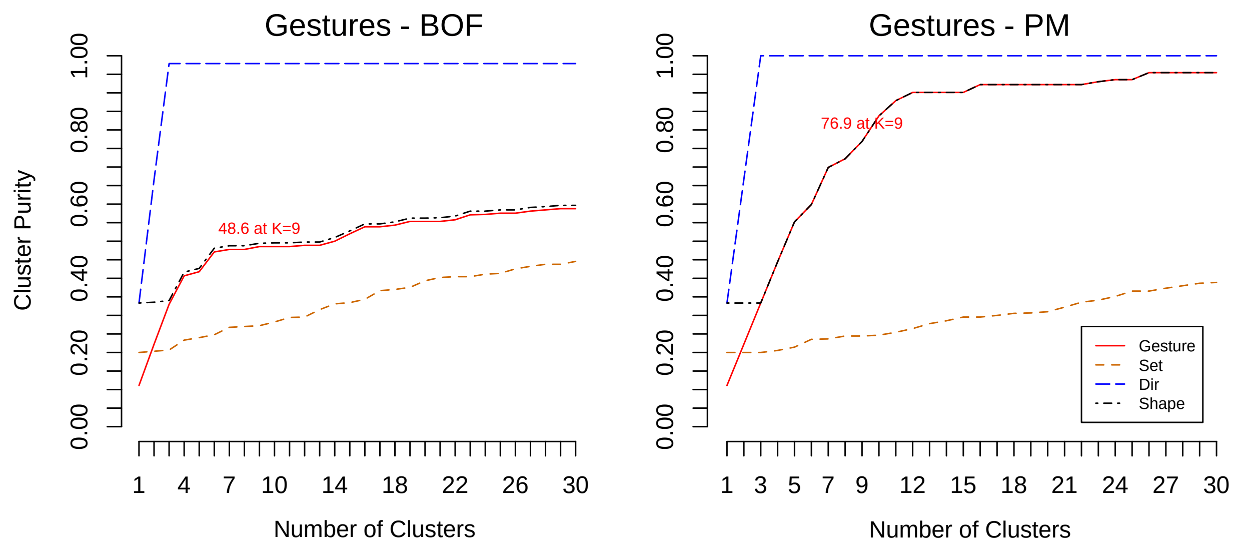


Figure 3.5: Clustering of Gestures data. Direction of motion is nearly perfectly separated by both methods. Both methods show that Gesture class labeling is limited by the Shape.

hand may look very similar, and thus difficult to differentiate. The PM method, however, treats all pixels equally, preserving location information, and thus having less confusion between the hand shapes. To test this inference, we further investigate the details of how clusters align to labels in the Gesture data set.

Table 3.1: Gesture labels compared to 3 clusters. Please refer to the body text for interpretation.

	BOF Cluster ID			PM Cluster ID		
Label	1	2	3	1	2	3
Flat Left	99	0	1	100	0	0
Spread Left	100	0	0	100	0	0
Pair Left	100	0	0	100	0	0
Flat Right	0	98	2	0	100	0
Spread Right	1	99	0	0	100	0
Pair Right	1	97	2	0	100	0
Flat Contract	0	4	96	0	0	100
Spread Contract	0	6	94	0	0	100
Pair Contract	0	2	98	0	0	100

Given the strong affinity for both methods with the three gesture directions, we investigated the cluster purity when comparing the nominal class labels to clusters when $K = 3$. Note that the gestures are defined by three shapes (Flat, Spread, Pair of fingers) combined with three motions (Left, Right, Contract), and so by limiting the number of clusters to three, we can determine if either of these aspects (shape/motion) is the first aspect of similarity that guides the grouping.

The results in Table 3.1 show both methods divide the videos into clusters according to direction of motion first. In the table, the three clusters are the columns 1-3 shown for both methods. The rows indicate the count of gestures with the given label in that cluster. Reading the table, one can see that the first cluster of BOF consists of 99 "Flat Left" gestures, 100 "Spread Left", 100 "Pair Left", and only two other videos that show a non-leftwards motion direction. Similarly, the elements of cluster 2 are dominated by rightward gestures, and the final cluster consists mostly of gestures with a contracting motion. For PM the grouping of gestures according to motion direction is 100% pure – all leftwards gestures are in cluster 1, all rightwards in cluster 2, and all contractions in cluster 3.

Recall that the clustering is hierarchical, so each of the three clusters is further broken down into sub-clusters when one cuts deeper in the dendrogram. Allowing each of these three top-level clusters to be further divided into threes, we raise K from three to nine. Since there are nine gesture labels in the data set, and we know from Figure 3.5 that the cluster purity is 48.6% for BOF and 76.9% for PM, we expect to see different behavior at this level.

Table 3.2 shows the results, and is read in the same was as explained for Table 3.1. While PM begins to differentiate based on shape, BOF struggles to do so. BOF maintains its confusion between shapes within the same direction, while PM manages to cleanly separate Leftward motion into the three Shapes, and partially separate the Contraction motion as well. BOF has only two gestures, Flat Left and Flat Contract, that are over 80% pure in their respective clusters, while PM has five of the nine at this level of purity.

This evidence supports our idea that shape is secondary to the clustering behind motion, and it appears to be the limiting factor on the overall agreement between the class labels and the clusters. Since BOF discards the feature locations in the feature vector representation, it has less power to differentiate between different hand shapes undergoing the same motion.

Table 3.2: Gesture labels compared to 9 clusters.

	Cluster ID - BOF Representation								
Label	1	2	3	4	5	6	7	8	9
Flat Left	95	4	0	0	0	0	0	1	0
Spread Left	46	54	0	0	0	0	0	0	0
Pair Left	26	74	0	0	0	0	0	0	0
Flat Right	0	0	24	35	9	13	17	2	0
Spread Right	0	1	30	25	20	6	18	0	0
Pair Right	0	1	25	26	18	4	24	2	0
Flat Contract	0	0	1	2	0	0	1	92	4
Spread Contract	0	0	2	3	0	0	1	40	54
Pair Contract	0	0	1	0	1	0	0	52	46

	Cluster ID - PM Representation								
Label	1	2	3	4	5	6	7	8	9
Flat Left	93	0	0	0	0	0	0	7	0
Spread Left	3	0	0	0	94	0	0	3	0
Pair Left	0	0	0	0	0	0	0	100	0
Flat Right	0	43	57	0	0	0	0	0	0
Spread Right	0	85	15	0	0	0	0	0	0
Pair Right	0	62	38	0	0	0	0	0	0
Flat Contract	0	0	0	100	0	0	0	0	0
Spread Contract	0	0	0	0	0	80	20	0	0
Pair Contract	0	0	0	17	0	0	41	0	42

Restating an earlier point, with no supervision, it is the inherent biases of the two representations and their related distance measures that dictate which generates clusters that are better aligned with the desired labels. As we saw with Expressions, the BOF representation aligns more strongly with subject identity than PM, which is not desirable if the behavior clustering is meant to be subject-invariant. In the case of Gestures, the design bias of BOF to ignore relative spatio-temporal positions causes it to fail in many instances to match the nominal gesture label.

3.2.3 KTH Actions

We chose the following labels to apply to the KTH Actions data set: Action (the nominal class label, 6 classes), Scene (4 scene types), Gait (2 types: gait or stationary actions, as per Figure 2.1), Location (2 types: indoors and outdoors, 75% are outdoors), and Subject (25 people). Results are shown in Figure 3.6. We did not expect either method to align clusters against the Subject label, as

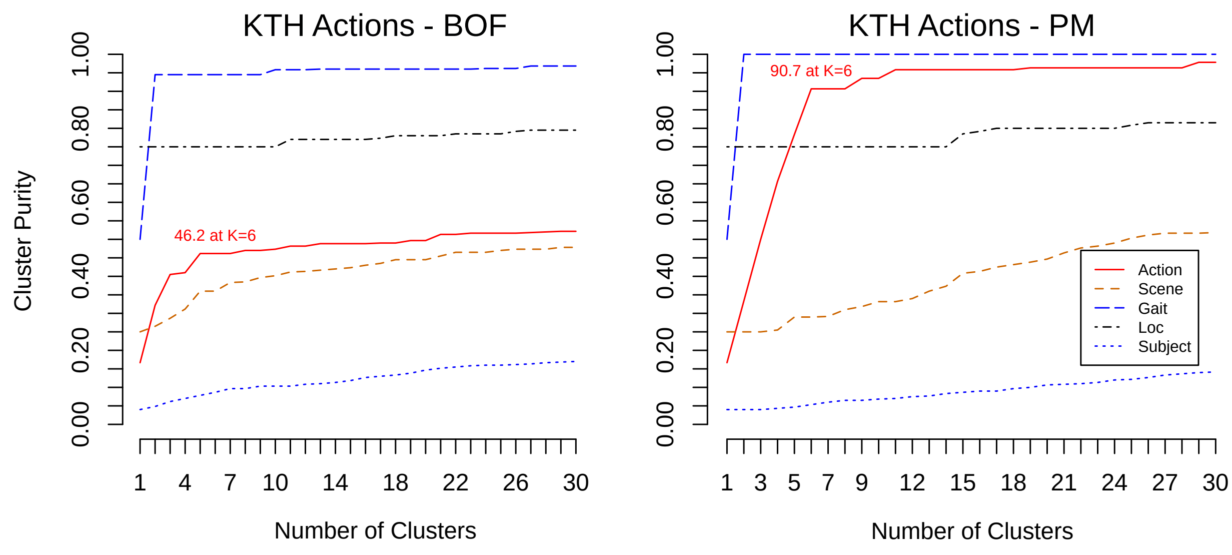


Figure 3.6: Clustering of KTH Actions. Both methods easily distinguish between Gait and Non-Gait actions. PM clustering is over 90% pure when judged against the nominal labels and restricted to $K = 6$, the number of classes in the set.

the individuals can be hard to discern, there are 25 of them, and Scene 3 uses changes of clothing to further make identifying the subject difficult. Separating the actions based on Gait labeling proved easy for both methods. Although the performance curve for Location appears high, the base rate is 75% outdoors, and the results did not rise much above that minimum score. Clustering based on PM distances was very closely aligned to the nominal class labels, as shown by the 90.7% cluster purity at $K = 6$.

Unlike with Gestures, we did not find a semantic labeling that best explains the performance of the nominal class labels. Given the high performance of PM clustering on the class labels, one is led to believe that the classes are inherently separable in most cases when using the PM representation, but not when using BOF.

Given that Support Vector Machines trained with similar BOF representations achieve classification accuracies in the upper 80's to lower 90's% (see Wang et al. [WUK+09]), it is revealing that the clustering performance is comparatively poor on KTH Actions. Because of this, we believe that supervised training may be more important for achieving high accuracy with BOF representations of full-body actions than it is for PM representations.

3.3 Conclusions

Lacking any supervision, and outside a forced-choice paradigm, it is important to design representations that are amenable to clustering human activities along semantically meaningful aspects. In this chapter, we presented performance differences between Product Manifold and Bag of Features representations over three data sets representing human facial expressions, hand gestures, and full-body actions. The pair-wise distance matrices generated by PM representations of the video samples led to superior clustering purity when compared with the nominal class labels of actions and gestures, and performed similarly to BOF on expressions.

We also found that while gross motions were easily clustered by both methods, the lack of preservation of structural information inherent to the BOF representation leads to limitations that are not easily overcome without supervised training. This was evidenced by the poor separation of shapes in the hand gestures data by BOF, and the overall poor performance on full-body actions. There are BOF-based action recognition approaches that add additional spatial information to the representation, such as by using pyramid structures [KG10]. While it may be likely that BOF clustering performance would improve with the added spatial information, the cost is in increased design and computational complexity. That said, we encourage other researchers to follow the protocol we presented to facilitate comparative evaluations for unsupervised action learning.

Chapter 4
Learning from Streaming Video

In the previous chapter, we compared the Product Manifold representation with Bag of Features and determined that the Product Manifold is more amenable to unsupervised learning. In this chapter, we present a method for learning actions from unlabeled streaming video data using the PM representation. [1]

The method proposed in this chapter requires no prior knowledge of an expected number of labels/classes, requires no silhouette extraction, is tolerant to minor tracking errors and jitter, and can operate at near real-time speed on the tested data set. Further, we show that the system is amenable to incremental learning as anomalous activities are detected in the video stream. We demonstrate performance using the ETHZ Living Room data set which is described in Chapter 2. We make no assumption on the number of actions a subject may exhibit in a given length of time, and we allow for multiple labels to be applied simultaneously. Because there remain scalability issues with the method as presented in this chapter, we address scalability of both learning and recognition in subsequent chapters.

4.1 Method

We propose an unsupervised learning method for action recognition based on clustering tracklets (as defined in Section §2.4) that are extracted from tracking subjects in streaming videos. Each tracklet captures the appearance and motion of an entity for a second or two of time. We cluster the tracklets using the Product Manifold distance. In grouping similar tracklets, we find the repeated actions in the video. We perform clustering with no foreknowledge of either the expected types or numbers of actions present in the data. The idea is to *discover* the set of action labels from the video.

[1]Material presented in this chapter was previously published by the author in [OD11b, OLD12].

The set of clusters may be given labels by the users of the system, a process we call "Selective Guidance." It is important to note that the system would work with internally generated identifiers, although it would be challenging for the user to know the significance of a generic output like "action 12" instead of "walking." From each labeled cluster, we identify a small number of exemplar tracklets that best represent the group. Not all clusters are easily described with a concise label. For those that are easily described, we can apply that label to the cluster's exemplar(s). For those clusters that are semantically meaningless, we apply no label and extract no exemplar for run-time matching. Another strategy for handling the difficult-to-label clusters would be to explicitly label them as "unknown". However, we did not explore this option.

The set of exemplars is used in a nearest-neighbor matching strategy to detect and label actions on previously unseen test video. We perform detection on streaming video without pre-segmenting the space-time regions of interest. As an entity being tracked changes behavior, the system will detect the change and apply a new action label to the subject when appropriate.

At times, a tracklet from the test video may not be a good match to any of the exemplars. In such instances, the system will apply no label to the tracklet, and it will be remembered as a novel detection. The set of novel detections can be evaluated to produce additional exemplars, and thus the system can learn over time, boot-strapped from an initial training set. Further details on the various aspects of our approach are presented below.

4.1.1 Data

We use the ETHZ Living Room data (ETHZ) in this Chapter. We selected this data set because it represents the continuous surveillance problem better than many of the more popular action recognition benchmarks, such as those used in Chapter 3.

Recall from the description in Chapter 2 that ETHZ provides three longer-duration video samples, the first of which is over 7,000 frames long. The first video (Seq1) is intended to allow an unsupervised system to learn the nominal behavior of the room's occupant. The second two videos (Seq2 and Seq3) are shorter, and are used to present novel behaviors, such as falling down or panicked gesticulations, to measure a system's ability to detect anomalous events. Figures 4.1

and 4.2 show sample images from the first and third sequences, respectively, of the data set. The annotations shown in the images, i.e. the blue bounding boxes and the action labels, are the result of the method described in this chapter.

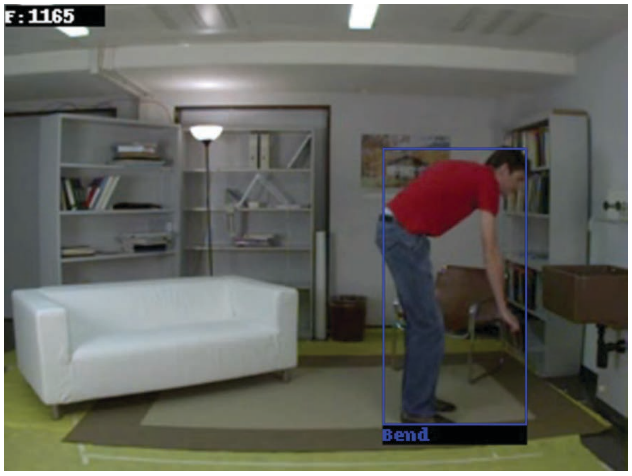


Figure 4.1: Example of an activity detection from ETHZ Seq1.

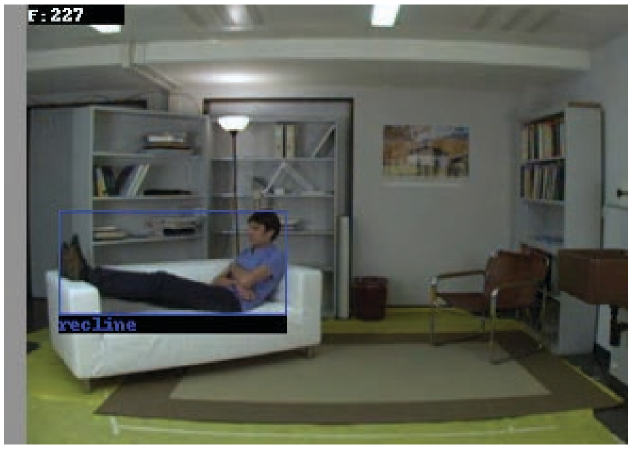


Figure 4.2: Example of an activity detection from ETHZ Seq3.

4.1.2 Tracks and Tracklets

To generate the tracks on ETHZ video sequences, we perform background subtraction using the median image of the first 2,000 frames as the background model. We use the bounding box of the foreground mask to track the subject in the video. Processing is performed using gray-scale imagery.

Action recognition approaches that rely on silhouette extraction [LJD09, NGV10] can be negatively impacted when the foreground mask is inaccurate. An important advantage to our method is that it processes all pixels within the bounding box, requiring no silhouette mask, and is therefore less sensitive to foreground/background segmentation challenges.

A single track of a person over time will give rise to numerous tracklets, some of which may clearly contain an action, and others may represent transitions between actions and thus have no clear semantic label (see Figure 4.5 for an example). The size of each frame in the tracklet is kept small in order to capture only large-scale structure, eliminate high-frequency features, and de-emphasize individual appearance. In this investigation, we create tracklets having 48 frames of 32x32 pixels, which captures about 2 seconds of activity at a time, at a frame rate of 24 fps.

To extract tracklets from the video, we employ a sliding temporal window strategy, overlapping by 16 frames, for slicing the long duration track of the room's occupant into many shorter tracklets. The bounding box of a track typically varies from frame-to-frame, and so the resulting tracklet can suffer from significant instability that negatively impacts the PM distance computation. To stabilize the tracks, we compute the bounding box that contains the spatial extent of the *entire* tracklet, and we use that box to clip tracklet tiles from corresponding frames in the video. The benefit of this simple stabilization strategy is illustrated in Figure 4.3. Beyond the stabilization benefit, this method for extracting tracklets from video also has the effect of better preserving direction and speed of motion than simply clipping the tracked region using a tight bounding box.

4.1.3 Clustering and Exemplar Selection

Given a set of tracklets extracted from the training video, we compute the pair-wise PM distances to form a distance matrix. As in Chapter 3, we use agglomerative hierarchical clustering with Ward's linkage to generate a cluster tree. The tree can be cut at a particular linkage threshold value to generate a set of clusters. With no prior knowledge of the expected number of clusters, it can be challenging to select the appropriate cut. It is an open question on how best to measure the clustering quality lacking any prior information. However, we have observed that the performance of our method rises quickly as K is increased, and then plateaus at a high level for K greater than

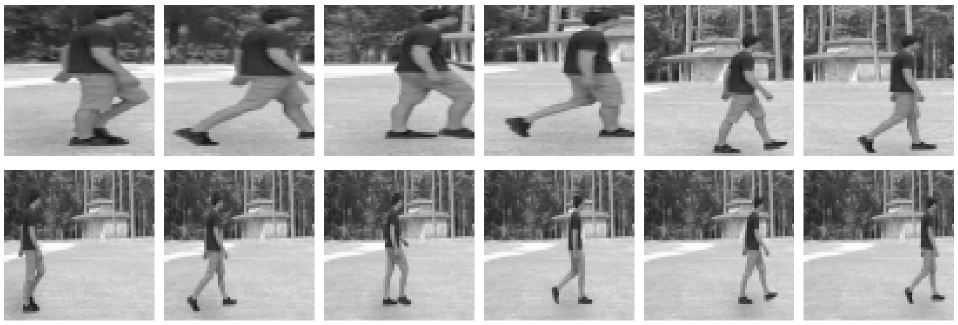


Figure 4.3: Example tracklet created without (top) and with (bottom) stabilization strategy. Top tracklet uses the bounding rectangles from the track to clip tiles from the source. Bottom uses the full spatial extent of the track within the temporal window to define a single clipping region, and thus stabilizes the images and corrects for minor track drift. In both cases, the clipped tiles are rescaled to fit the fixed tracklet dimensions.

approximately ten percent of the training sample size.

To convince ourselves that this is true, we measured the Cluster Purity (Eq. 3.2) against the choice of K, illustrated in Figure 4.4. Generating this plot requires labeling the training data, but this is not an integral part of our method. Instead, for the evaluations described later, we arbitrarily chose values of K equal to 5, 10, 15, and 20 percent of the training set size.

The PM distance measure is non-Euclidean, and there is no closed-form computation for the mean value of the samples in a cluster. Instead, exemplars (medoids) can be selected from within each cluster that minimize the sum of the distances to the other cluster members. More than one exemplar can be selected from within a cluster by removing the best medoid and repeating the process. Interestingly, we found that pulling two exemplars from clusters that represent "sitting" or "bending" resulted in one of the samples exhibiting the downward aspect of the motion and the other exemplar exhibiting the upward aspect (e.g., bending down vs. straightening back up). It may be that these videos are close enough on two of three factor manifolds to form a well-defined cluster, yet in the third factor (e.g.,vertical motion), they are separable into two sub-classes. However, we did not further explore the implications of this observation.

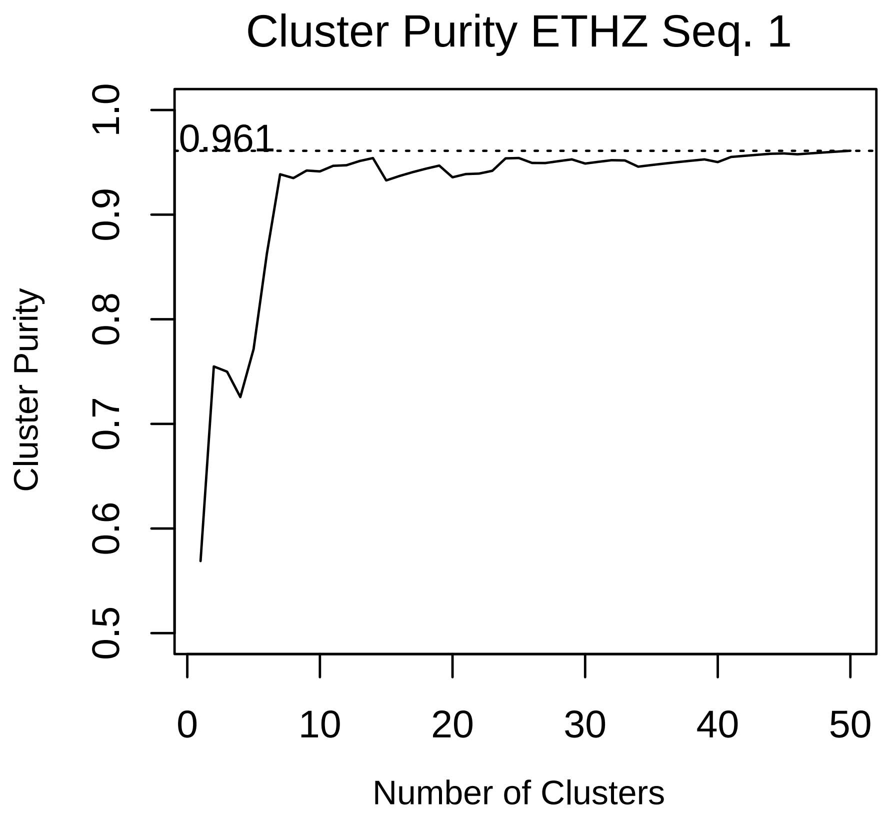


Figure 4.4: Cluster purity on ETHZ Seq1 tracklets. The sharp rise followed by a long plateau indicates many K values work well.

4.1.4 Detection

After exemplars have been trained, we use them to match against tracklets in the test videos. This process occurs in near-real-time on the streaming video. We compute the 3-Nearest-Neighbors using the PM distance between each new tracklet and the exemplars. Soft weighting is used so that selected exemplars contribute their labels to the new tracklet based on how close they are. A standard Gaussian decay is used with σ determined from the distribution of distances in the training samples. The use of soft weighting is important for detecting anomalies, so that when the 3-Nearest Neighbors are far away, we avoid blindly applying existing labels to what might be a novel observation. This consideration will be more clear as we further describe our labeling method, below.

We allow for multiple labels. Each tracklet maintains a bit vector of length equal to the cardinality of the label set. In the bit-vector, a 1 indicates the corresponding label applies to the tracklet, and a 0 means it does not. The weighted label vectors from the nearest exemplars are summed component-wise to produce the raw label vector of the new tracklet. A score threshold is applied to each component to generate the label bit vector. It is possible, even desirable, that the label vector will result in all zeros should none of the nearest exemplars be close enough to the sample.

Formally, the scoring computation is shown in Equation 4.1, where ω_i is the weight based on the PM distance $d(i, x)$ between exemplar i and tracklet x, L_i is the label vector for exemplar i, P is the number of labels, s^p is the component score computed as the weighted sum of the corresponding components from the K nearest exemplars, and L_x is the computed label for tracklet x by comparing the component scores to a constant threshold t.

$$\begin{aligned}
\omega_i &= e^{-d(i,x)^2/2\sigma^2} \\
L_i &= (l_i^1, l_i^2, \ldots, l_i^P) \\
s^p &= \sum_{i=1}^{K} \omega_i l_i^p \,, \forall p \in \{1 \ldots P\} \\
L_x &= (s^1 \geq t, s^2 \geq t, \ldots, s^p \geq t)
\end{aligned} \tag{4.1}$$

4.1.5 Anomalies and Incremental Learning

An anomaly is a tracklet that is too far from the exemplars to produce a non-zero label set. After the initial exemplars have been produced from the training data, we can run the system with a relatively high score threshold in order to generate a set of anomalous samples. We combine the anomalous tracklets with the current exemplar set, and then recompute the clustering over only the combined set (i.e. omitting all of the original training tracklets), yet keeping the number of output clusters constant. In the resulting clusters, we look for any of the anomalous samples that are not grouped with current exemplars. This subset of the anomalous samples is selected to be added to the updated exemplar set, and selective guidance is used to generate labels where appropriate, or to assign the new exemplar to an existing label if it represents a novel aspect of a known action.

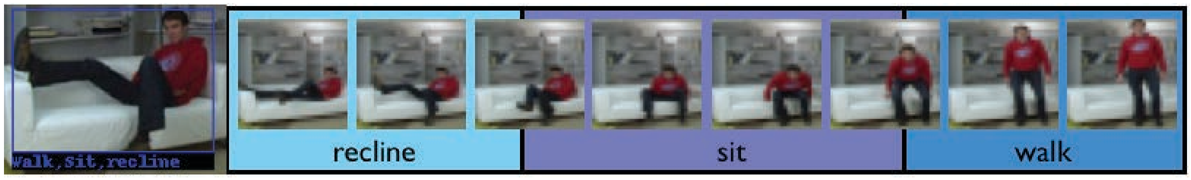


Figure 4.5: Example of a tracklet labeled from multiple exemplars. This tracklet captures the transition between multiple states, and is thus correctly described by the unordered label set {walk,sit,recline}. Sample frames are from the 48 frame, 32x32 pixel tracklet.

4.2 Results

4.2.1 Exemplar Selection

We used Seq1 to learn clusters and to generate the initial set of exemplars. We tried four values of K to use in our initial clustering, where we selected a number of clusters equal to 5%, 10%, 15%, and 20% of the number of training tracklets. Having extracted 283 tracklets from a sampling of video Seq1 for training, the values for K were 14, 28, 42, and 56, respectively. We selected one exemplar per cluster (the medoid).

Figure 4.6 shows the accuracy when performing action detection on the training set using each set of examplars, and over a range of score thresholds. The accuracy is measured in terms of the average $F1$ score between the predicted and ground truth label bit vectors. The $F1$ score is the harmonic mean of precision and recall computed over the bit vector.

It is not surprising that having more exemplars leads to better overall performance, yet the performance drop when decreasing from 56 to 42 exemplars is not severe. When using the best score threshold of 0.8, the performance drops by 3% from 56 to 42, and 8% from 56 to 28. This adds support to our belief that performance is not overly sensitive to the choice of K, as long as K is beyond the steeply rising part of the curve, as described earlier (see Figure 4.4).

In the next section we apply exemplars learned from Seq1 to detection on Seq2 and 3. We expect accuracy to drop because the testing sequences contain actions which do not appear in the training data, however this aspect also allows us to evaluate whether incremental learning of new exemplars has a positive effect on the accuracy.

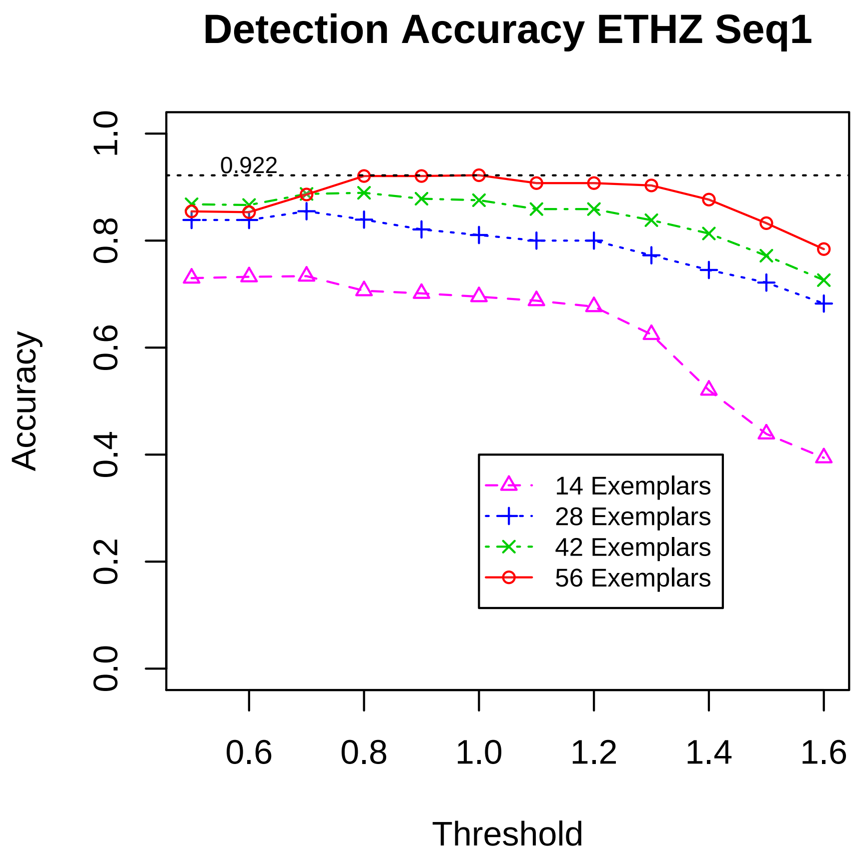


Figure 4.6: Validation accuracy using different exemplar sets and thresholds on Seq1. Accuracy is the average F1 score between predicted and ground truth label bit vectors. Threshold is the minimum sum of the weighted bits from the 3 nearest exemplars required to activate the corresponding label bit on the tracklet.

4.2.2 Action Detection and Incremental Learning

Using each set of exemplars trained from Seq1, representing approximately 5, 10, 15, and 20% of the training tracklets, we perform action detection on the other two sequences, Seq2 and Seq3. Figure 4.7 shows the results. Surprisingly, on the test data, the set of 28 exemplars performs as good or better than the larger sets over most values of the threshold parameter.

Impairing the performance of the action detection is the fact that the testing sequences contain novel actions not represented in the training. Each novel action generates multiple tracklets, each of which fail to be labeled correctly. Seq3 appears to be slightly harder than Seq2. This may be in part due to the fact that there is a lighting change in Seq3 about half way through, which causes some additional error in the low level tracking of the subject.

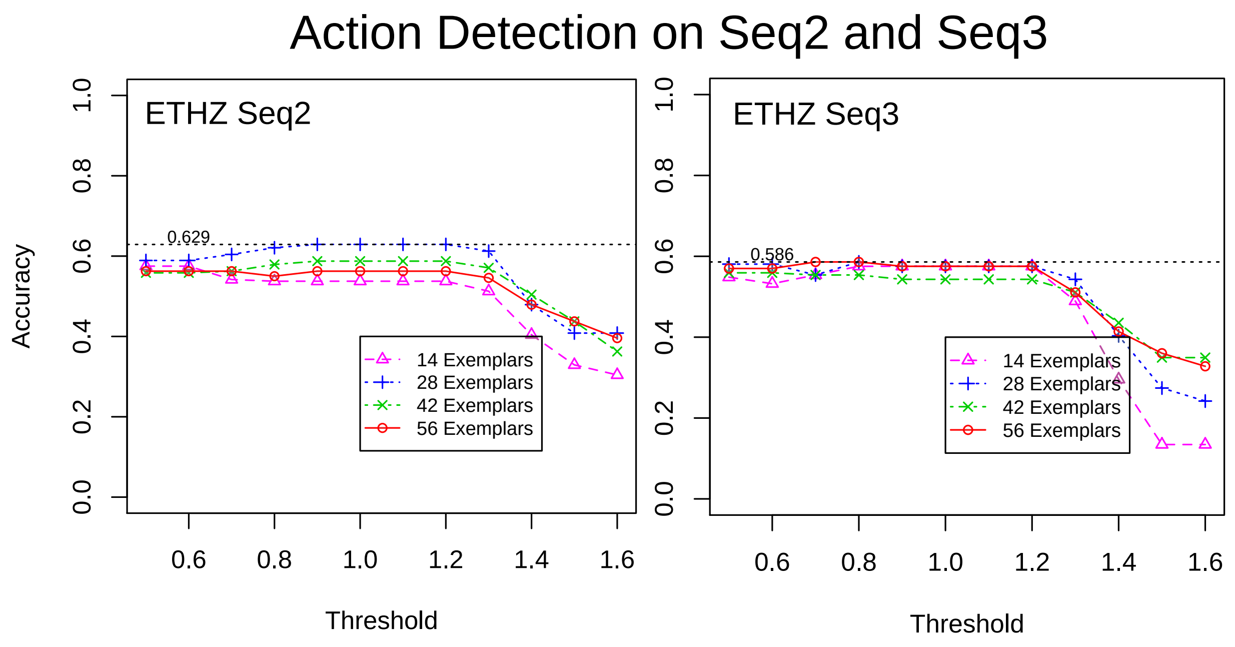


Figure 4.7: Detection accuracy on Seq2 (left) and Seq3 (right) using each of the exemplar sets learned from Seq1. Seq2 and Seq3 contain repetitions of novel actions not seen in the training sequence, so detection accuracy is expected to be impaired.

Figure 4.5 shows an example of a single tracklet from Seq2 that was given multiple labels. The detection was on a tracklet where the tracklet duration happened to contain the transition between three actions. The advantage of allowing multiple labels is that such interstitial observations may be described as a set of appropriate labels. There are no exemplars that were learned that had more than two labels. This result required the contribution from two or more exemplars that had parts that were similar enough to a corresponding exemplar.

Next, we evaluate the effect of incremental learning. Seq2 and Seq3 contain the novel behaviors: falling down, jumping, reclining on the couch, and panicking. We perform action detection on Seq2 with the expectation that novel actions will generate tracklets that are assigned no labels because they are too far from any of the exemplars, and thus will be marked as anomalous. The anomalous tracklets from Seq2 are "folded" into the labeled exemplars according to the method described in §4.1.5. The new combined exemplar set is then used for detection on Seq3. Finally, we repeat this process, reversing the roles of Seq2 and 3.

Figure 4.8 shows the set of fourteen new exemplars identified from Seq2. Three of the new exemplars were variants of the "walk" action and two represented "sit." Of the novel actions, one exemplar was selected for "jump," two for "recline," three for "fall," and three for "panic."

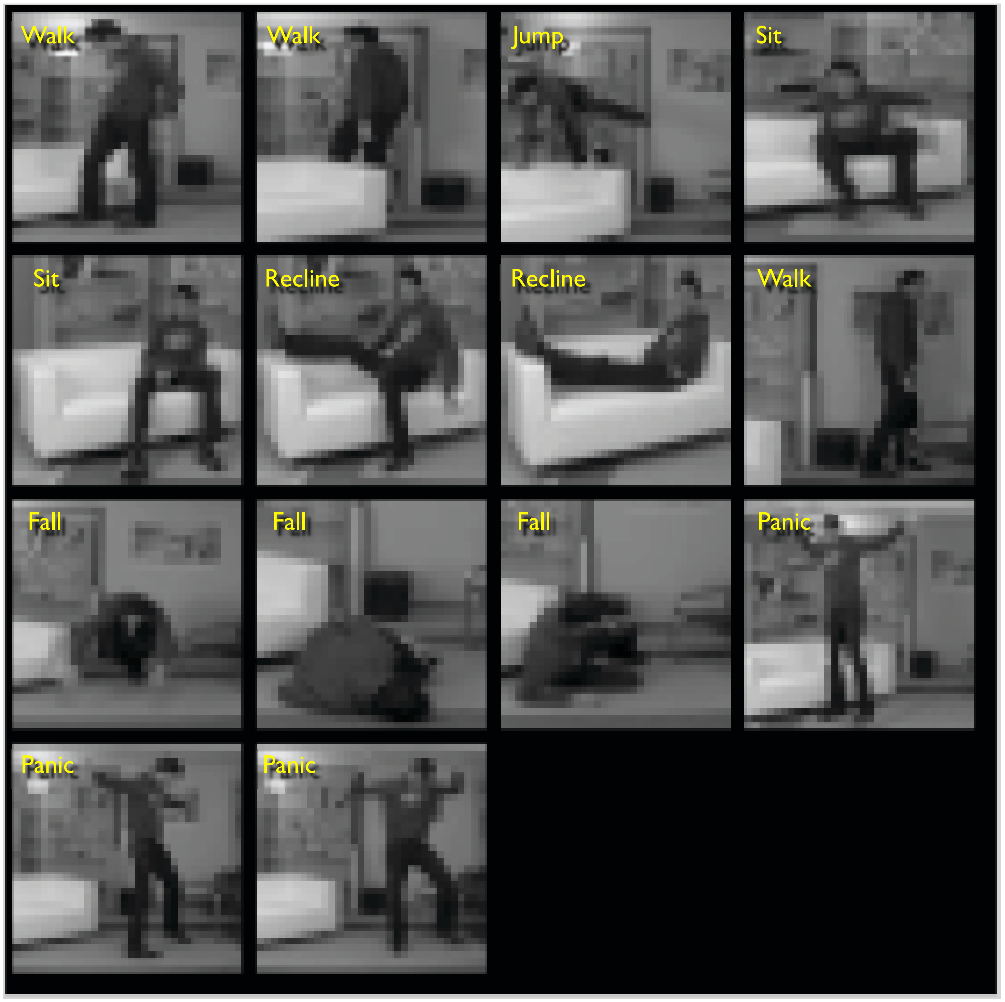


Figure 4.8: Fourteen new exemplars were learned from the ETHZ Seq2 video, representing the novel actions of jumping, reclining, falling, and panicking. Images are representative frames from the 48 frame, 32x32 pixel tracklets.

Detection accuracy before and after incremental learning is shown in Figure 4.9, which presents the detection accuracy on the test sequences with and without the incremental learning step. For this figure, only the performance for the set of 28 exemplars is shown. There is nearly a 10% performance improvement after incorporating the new exemplars. Of note is that the threshold score for best performance is lower than before. This is because with 3NN detection and novel actions that are only supported by a few exemplars, a more sensitive threshold is required.

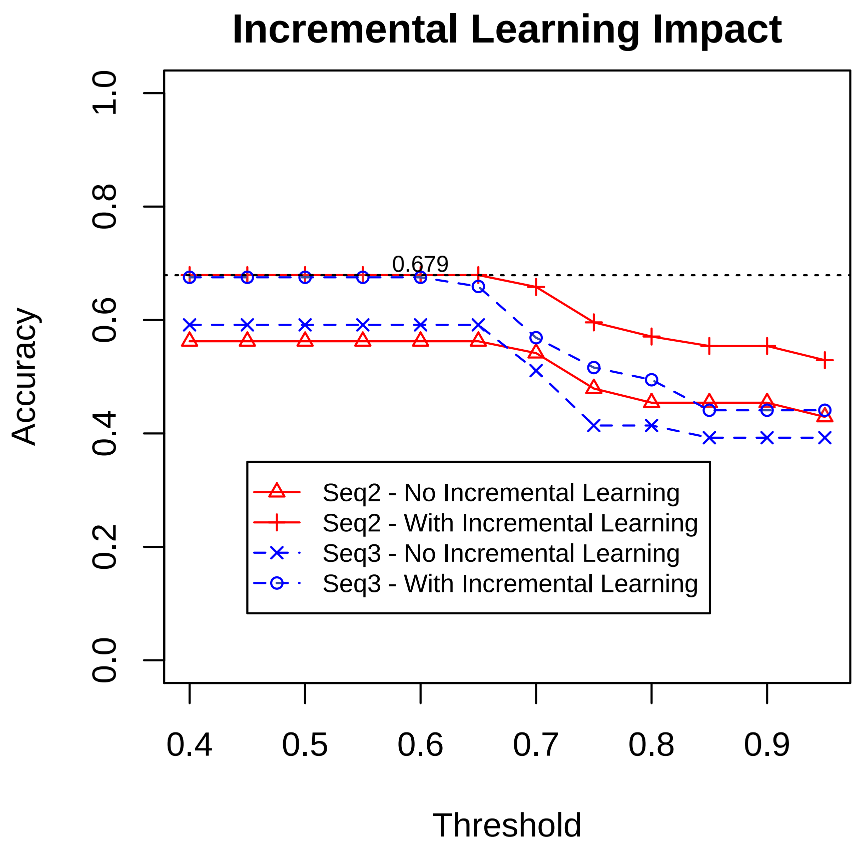


Figure 4.9: Detection accuracy with and without incremental learning.

4.3 Conclusions

We believe that to make progress on the fundamental challenge of human behavior recognition in continuous video, more research is required on open-world, incremental learning methods that require a minimum of supervision. In this chapter, we presented a step in this direction by showing how the Product Manifold representation for measuring similarity between video tracklets can be applied to unsupervised, incremental learning of actions.

In addition to those described earlier, our approach has the additional advantage of not requiring a large number of parameters and design choices. This is a clear improvement over many approaches that require parameter selection and design optimizations for feature detection, feature extraction, dimensionality reduction, codebook size, and so on. As an unsupervised method, we require no extensive training and validation stages.

However, there is a drawback with the proposed method. The clustering stage requires the construction of a distance matrix, which is an $O(N^2)$ operation in terms of the number of training tracklets. Each distance computation takes approximately 100ms, depending upon the dimension of the tracklets. For a modest sized data set, this poses no problem (less than one hour for the ETHZ data). For a data set that has ten thousand tracklets, however, computation of the distance matrix is time-consuming. There will be $N \cdot (N-1)/2$ distance computations required, each taking 100ms. Total time to construct a 10K by 10K distance matrix would be about 58 days, assuming a serial computation. In later chapters of this thesis, we address this scalability concern.

Chapter 5

Approximate Nearest Neighbors

In the previous chapter, we presented a method for learning actions from unlabeled video streams using the Product Manifold representation. We also stated that the method presented in Chapter 4 has scalability limitations. In this chapter, we begin to address the scalability concerns involved when clustering actions represented using the Product Manifold.

Our high-level approach is to use an Approximate Nearest Neighbor (ANN) index, which we define below, to improve the scalability of similarity queries over a large corpus of tracklet samples. A challenge that arises in doing so relates to the difficulties in applying existing ANN indexing methods to non-vector data representations. This chapter addresses that challenge in general, and thus takes a short detour from action recognition in order to present a method for indexing general metric spaces. In Chapters 6 and 7 which follow, we apply this ANN indexing method to action classification and clustering.[1]

Finding the (exact) K nearest neighbors of a query item in large data sets can be computationally prohibitive. In such cases, finding a set of neighbors which approximates the nearest neighbors may suffice. Approximate Nearest Neighbor (ANN) algorithms are designed to quickly yield a set of K neighbors which are close to a given query item, but with no guarantee that the set represents the K nearest.

The popularity of ANN indexing methods in computer vision has grown with the interest in large-scale image retrieval using Bag of Features representations. Nister and Stewenius were among the first to demonstrate fast image retrieval from a data set consisting of a million images and billions of descriptors [NS06]. Underlying their method's scalability is a Hierarchical K-Means (HKM) structure for indexing the visual vocabulary. Along similar lines, Mooseman et al. present randomized clustering forests for building fast visual vocabularies [MTJ06]. Philbin et

[1]Material presented in this chapter was published by the author in [OD13].

al. [PCI$^+$07] use a forest of kd-trees (KDT) to speed up k-means clustering of visual vocabularies. Silpa-Anan and Hartley explored optimizations for KDT to speed up image descriptor matching [SH08]. Another popular approach to ANN indexing is through Locality Sensitive Hashing (LSH) [GIM99], which uses a hashing mechanism such that neighboring data samples have similar hash codes.

Today, there are popular implementations of KDT and HKM that make these methods de facto standards in the vision community. FLANN (Fast Library for Approximate Nearest Neighbors) is a software package developed by Muja and Lowe that includes implementations of KDT and HKM and a mechanism for automatically selecting and tuning these algorithms for a given data set [ML09]. FLANN is also packaged as part of the well-known OpenCV library [BK08]. Another popular computer vision library containing implementations of HKM and KDT is the VLFeat package from Vedaldi and Fulkerson [VF10].

Although KDT, HKM and LSH are well-known and widely used ANN algorithms, they assume the data samples are points in a vector space. In this thesis, we focus on manifold representations for action recognition, and thus our data elements are not vectors. The geodesic distance on a Grassmann manifold obeys the triangle inequality, positivity constraint, and symmetry requirement of a metric. There are methods, which we generically refer to as metric trees (detailed in §5.1), for nearest-neighbor and ANN indexing of general metric spaces.

However, there is little precedent in the literature for employing metric trees in contemporary computer vision applications. Since data is often represented by feature vectors with similarity measured using metrics such as the Euclidean or $L1$ norm, in principle, metric trees could be used for image retrieval or nearest-neighbor classification. Instead, the use of KDT and HKM has dominated the discussion. Is the reason for this because metric trees perform poorly on vector data, or have they simply have been overlooked by those in the field?

An answer to this question comes from the analysis presented in this chapter. We introduce an ANN indexing method, called a Proximity Forest, which is a forest of randomized metric trees. In later chapters, we apply specializations of the Proximity Forest to action recognition. Before that discussion, we first validate the method in a comparative evaluation with KDT and HKM methods on vector data.

The contribution from this chapter is two-fold. First, the Proximity Forest may be the first description of a forest of randomized metric trees. Second, and more central to this discussion, upon comparing Proximity Forests with KDT and HKM, the results suggest that randomized forests of metric trees yield substantially more accurate results on traditional computer vision data representations. In the following chapters, we show how a specific variant of a Proximity Forest can be applied for scalable nearest-neighbor action recognition and unsupervised action clustering.

5.1 ANN Algorithms

A kd-tree is an index for exact nearest neighbor query that partitions a vector space by recursively generating hyperplanes to cut along coordinates where there is maximal variance in the data [FBF77]. A modification to support ANN indexing with kd-trees is the addition of a priority queue to eliminate backtracking in queries [BL97, AMN$^+$98]. Forests of kd-trees consist of kd-trees that randomize the splitting criteria by stochastically choosing the partitioning coordinate. The effectiveness and speed of KDT (e.g., [SH08]) has made this approach very popular.

ANN queries can also be supported via hierarchical k-means clustering. Nister and Stewenius describe HKM in [NS06]. There is evidence showing that HKM performs competitively with KDT, and that both outperform locality sensitive hashing (LSH) on real-world data [ML09].

Uhlmann introduced metric trees for the partitioning of general metric spaces [Uhl91]. Uhlmann presented two methods for constructing metric trees. The first way is to randomly select one of the points in the data set to be a pivot. The distances from each of the remaining points to the pivot are computed. The median distance is chosen as the partition boundary, which divides the data into two balanced sets: those closer to the pivot than the median distance, and those further away. Repeating this process recursively yields a tree structure which covers the data set. It can be viewed as recursively partitioning the metric space with median-radii hyperspheres centered on the pivot points. The second metric tree described by Uhlmann is the Generalized Hyperplane tree (GH-tree). In a GH-tree, two pivots are selected at each tree node in such a way that they are relatively far apart. The remaining points are split according to which of the two pivots they are closest to.

Contemporary with Uhlmann's developments, Yianilos developed the Vantage Point tree (vp-tree), which is essentially the same as Uhlmann's median-splitting metric tree, but the pivots are denoted as "vantage points" [Yia93]. Yianilos analyzes the performance guarantees of vp-trees, and additional pivot selection strategies. M-trees, introduced by Ciaccia et al. [CPZ97], extend metric trees with optimizations for dynamic data sets. Chavez et al. [CNBYM01] provides a survey of these and other related metric space searching techniques.

Liu et al. [LMGY04] compares GH-trees with Locality Sensitive Hashing (LSH) [GIM99], noting the following advantages of GH-trees: they automatically adapt their splitting resolution according to the density of the local data, and the hyperplanes used to partition the data can be along any direction, whereas in LSH the hyperplanes are constrained to align with coordinate directions. Importantly, Liu considers how to adapt metric trees for use in approximate similarity search applications. Liu introduced the spill tree, which is a modification of the GH-tree that incorporates an overlap region to allow for efficient approximate nearest neighbor search by eliminating back tracking for points close to partition boundaries.

For the remainder of this thesis, the term "metric tree" (uncapitalized) indicates any of several variations of tree structures for partitioning metric spaces. "Metric Tree" (capitalized), will specifically indicate Uhlmann's algorithm for median-distance based hypersphere partitioning of a metric space.

5.2 Method

Two questions are addressed by this chapter. The first is whether there is an ANN mechanism that will work on general metric data with performance comparable to vector-space methods when applied to vector data. The second is whether the most popular methods used by computer vision experts for large scale similarity search, namely KDT and HKM, have the best performance on real-world data sets compared with other options. The results indicate that the answer to the second question is also answered by the first. A simple structure based on metric trees not only provides accurate proximity queries of non-Euclidean metric data (which is demonstrated in later chapters), but is also notably more accurate than KDT and HKM when applied to commonly used image feature vectors.

A method for supporting ANN queries based on generating a forest of randomized metric trees is described below. This structure is called a *Proximity Forest*. The Proximity Forest is compared to KDT and HKM on data sets consisting of 128-dimensional integer-valued SIFT features (Scale Invariant Feature Transform [Low04]), 12-dimensional real-valued MSER features (Maximally Stable Extremal Regions [MCUP04]) and 3-dimensional real-valued point cloud data. More information on these data sets, including our motivation for choosing them, is discussed below in §5.3.1. The performance impact of the following variables is evaluated: dimensionality, data set size, distance function, and the number of neighbors to return in the query. Each evaluation is described in more detail in §5.3.

We do not directly compare Proximity Forests to LSH, as it has been shown in other work [ML09] that KDT and HKM outperform LSH for indexing SIFT features and similar computer vision data.

5.2.1 Proximity Forest

A Proximity Forest consists of a set of *Proximity Trees*, described below. Nearest neighbor computations are performed on each tree in the forest, and the best results from each tree are compared to return the nearest neighbors from the forest. Proximity Trees are constructed in a manner similar to Metric Trees. However, unlike Metric Trees, the Proximity Tree approximates the median by computing the distance between the pivot and a random subset of the input points. Doing so allows the Proximity Forest to be constructed incrementally as data arrives in a streaming source. A Proximity Tree can be characterized as a randomized metric tree.

A batch algorithm for constructing a Proximity Tree is shown in Algorithm 1. We present the batch construction for clarity even though our actual implementation builds the tree in an incremental fashion. To determine the neighbors of a sample, it is submitted to the tree and sorted to a leaf node. Sorting at each node occurs based on whether or not the distance between the sample and the pivot element is less than the current node's distance threshold. As shown in the

```
Algorithm 1: ProximityTree($S$,$\tau$,$\delta$)
    /* Recursive construction of a Proximity Tree */ ;
    Input: $S$, a set of data elements from which to construct the tree
    Input: $\tau$, the number of data elements to sample before splitting
    Input: $\delta$, a distance function
    if $|S| < \tau$ then Return /*Leaf node. Base case for recursion */ ;
    $\hat{S} \leftarrow$ random selection of $\tau$ elements from $S$ ;
    $P \leftarrow$ random selection of a pivot element from $\hat{S}$ ;
    $D \leftarrow \{\delta(x, P),\ \forall x \in \hat{S}\}$ ;
    $dt \leftarrow median(D)$ ;
    /* Partition S into left and right subsets */ ;
    $S_{\leq} \leftarrow \{x \in S \mid \delta(x, P) \leq dt\}$ ;
    $S_{>} \leftarrow \{x \in S \mid \delta(x, P) > dt\}$ ;
    $LeftChild \leftarrow$ ProximityTree($S_{\leq}$, $\tau$, $\delta$) ;
    $RightChild \leftarrow$ ProximityTree($S_{>}$, $\tau$, $\delta$) ;
```

algorithm, the distance threshold (which varies from node to node) is based on the median distance between a subset of elements to the randomly selected pivot element. The elements of the leaf node are considered the sample's neighborhood, from which the K nearest can be selected.

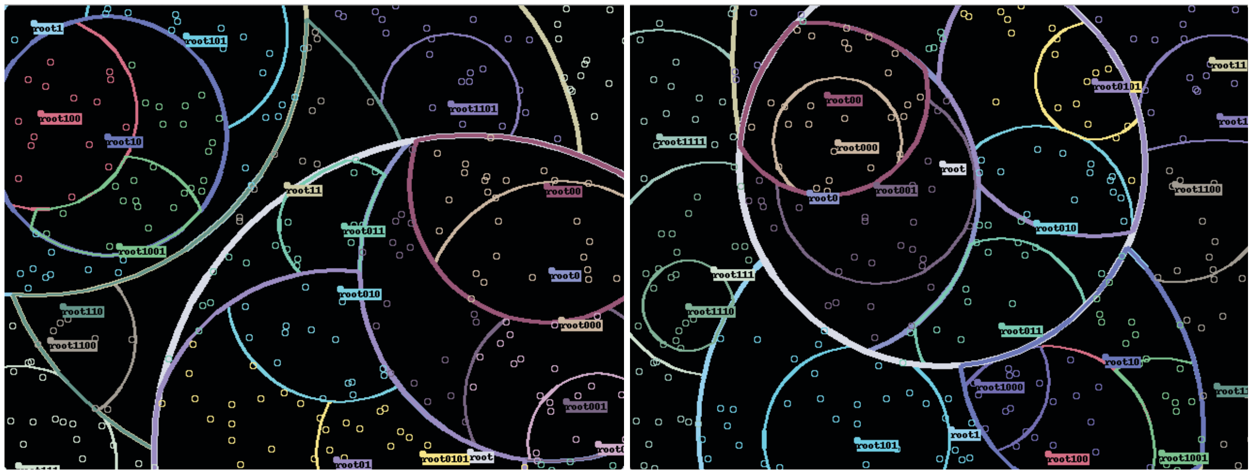


Figure 5.1: Illustration of partitioning planar data with two trees of a Proximity Forest.

Due to the near/far nature of the splitting criterion, nodes in a left subtree may often be more self-similar than nodes of a right subtree. With enough samples, the right subtrees will be broken down, recursively, into smaller subsets of greater self-similarity. Yet it remains true that the far right leaf nodes of a Proximity Tree are often less compact neighborhoods than the rest. This is mitigated by using a forest of trees, and selecting the nearest from the set returned by each tree.

5.2.2 Software Implementation

The Proximity Forest implementation uses the Python programming language. It is an unoptimized serial implementation which is available to the public under an open source license. Muja's FLANN library [ML09] provides both the KDT and HKM implementations used for comparison. The FLANN library is mature, optimized, and freely available. All code required to reproduce the results of the evaluations reported herein is also publicly available as part of the Proximity Forest code base, which is available as the "ProximityForest" project on SourceForge [O'H12].

5.3 Empirical Evaluation

This section describes five evaluations that were conducted to compare the accuracy of the three methods while controlling for: 1) data dimensionality, 2) the number of K nearest neighbors to return, 3) data set size, 4) distance measure, and 5) forest size, in the case of Proximity Forests and KDT.

All evaluations we run are using real-world data. We do not evaluate on synthesized data. Our focus is on data of broad interest to the computer vision community, and synthetic data can introduce regularities and other artifacts that can skew results. See the discussion in §4 of Gionis et al. [GIM99] on the importance of using real data sets for evaluating approximate nearest neighbor algorithms.

There is only one parameter to select when using the Proximity Forest, which is τ, the maximum size of a node before it is split (maximum neighborhood size). We set $\tau = 15$ for all evaluations reported upon in this chapter. In all charts, Proximity Forest results are denoted as PF for brevity.

5.3.1 Data Sets

The evaluation uses three data sets: 128-dimensional SIFT descriptors, 12-dimensional MSER detections, and 3D point cloud data. We use SIFT and MSER because they have been used exten-

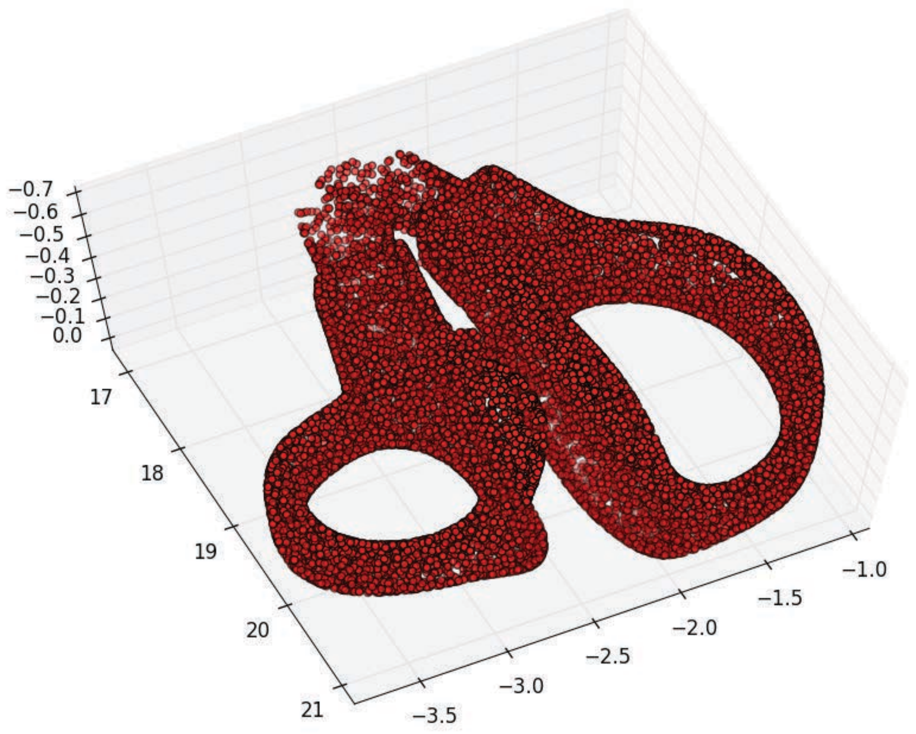


Figure 5.2: Point cloud data set.

sively in image retrieval and other computer vision applications requiring ANN indexing [SZ03, NS06, PCI$^+$07]. The SIFT descriptor [Low99] is the most cited feature representation in computer vision. The journal version [Low04], published in 2004, has over 16,000 citations according to Google Scholar, which is a remarkable average of nearly 2,000 citations per year since publication as of 2012. Indexing SIFT descriptors is thus of obvious interest to the computer vision community. The MSER data consists of 12-dimensional real-valued vectors, a convenient choice for our evaluation because the dimensionality is in between SIFT and 3D points. We use 3D point cloud data because it allows us to compare performance on low-dimensional spatial data.

The first data set consists of 10,000 SIFT features, available from the FLANN library repository.[2] SIFT features are 128-dimensional integer-valued histogram vectors.

[2]http://people.cs.ubc.ca/ mariusm/uploads/FLANN/datasets/dataset.hdf5

The second data set consists of over seven million MSER points, which were sampled to produce testing sets of size 10K, 100K, and 1M. MSER points were proposed by Matas et al. for wide-baseline stereo correspondence [MCUP04], but have since been used as feature detectors for image retrieval and other tasks (e.g., [NS06]). The MSER data are available for download from the University of Kentucky benchmark data site.[3]

The final data set consists of 250,000 points from a 3D point cloud generated by scanning the handles of a pair of scissors.[4] For illustration, 20K of these points are shown in Figure 5.2.

5.3.2 Data Dimensionality

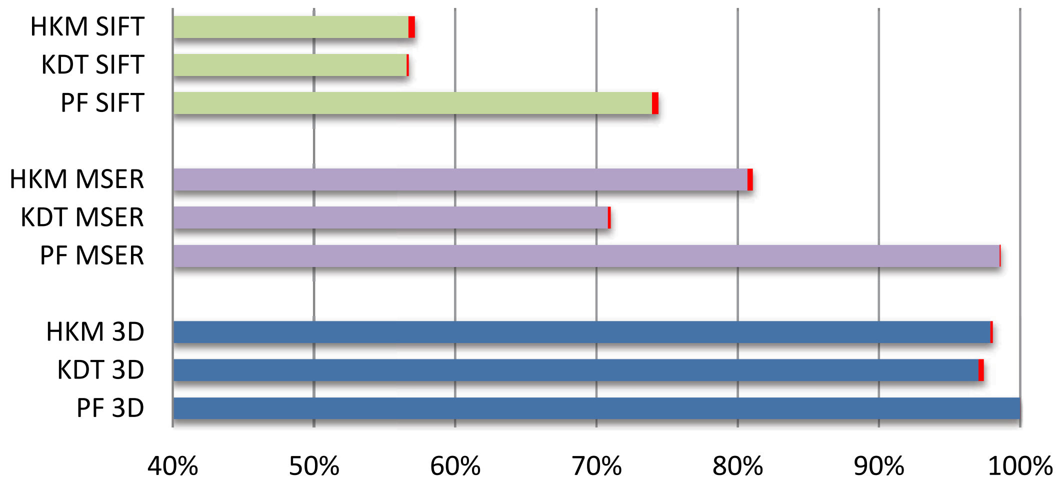


Figure 5.3: ANN Accuracy vs. Data Dimension. PF accuracy is superior to KDT and HKM on SIFT, MSER, and 3D point cloud data. Red caps at end of bars indicate the width of one standard error in the results.

In this evaluation, the data set size is fixed at 10K points, randomly divided into 9K target points used to build the ANN index and 1K query points for testing. Performance is evaluated on 3D, MSER, and SIFT features, which represent dimensions of 3, 12, and 128, respectively. For each query point, the ANN index returns the three nearest neighbors (3NN), using the Euclidean

[3]http://vis.uky.edu/ stewe/ukbench/

[4]http://www.qcgroup.com/engineering/resources/

distance.

The accuracy of the three methods, PF, KDT, and HKM, is compared against the true 3NN results. For each query, we score a point for each of the results that are one of the true 3NN. For a single trial, there are 1,000 queries, so the maximum score is 3,000. Accuracy is the percentage of the max score. All results reported in this chapter represent the average over five trials of 1,000 queries each. In addition, for each trial a new partitioning of the data into target and query points is generated.

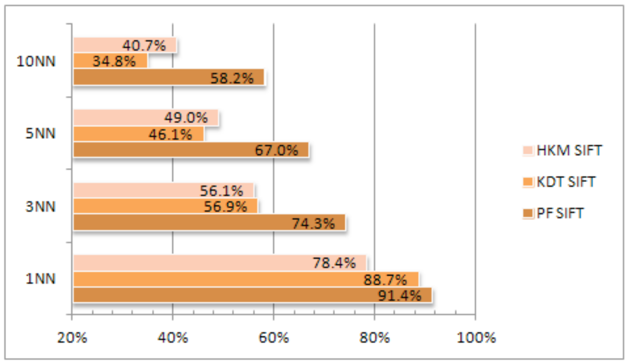
For PF and KDT, 15 trees in the forest were employed because this number is a good balance between accuracy and computational efficiency, which is shown in §5.3.6. For HKM, initially a branching factor of 128 with 15 iterations was used, in agreement with published settings from [ML09]. However, HKM accuracy was improved by using the default settings from the FLANN implementation. Consequently, the default parameter settings, a branching factor of 32 with 5 iterations, are used in all evaluations unless otherwise stated.

Results are shown in Figure 5.3. The variation in the results of all three methods is small – the standard error is shown in the figure as the red section of each bar. All three methods perform well on the 3D point cloud data, with PF correctly finding nearly all exact neighbors on all five trials. The performance difference between PF and the other methods is more pronounced for higher dimensional data, with a 15 to 20% improvement over the next best method. For the 12-dimensional MSER data, PF finds 98.5% of the exact 3NN, while the next best method is HKM at 80.5%. With the 128-dimensional SIFT data, PF gets nearly 75% of the 3NN, while KDT and HKM score less than 57%.

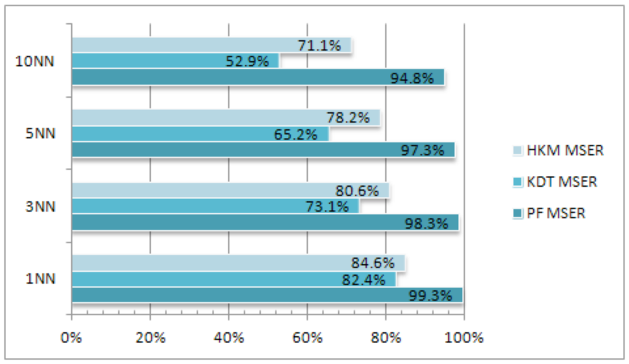
5.3.3 Varying K

In the second evaluation, the settings and data sets are as before, but we vary K to find the 1, 3, 5, or 10 nearest neighbors of every query point.

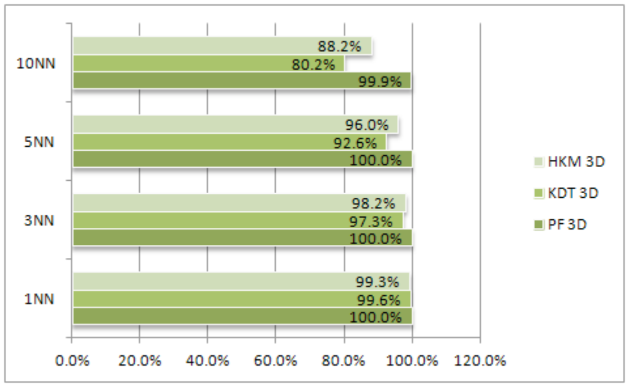
Results are shown in Figure 5.4. PF outperforms the other two methods for every value of K on all three data sets. As K increases, all methods suffer a penalty to accuracy, which is more pronounced on the higher-dimensional SIFT data. PF outperforms the best of the other two



(a) SIFT



(b) MSER



(c) 3D

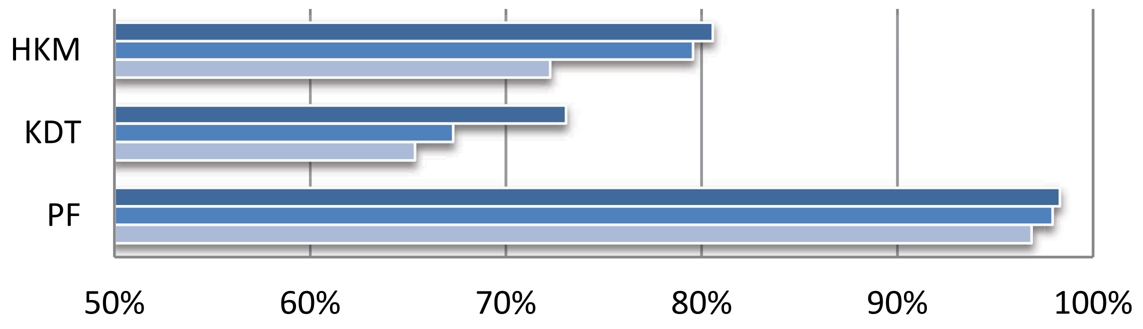
Figure 5.4: ANN Accuracy vs. K. Performance of the three methods is shown varying the number, K, of nearest neighbors to return. On SIFT data, the accuracy advantage of PF over the other methods is more pronounced for $K \geq 3$. On MSER, the performance advantage is consistent for all K. On the 3D data, the advantage is less clear due to ceiling effects.

methods by more than 15% on MSER and SIFT data for all settings with $K \geq 3$.

5.3.4 Data Set Size

The third evaluation tests whether the size of the data set used to build the index affects the relative accuracy of the three ANN methods. The MSER data was used to create subsets of size 10K, 100K, and 1M, from which 1K were selected as query points and the remaining as the target set. All other settings were as previously described, with K=3.

Results are shown in Figure 5.5. The Proximity Forest outperforms the other methods over all three data sizes. Compared to PF, HKM and KDT both experience a larger drop in accuracy between the smallest and largest data sets. These results suggest that PF accuracy may scale better to very large data sets than the other two methods.



	PF	KDT	HKM
■ 10K MSER	98.3%	73.1%	80.6%
■ 100K MSER	97.9%	67.3%	79.6%
■ 1M MSER	96.9%	65.4%	72.3%

Figure 5.5: ANN Accuracy vs. Data Set Size. PF performs between 15 to 20% better than KDT and HKM on MSER data regardless of data set size. Variation in PF performance on the three data set sizes is less than that of the other methods.

5.3.5 Distance Measure

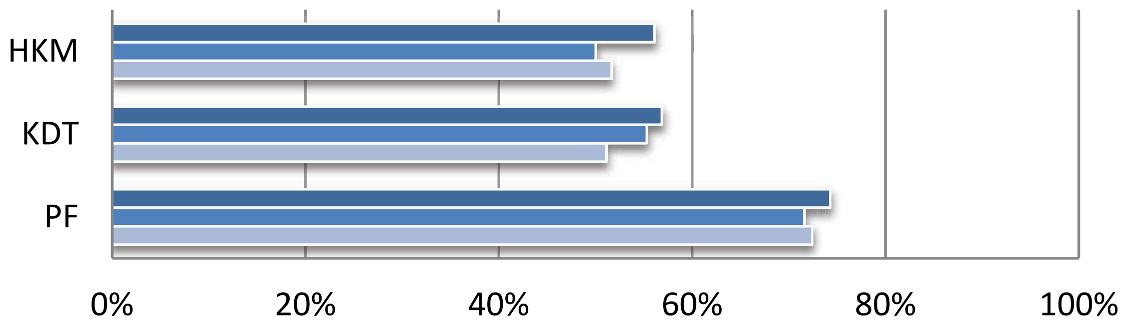
In this evaluation, three commonly used distance metrics are used to determine if this choice affects the relative performance of the three ANN indexing methods. ANN indexes were constructed using Euclidean, Manhattan, and χ^2 distance functions on the 10K SIFT data set with $K = 3$. All other parameters are the same as in the previous evaluations.

Figure 5.6 shows that PF outperforms the other two methods regardless of the choice of distance function used in the test.

5.3.6 Forest Size

PF and KDT are both methods that employ a forest of randomized trees. This evaluation compares the accuracy of these two methods against the size of the forest. We apply both methods to the three data sets, using 10K points each, and compute the 3NN accuracy with forest sizes: $\{1, 3, 5, 10, 15, 20, 25\}$. Other settings are the same as previously described.

The results are shown in Figure 5.7. PF performance suffers with forest sizes of three or fewer trees. A single randomized metric tree is not a good ANN indexing method, but with a modest



	PF	KDT	HKM
■ Euclidean	74.27%	56.87%	56.10%
■ Manhattan	71.63%	55.33%	50.03%
■ Chi-Squared	72.43%	51.17%	51.67%

Figure 5.6: ANN Accuracy vs. Distance Metric. PF maintains a 15 to 20% accuracy advantage with any of the three tested distance measures on SIFT data.

number of trees, performance significantly outperforms KDT, especially on higher-dimensional data.

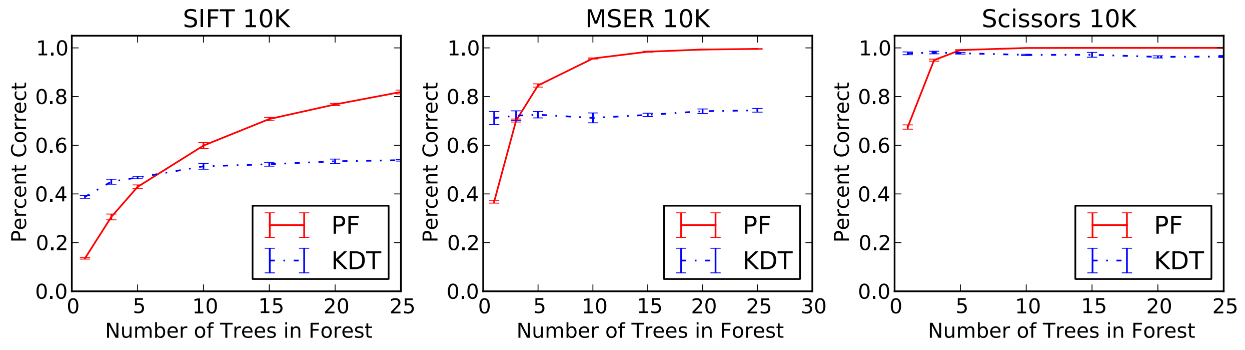


Figure 5.7: ANN Accuracy vs. Forest Size. Error bars, which are very small, show the upper and lower 95% confidence interval around the mean values. The point of the error bars is to show that the variation in performance over multiple trials is negligible compared to the differences between the methods.

5.4 Conclusion

In this chapter, attention was drawn to a method of performing approximate nearest neighbor queries that may have been overlooked by many in the computer vision community. Metric trees are structures that allow for the partitioning of general metric spaces. A forest of randomized metric trees, called a Proximity Forest, appears to return substantially more accurate ANN queries on a variety of real-word feature vector data.

No claim is made that a Proximity Forest is the best possible ANN indexing method under all circumstances. Instead, this chapter serves to highlight a practical alternative indexing strategy, and to encourage those building vision systems to think beyond current black-box ANN implementations.

In the data sets explored in this paper, the accuracy improvement over randomized kd-trees and hierarchical k-means is 15 to 20 percentage points for MSER and SIFT data. This performance gain is consistent across tests controlling for various factors that could influence indexing performance, such as data set size, distance function, and desired number of nearest neighbors (K).

By using metric trees, the Proximity Forest also has the significant advantage of being able to index any metric data. This allows the computer vision practitioner interested in large-scale similarity search to explore a wider variety of potential image/video representations. The next chapter illustrates this utility by introducing a specialization of a Proximity Forest for highly scalable and accurate action recognition using a non-Euclidean data representation and a geodesic metric.

Chapter 6

Subspace Forests

A goal of this chapter is to demonstrate an accurate action recognition method that avoids overly-complex designs (those with many parameters to tune) and can scale to large, real-world problems. From the evaluation presented in Chapter 3, we believe that relatively simple manifold representations can provide powerful discriminative properties. An open question is how to scale a nearest-neighbor manifold similarity measure to large data sets without losing accuracy.[1]

In this chapter, we present a novel structure, called a Subspace Forest, designed to provide an efficient approximate nearest neighbor query of subspaces represented as points on Grassmann manifolds. The Subspace Forest is a specific implementation of a Proximity Forest, which was presented in the previous chapter.

We apply the Subspace Forest to action recognition using the Grassmann manifold tracklet representation discussed in earlier chapters. The Subspace Forest lifts the concept of randomized forests from classifying vectors to classifying subspaces, and employs a partitioning method that respects the underlying manifold geometry. The Subspace Forest is an inherently parallel structure and is highly scalable due to average-case $O(\log N)$ recognition time complexity.

Our results demonstrate classification accuracies that are equal or superior to known published results on KTH and UCF Sports action benchmarks, and yield competitive scores on Gestures. In addition to being both highly accurate and scalable, the Subspace Forest is built without supervision and requires no extensive validation stage for model selection. Conceptually, the Subspace Forest could be used anywhere set-to-set feature matching is desired.

[1]Material presented in this chapter was previously published by the author in [OD12].

6.1 Background

In Chapter 2, we described the representation of tracklets as points on three Grassmann manifolds, each relating to the chosen axis used to unfold the 3-mode tensor into a matrix. Equation 2.3 presented the chordal distance function between points on Grassmann manifolds, or equivalently, between fixed dimensional subspaces.

The scalability challenge is to perform nearest-neighbor based classification or clustering using this distance metric between subspaces without having to compute an all-pairs comparison, which is computationally intractable for large data sets. In addressing this challenge, we adapt the Proximity Forest structure presented in the previous chapter. We call the resulting structure a Subspace Forest because it is based on the application of randomized forests to sorting subspaces of a fixed dimension.

A structural difference between the Subspace Forest and the Proximity Forest is that the Subspace Forest consists of three different types of trees – one type corresponding to each of the three factor manifolds. The Subspace Forest does not directly create an ANN indexing structure for the Product Manifold. Instead, it individually indexes each of the three factor manifolds and selects nearest neighbors of tracklets via a voting scheme.

Before presenting the details, we first provide background on related work. The following point is important to understanding a key contribution of this chapter. Our representation of actions as points on Grassmann manifolds does not directly provide a global coordinate system for the samples. Distances are computed pair-wise, and Grassmann manifolds have no unique origin. ANN indexing methods that assume vector data, such as those in the FLANN library as discussed in the previous chapter, can not be directly applied to our selected representation.

Turaga et al. describe a general method for the application of existing nearest-neighbor search algorithms to non-Euclidean manifolds [TC10]. Points are mapped from the manifold onto a tangent space, and then a vector-space ANN algorithm is applied. The mapping from the manifold to a tangent space is defined at a specific point on the manifold called the "pole", which forms the origin in the tangent space. Any point can serve as the pole, but one should select this point carefully because the mapping from the manifold to the tangent space is most accurate (in terms of preserving distances) for points closest to the pole. One mechanism to select the pole is by computing the Karcher Mean [Kar77] of the data set, which is a computationally expensive iterative process. Our approach is to avoid the errors induced by mapping points onto tangent planes by using manifold distances in our randomized forest structure.

Wang et al. present a method for developing spatial hashing functions for high dimensional data [WLZY11]. Conceptually similar to Locality Sensitive Hashing (LSH), [GIM99], Wang's Grassmann Hashing (GRASH) improves over LSH by selecting optimal subspaces for hashing using a Grassmann metric. A key step in this approach is the use of Linear Discriminant Analysis (LDA) to provide a set of subspace candidates. Unlike the Subspace Forest, GRASH requires supervision to develop the hashing functions, and requires a vector space in which LDA can be computed over the training samples.

Basri et al. [BHZM07] present a method for determining the approximate nearest subspace to a query item. Basri et al.'s method relies on random projections to reduce dimensionality, and also to map the problem to a vector-space ANN query.

The most similar work to the Subspace Forest is a randomized manifold forest developed by Bonde et al. for learning kernel hyperparameters of a set-to-set face recognition algorithm [BKR10]. Bonde et al.'s method shares the general idea of constructing randomized forests using pair-wise principal angle computations at the interior nodes of a decision tree. In comparison, the Subspace Forest uses no kernel projections, no iterations to select optimal node splitting parameters, and no supervision. Although different, Bonde et al.'s results on set-to-set face recognition provides additional support for the general utility of manifold-based randomized forests.

6.2 Subspace Forest Construction

Below, we describe the Subspace Forest and related data structures, illustrating the construction of a Subspace Forest and how to apply it for nearest-neighbor-based action recognition. To validate the benefit of the Subspace Forest, we evaluate its accuracy in comparison to contemporary methods on standard benchmark classification data sets. We investigate its scalability by demonstrating

Algorithm 2: Subspace Tree Construction

```
Data: $Node.items == \emptyset$
Data: $Node.status == Collecting$
Input: Orthogonal matrix $X$
begin
    if $Node.status == Collecting$ then
        $Node.items \leftarrow Node.items \cup X$
        if $|Node.items| > \tau$ then
5           $Node.pivot \leftarrow selectPivot()$
6           if $checkSplittingCriteria()$ then
7               $Node.threshold \leftarrow selectThreshold()$
                $Node.Left \leftarrow new(Node)$
                $Node.Right \leftarrow new(Node)$
                for $x_i \in Node.items$ do
                    $D_i \leftarrow d(x_i, Node.pivot)$                //Eq.(2.3)
                    if $D_i \leq Node.threshold$ then
                        $Node.Left.add(x_i)$
                    else
                        $Node.Right.add(x_i)$
                $Node.status \leftarrow Splitting$
                $Node.items \leftarrow \emptyset$
    else
        $D \leftarrow d(X, Node.pivot)$                //Eq.(2.3)
        if $D \leq Node.threshold$ then
            $Node.Left.add(X)$
        else
            $Node.Right.add(X)$
```

timing results on a much larger data set.

6.2.1 Subspace Tree

We define a Subspace Tree as a tree structure used to sort points on a Grassmann manifold, which define subspaces of a fixed dimension. In principle, this structure could be used to sort any set of orthogonal matrices of fixed dimension. In our application, we use it to sort tracklets which have been flattened from a data cube into a matrix, from which an orthogonal basis is computed.

The basic formulation of a Subspace Tree (SSTree) is illustrated in Figure 6.1 and outlined in Algorithm 2. Samples are added to an initially empty node until it becomes large enough to

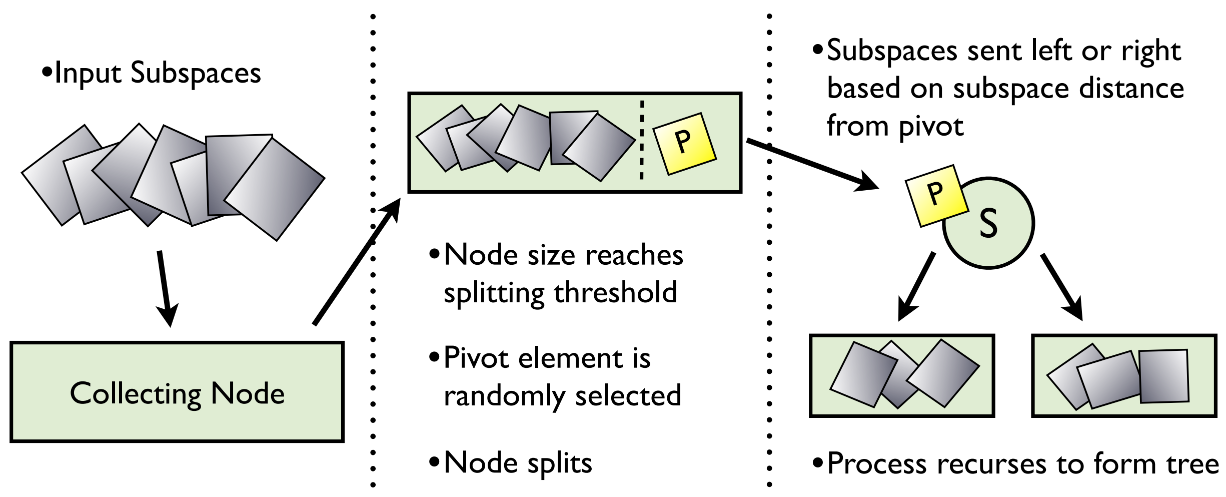

•Input Subspaces
Collecting Node
P
•Node size reaches splitting threshold
•Pivot element is randomly selected
•Node splits
•Subspaces sent left or right based on subspace distance from pivot
P
S
•Process recurses to form tree

Figure 6.1: Illustration of Subspace Tree construction.

consider splitting. We define τ to be the minimum number of elements in a node before it may be split. An element of the node being split is selected to be the pivot element. The chordal distance, Eq. (2.3), is computed between each element of the node and the pivot. Those having a distance less than or equal to a threshold are added to the left child node, the others to the right. The now-empty node is marked as a splitting node. This process recurses to form a tree, where all the samples are at leaf nodes, and all interior nodes are splitting nodes. Essentially, an SSTree is a Metric Tree (see Chapter 5) using the chordal distance function between samples.

As with Metric Trees, the SSTree imposes a local partial ordering at each splitting node, yet there is no global sort order. By voting across a forest of SSTrees, we mitigate the errors inherent to any single tree that is constructed in this manner. The Subspace Forest constructed from a set of SSTrees is discussed in §6.2.3.

6.2.2 Subspace Tree Variations

In Algorithm 2, lines 5-7 refer to functions that may implement different strategies for determining when and how to split a node in an SSTree. In this section, we discuss two node splitting variants called Median Splitting and Entropy Splitting. We also present a variation of the SSTree called the Random Axis Subspace Tree.

6.2.2.1 Median Splitting

With the median splitting strategy, the node is split as soon as the number of items in a node exceeds τ. There is no additional selection criteria, so the function of line 6 always evaluates as true. Pivot selection (line 5) is a random selection of one of the samples in the node. Threshold selection (line 7) is done by computing the median chordal distance between the pivot element and the items in the node. No labels are considered during the construction of this tree.

Note that a Median Splitting SSTree is essentially the same as a Metric Tree, as presented in the previous chapter, only the data points and distance function are designed for indexing samples represented as fixed-dimensional subspaces.

6.2.2.2 Entropy Splitting

With Entropy Splitting, the splitting criterion (line 6) is based on the distribution of the distances between the elements of the node and the selected pivot. The pivot is chosen randomly as with Median Splitting. Entropy is computed over a normalized histogram with a fixed number of bins covering an appropriate range of distances. Eq. (6.1) shows the entropy computation, where H is entropy, k is the number of histogram bins, p_i is the proportion of samples in bin i.

$$H = -\sum_{i=1}^{k} p_i \log p_i \tag{6.1}$$

When the entropy falls below an empirically-determined threshold, H_t, this indicates that the distribution has become concentrated in a small number of bins. Entropy is maximized by a uniform distribution and is zero when all samples are within a single bin. Threshold selection (line 7) is as follows. When the entropy falls below H_t, the distances are divided into two clusters and the midpoint between the cluster centers is used as the splitting threshold. Assuming τ is small, the entropy computation and clustering of the scalar distances adds a negligible amount of computational overhead in comparison to the median splitting strategy. No labels are considered during the construction of this tree. Distinct from supervised decision trees, our entropy computation is based on the distribution of distances, not distribution of labels.

6.2.2.3 Random Axis SSTree

In the previous discussion, we assumed the input to the tree was an orthogonal matrix, where the unfolding axis was selected a-priori and used for all samples. A forest can have trees representing each of the axes, but any single tree will represent only one axis. The Random Axis SSTree (RA-SSTree) takes a complete tracklet data cube as input, and sorts it per Algorithm 2, but with each new child node randomly selecting an unfolding axis. For the evaluations reported herein, we use Median Splitting for all RA-SSTrees.

6.2.3 Subspace Forest

The Subspace Forest consists of a set of SSTrees. When applied to action recognition, we select forest sizes in multiples of three so that we can have an equal number of SSTrees for each unfolding axis of the tracklets. With RA-SSTrees, any number of trees can be used because each tree has information about all unfoldings. For classification, a video clip is flattened into three factor matrices, one each along the X,Y, and T dimensions. We compute an orthogonal basis for each of the three factors. Each of the three orthogonalized unfoldings of the input video are presented to corresponding subspace trees in the forest to determine the approximate nearest neighbors for that factor. To classify a probe sample, the label of the K-Nearest-Neighbors from each leaf node is used in simple majority voting. Other voting strategies could be applied, for example soft weighting, but they are not explored in this dissertation.

Figure 6.2 shows an example nearest-neighbor query result from a small forest of three Median Splitting SSTrees. The input sample is a hand gesture from the Gestures data set (see §6.3.1.3), and the top matches from each tree are shown. Those outlined in red have the same class as the probe sample. This figure helps illustrate the benefit of including all three tracklet unfoldings in the forest. For different classes of tracklets, different unfoldings may contain the most discriminative information.

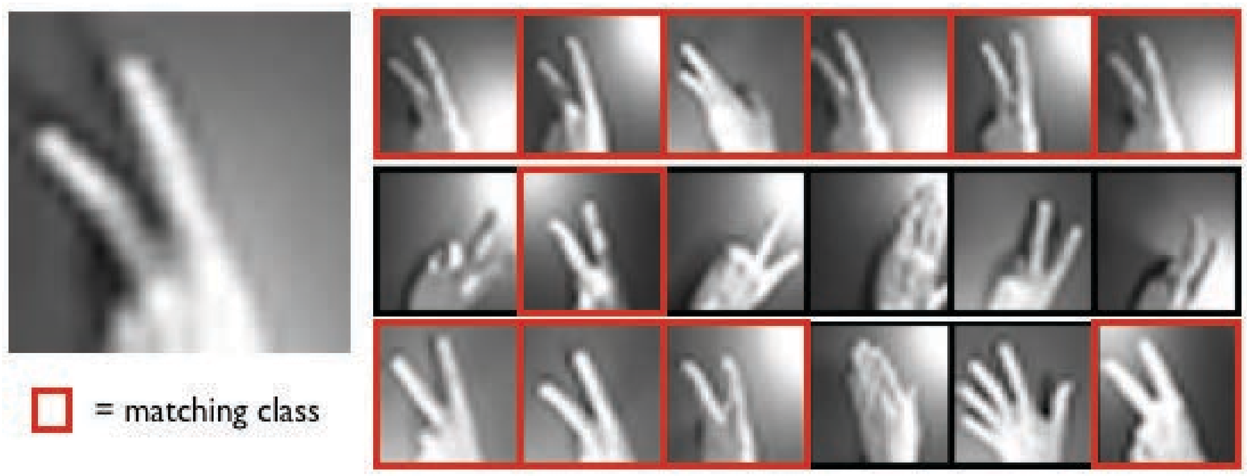


Figure 6.2: Nearest six neighbors from each of three trees of a Subspace Forest using the Gestures data set. The rows are the nearest neighbors from the three trees, one tree per unfolding. Within each row, the samples are sorted left to right according to distance from the query tracklet, which is on the left. Those matching the class of the query tracklet are outlined in red. The "Vertical Motion" unfolding, represented by the second tree (middle row), cannot distinguish left-moving from right-moving gestures.

6.2.4 Tree and Parameter Selection

We compared the performance of Subspace Forests using different SSTree variants. We use the KTH Actions data set (see §6.3.1.1) for comparing tree variants. We varied the number of trees in the forest and ran 10 trials for each combination of variant and size. The results are shown in Figure 6.3. Overall, we felt that the Entropy Splitting strategy performed better than the other two, although the difference may not be statistically significant. The best average result was 97.9% when using Entropy Splitting and 27 trees in the forest. The RA-SSTree variant performed better than the other two variants when the forest size was limited to fewer than nine trees, and yields competitive results (95.5%) even when limited to only three trees.

Beyond tree selection, only a few parameters are required with a Subspace Forest: the size of the data cube to represent the action, the number of trees in the forest, and the number of samples a node will collect before splitting, τ. When using the Entropy Splitting strategy, H_t must also be selected.

Using an Entropy Splitting Subspace Forest, we selected the value for H_t as follows. When we

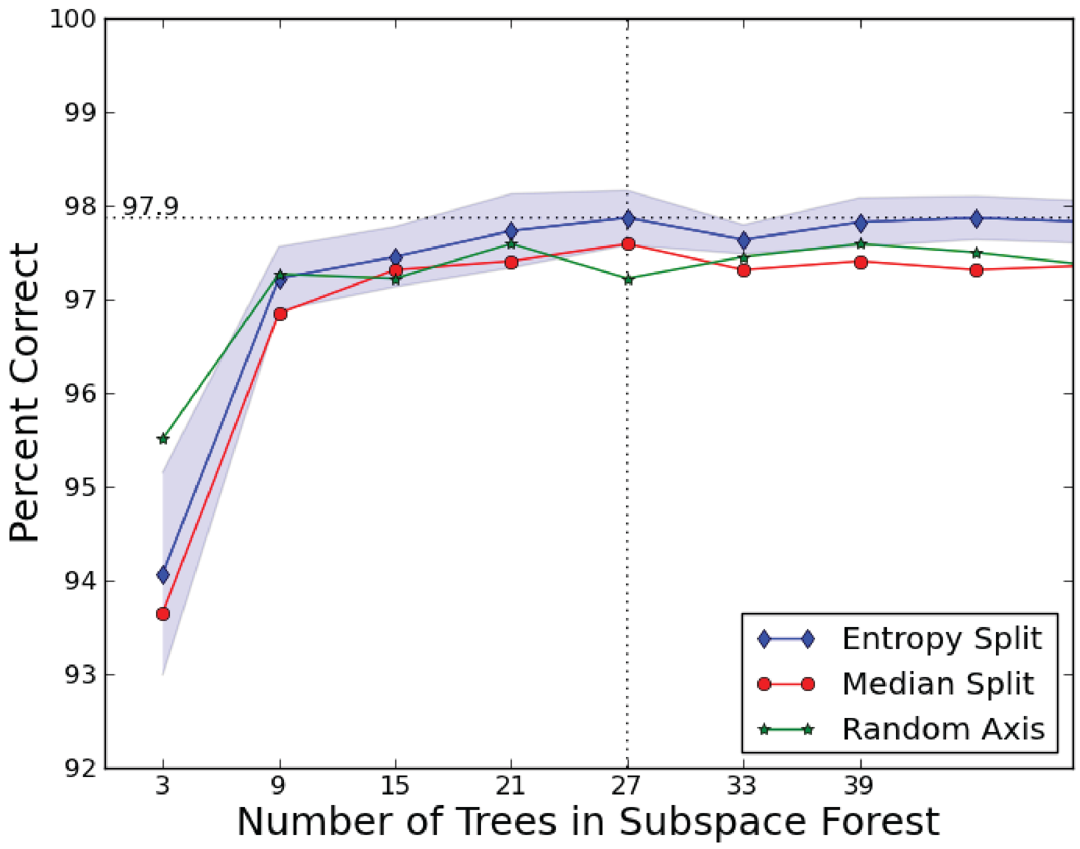


Figure 6.3: Accuracy vs. Forest Size for different tree types on KTH Actions. Each data point represents the mean of ten trials. The shaded region represents the 95% confidence interval around the mean values of the Entropy Splitting results.

compute the entropy of the distribution of distances to a pivot, we quantize the distances into 10 bins. Thus, the maximum entropy in nats is $\ln(10) \approx 2.3$. This occurs when the distances to the pivot are uniformly distributed over all the bins. When the distances to the pivot are concentrated into a single bin, the minimum entropy is reached, which is zero. Since we do not want to split tight groupings, we tested a range of H_t values between 1.8 and 2.3, which corresponds to distributions that are spread over a majority (six or more) bins. The results are shown in Figure 6.4. There are a range of values that are not significantly different in performance. From these, we selected $H_t = 2.19$ to use for the remaining evaluations.

We selected the value for τ after testing performance over a range of values from five to 25,

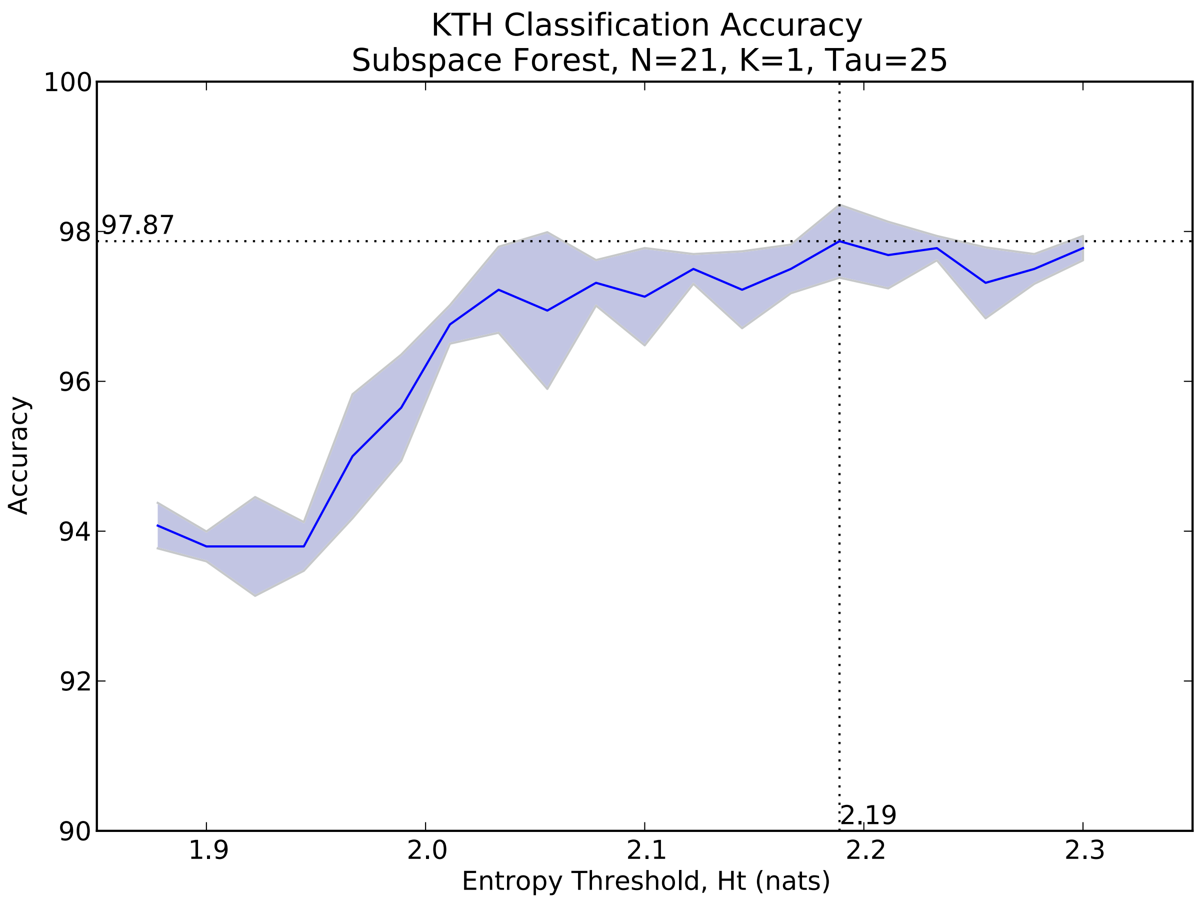


Figure 6.4: Accuracy vs. Entropy Threshold on KTH Actions. Each data point represents the mean of ten trials. The shaded region represents the 95% confidence interval around the mean values.

using an Entropy Splitting forest with 27 trees and Ht=2.19. We found little significant difference for the range of values between 15 to 25. The choice of τ is somewhat less important in an Entropy Splitting Subspace Forest than in the other variants. This is because τ controls when a node has reached a size such that it can be considered for splitting. With the Median Splitting variant, the node will always be split as soon as it collects more than τ samples. However, with the Entropy Splitting strategy, this will not occur unless the entropy of the distribution of the distances to the pivot falls below H_t.

We chose $\tau = 21$ to use for the following evaluations, but this choice was an arbitrary selection over several possible values that are statistically equivalent. The benefit of a relatively low value for τ is that the leaves will have fewer neighbors, and thus the linear search time within the leaf for the nearest neighbor is reduced. Larger values, on the other hand, have the benefit of improving the estimated median distance (for Median Splitting) or having more samples over which to compute the entropy, and thus potentially choosing better partitions.

To illustrate that the Subspace Forest does not have to be strongly tuned to work well, we use the same size and type of Subspace Forest, with the same node splitting and entropy thresholds applied to all three data sets presented in our results. The only difference in application was the selection of an appropriate cube size based on the source video resolution and temporal extent of the actions. Specifically, we used a Subspace Forest of 27 Entropy Splitting trees with τ=21, and H_t=2.19.

6.3 Evaluation

In this section, we present our results on Subspace Forest classification accuracy and scalability.

6.3.1 Classification Accuracy

We evaluate the Subspace Forest using three data sets: KTH, Gestures, and UCF Sports. We use KTH and Gestures because both data sets have a large number of samples and we know from Chapter 3 that our manifold-based representation works well. We employ UCF Sports because this data set is considered more challenging due to unequal class distribution, less controlled imagery, and fewer number of samples.

6.3.1.1 KTH Actions

Table 6.1: Accuracy comparison on KTH Actions. All results use Schuldt's training/testing partitioning of the data. CMOF is Covariance Manifolds of Optical Flow, and MSTF is Mined Space-Time Features.

Method	**Accuracy**
Subspace Forest	**97.9**
CMOF [GIK10]	97.4
Product Manifold [LBK10]	96
MSTF [GIB09]	94.5

Table 6.2: KTH Actions confusion matrix. Overall accuracy 97.9%.

	walk	jog	run	box	clap	wave
walk	100	0	0	0	0	0
jog	0	100	0	0	0	0
run	0.6	0	99.4	0	0	0
box	0	0	0	100	0	0
clap	0	0	0	0	100	0
wave	0	0	0	0	12.0	88.0

We extracted tracklets from KTH as was previously described in Chapter 3. Tracklets were 32 frames long with resolution 20x20 pixels. For testing nearest-neighbor classification accuracy, we followed the training/testing split outlined by Schuldt et al. [SLC04]. The Subspace Forest exceeds the performance of other methods, as shown in Table 6.1. Values for the other algorithms were taken from their respective publications, and all followed the same training/testing protocol. It is hard to draw strong conclusions on the significance of the difference in accuracy among top methods, given ceiling effects and implementation variations, yet the Subspace Forest classification method is clearly a top-performer with the additional practical benefits of simplicity, speed, and scalability.

We believe the Subspace Forest is one of the fastest methods as well, justified as follows. With our single-threaded python implementation, the total time to build the Subspace Forest on the training partition and recognize all the actions in the test partition is 6.5 minutes combined.

To our knowledge, none of the authors who have reported leading recognition accuracies on KTH have published the computational times required by their algorithms. It is our experience with feature sampling methods that leads us to believe that the Subspace Forest is much faster in both training and recognition time that most published methods. In Wang et al.'s evaluation of feature detectors and descriptors for BOF action recognition [WUK$^+$09], the authors report that the Cuboids detector and descriptor, as a combined unit, operates at approximately one frame per second. Other combinations in the evaluation operate between 0.8 and 4.6 frames per second.

One frame per second equates to approximately three minutes per video sample (based on average 180 frames per segmented video clip) to extract features. The time required to train a classifier using feature sampling methods consists of sampling features from the training data, constructing a codebook, representing each of the training videos as feature vectors over the codebook entries, and finally training a Support Vector Machine or other classifier. Just processing the feature sampling component performed on the KTH training partition of 384 videos would take about 19 hours with the Cuboids algorithm. With the fastest feature sampling method from Wang et al.'s evaluation, this time might be reduced to about 4 hours. This is in strong contrast to the 6.5 minutes total time (training plus testing) required to process the KTH recognition protocol using the Subspace Forest.

Table 6.2 shows the confusion matrix. The Subspace Forest has little confusion between the similar classes of walking, jogging, and running, whereas it has some errors separating waving from clapping. This is in contrast to some other algorithms, and what might be expected of human judgment. Gilbert et al.'s Mined Space-Time Features [GIB09], reports the most confusion between the jogging and running classes. Some of the KTH subjects exhibit running in a way that is understandably confusing with jogging, yet no human would confuse any of the waving samples with clapping. This leads to the speculation that a combination of representations might further improve performance.

6.3.1.2 UCF Sports

Table 6.3: Accuracy comparison on UCF Sports. All results use leave-one-out protocol. AFMKL is Augmented Feature Multi-Kernel Learning and DTM is Discriminative Topics Modelling.

Method	**Accuracy**
Subspace Forest	**91.3**
AFMKL [WXDL11]	**91.3**
Tangent Bundle [LB11]	88
DTM [BLGX10]	86.9

For UCF Sports, we use 64-frame tracklets of 32x32 pixels, in order to capture the longer duration and spatial extent of the samples. Tracklets are selected from the middle frames of the samples, and using the bounding boxes provided by the data set, except for 10 videos where the bounds are not provided. For those 10, we created the associated tracks, and this additional information is available from us upon request. As with KTH, frames are repeated for samples with too few frames.

Table 6.4: UCF Sports confusion matrix. Overall accuracy 91.3%. Actions are: DV=diving, GS=golf swing, K=kicking, WL=weight lifting, HR=horseback riding, R=running, SK=skateboarding, PH=pommel horse swinging, HB=high bar swinging, and W=walking.

	DV	GS	K	WL	HR	R	SK	PH	HB	W
DV	100	0	0	0	0	0	0	0	0	0
GS	0	94.4	0	5.6	0	0	0	0	0	0
K	0	0	100	0	0	0	0	0	0	0
WL	0	0	0	100	0	0	0	0	0	0
HR	0	8.3	0	0	91.7	0	0	0	0	0
R	0	0	0	0	7.7	84.6	0	0	0	7.7
SK	0	0	0	0	0	0	58.3	0	0	41.7
PH	0	0	0	0	0	0	0	100	0	0
HB	0	0	0	0	0	0	0	0	100	0
W	0	4.5	0	4.5	0	0	9.1	0	0	81.8

We followed the standard protocol for UCF Sports, which is leave-one-out classification. With the small size of this data set, even using leave-one-out, there are only 149 samples for use in constructing the Subspace Forest, so we doubled the gallery size by adding horizontally-mirrored copies of each training tracklet. Our results are as good or better than other published methods, as shown in Table 6.3. The confusion matrix in Table 6.4 shows that many of our errors were from confusing skateboarding with walking.

6.3.1.3 Cambridge Gestures

Table 6.5: Classification accuracy on Cambridge Gestures using protocol from [KWC07]. TCCA is Tensor Canonical Correlation Analysis.

Method	**Accuracy**
Subspace Forest	89.8
Tangent Bundle [LB11]	**91**
TCCA [KWC07]	82

We extract tracklets from the Gestures data set in the same way as presented in Chapter 3. Tracklets are 32 frames long with resolution 20x20 pixels. For classifying the gestures, we use the training/testing split outlined by Kim et al. [KWC07], which requires training on Set 5 (180 samples), and testing on Sets 1-4. Our results are competitive with leading published results.

6.3.2 Scalability

Determining the nearest neighbors using a Subspace Tree requires $D + \tau - 1$ distance computations, D decisions to determine the leaf node, and then an additional $\tau - 1$ computations, at maximum, to determine the nearest within the leaf node. Each chordal distance computation takes about 10 milliseconds using a video cube of dimensions 20x20x32. For example, with a tree constructed from 2,500 gallery samples, the expected depth is $\log_2(2500) = 11.3$, and with $\tau = 21$, the time to select the approximate nearest neighbor is about 0.31 seconds. Note that this is the time to recognize an entire tracklet, not to process a single frame. With 32-frame video clips, our *effective frame rate is about 103 fps*, for a single tree. This structure is highly scalable due to the $O(\log_2 N)$ time complexity. Increasing from 2,500 to 10,000 gallery videos, the effective frame rate drops to 96 fps, and with 1,000,000 the recognition frame rate would remain at a respectable 80 fps. Tree construction is $O(N \log_2 N)$.

The Subspace Forest has the same time complexity as the Subspace Tree, with a constant factor per tree assuming a serial implementation. Forests can be trivially parallelized by having a computational node per tree, which would allow for very efficient build and recognition times for large-scale data sets. For the evaluations described in this Chapter, we used a serial, single-threaded, python implementation on commodity hardware.

Figure 6.5 demonstrates actual run-times required to generate Subspace Forests of different sizes. The figure shows how long it takes to construct a Subspace Forest, in minutes, in relation to the number of samples in the training data. The two lines represent Subspace Forests of three and six trees, using Entropy Splitting. Tracklet dimensions are 32x32x32. The slopes of the lines show a slight super-linear growth, which agrees with the $O(N \log_2 N)$ complexity analysis. The light gray guide lines show that as the number of trees in the forest doubles, the construction time with our serial implementation also doubles from 20 to 40 minutes. A larger forest of 27 trees would take nine times longer to build than a three-tree forest with our serial implementation, but this additional time is easily mitigated with parallelism.

The time to construct a forest in Figure 6.5 is longer than previously discussed for KTH due to the larger tracklet dimensions that were used. The chordal distance function involves using an

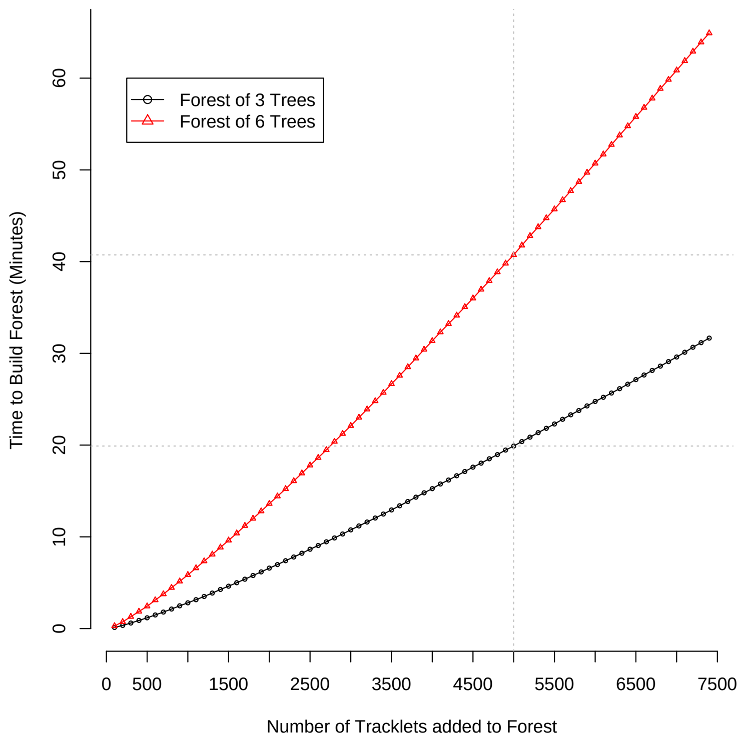


Figure 6.5: Subspace Forest construction time compared to the size of the training data.

SVD decomposition, which is approximately $O(n^3)$ where n is the total number of pixels in the tracklet. A tracklet with dimension 32x32x32 = 32,768 pixels has 2.56 times the number of pixels as a tracklet of dimension 20x20x32 = 12,800, which means each chordal distance computation takes about 17 times longer. The computational complexity of the Subspace Forest is given in terms of the number of tracklets, but there is a factor in the speed of each comparison that depends upon the tracklet dimensions.

6.4 Conclusion

Lui et al. [LBK10] showed that videos of human actions can be represented as points on Grassmann manifolds, and that the geodesic distances between two videos represented this way is an effective method for determining their similarity. In this Chapter, we introduced the Subspace Forest for determining the approximate nearest neighbors of points on Grassmann manifolds, which can be used to rapidly classify actions based on a set of labeled training samples. The Subspace

Forest works in a manner conceptually similar to randomized forests for approximating nearest neighbor computations, but respects the Grassmann manifold geometry. With the Subspace Forest, action recognition can be performed with high accuracy and with the ability to scale to much larger systems than is feasible with many previously-published approaches.

Other strengths of our method for action recognition include the simplicity of implementation, and the avoidance of the many design decisions and parameter selections that are required by Bag of Features methods. We build the Subspace Forest without using labels, so it is well-suited for unsupervised learning applications, which we describe in the next chapter.

Chapter 7

Latent Configuration Clustering

This chapter addresses the general challenge of efficiently clustering data samples where only a distance metric exists, and where the cost of computing all nearest neighbors is prohibitive. We present a method, called Latent Configuration Clustering (LCC), that transforms samples in $O(N \log N)$ time into a sparsely-connected weighted graph. From the weighted graph, a number of popular clustering mechanisms can be applied, and comparisons are between scalar values. Latent Configuration Clustering, unlike other techniques designed for clustering points on smooth manifolds, does not require explicit mapping to a Euclidean configuration space.

Some methods can cluster data given only a distance or affinity matrix. If an all-pairs comparison is required, generating the distance matrix is a limiting factor for large-scale applications. Existing techniques can generate distance matrices in $O(N \log N)$ time using spatial partitioning strategies (see the brief review in the next section), but many of these make vector space assumptions and cannot be applied to non-Euclidean metric data. For the specific case of manifold data, a general approach is to first map the data to a Euclidean space. Computing a Euclidean approximation of the data can be error prone and computationally expensive.

The ability to efficiently cluster samples using only pair-wise distance computations is not just a theoretical curiosity, but has applications to real-world problems. Previously, we demonstrated the benefit of using a non-Euclidean representation for clustering tracklets, but the cost of computing all-pair distances made the application to large scale data sets intractable.

With the Product Manifold representation, each distance comparison takes approximately 100 milliseconds, depending on the size of the video cube used to capture the actions. Given a data set of 5,000 samples, the all-pairs comparison would take about 347 hours (14 days). Doubling the size of the data set to 10,000 samples increases the computation time to nearly 1389 hours (58 days). Distributed computing could help mitigate some of the scalability concerns, but a clustering method that retains the advantages of the representation while reducing the computational complexity would be of significant value.

7.1 Clustering Data using Only Pair-Wise Distances

Manifold learning algorithms can find low-dimensional Euclidean configurations for data using only neighborhood information. A few notable examples include Isomap [TSL00], Local Linear Embedding (LLE) [RS00], and Locality Preserving Projections [HN03]. Similarly, one can map points residing on certain matrix manifolds to Euclidean tangent spaces. Assuming locality is sufficiently preserved in these quasi-isometric mappings, one can cluster data in the resultant Euclidean space using any technique appropriate for vector data.

Generally speaking, mapping non-Euclidean data to a Euclidean configuration involves some error and potential loss of locality. Isomap, for example, can be subject to topological instability when neighborhood distances are large enough to cause "short circuit" errors [BS02]. Tangent space mappings on manifolds require the selection of a "pole," or point at which the Tangent is incidental to the manifold surface. The approximate isometry of the mapping degrades the further away from the pole the samples lie (as discussed in [TC10], and other sources). Choosing the pole, e.g., via a Karcher Mean, can be a computationally expensive, iterative process.

Many manifold learning and clustering methods can take as input a distance/dissimilarity matrix. For scalability concerns, it is important to avoid an all-pairs comparison of the data. A common processing step for manifold learning methods is to compute the K-nearest-neighbors at every point (i.e., all nearest-neighbors). If the input data is in a metric space based on an L_p norm, then it has been shown that the all nearest neighbor problem can be solved in $O(N \log N)$ time [Vai89]. Additionally, there are computationally efficient Approximate Nearest Neighbor (ANN) algorithms, such as Locality Sensitive Hashing [GIM99], forests of randomized K-D Trees [ML09], and Balanced Box-Decomposition Trees [AMN$^+$98]. However, lacking a vector space, these methods can not be applied, and thus the all nearest-neighbor problem requires an $O(N^2)$ all-pairs computation.

In Chapter 5, we introduced the Proximity Forest structure for accurate ANN indexing, and in Chapter 6, we showed that it can be applied in the form of a Subspace Forest for indexing action tracklets. With LCC, we employ the Proximity Forest to speed up clustering of metric data, where the computation of the metric between a pair of data elements is computationally expensive. Although we are motivated to cluster tracklets represented as points on Grassmann manifolds, LCC is a general method to cluster any set of data objects equipped with a distance metric. Similar to manifold learning approaches, we use neighborhood connectivity, but we compute only a fraction of the all-pairs distances. LCC avoids potentially error-inducing explicit mappings to Euclidean configuration spaces.

7.2 Latent Configuration Clustering

The power of Latent Configuration Clustering is the ability to efficiently and accurately cluster arbitrary metric data. The key is the use of a Proximity Forest to transform the data into a sparsely connected graph using only $O(N \log N)$ distance comparisons. Having transformed samples from an unknown configuration space into a sparse affinity matrix, our final step is to apply an existing clustering method on the graph data. Some clustering methods, such as DBSCAN [EKSX96] and hierarchical single linkage [FHT09], can cluster connectivity graphs in $O(N^2)$. It is important to note that after the transformation, the complexity of the individual comparisons is reduced from being very costly (e.g., chordal distance) to being very cheap (scalar comparisons).

The high-level procedure for LCC consists of three steps, as shown in Algorithm 3. Each step is further explained below.

Algorithm 3: LatentConfigurationClustering(S, δ)

```
Input: $S$, the set of data objects of type $T$
Input: $\delta$, a distance function $\delta : T \times T \rightarrow \Re$
begin
1   $\Phi \leftarrow GenerateProximityForest(S, \delta)$ ;
2   $G \leftarrow GenerateConnectivityGraph(\Phi)$ ;
3   $return \quad WardsLinkage(G)$ ;
```

7.2.1 Generating the Proximity Forest

The first step requires the construction of a Proximity Forest on the data, as described in Chapter 5. Each tree in a Proximity Forest sorts samples into leaf nodes of a maximum size, τ. Each leaf is considered an approximate neighborhood for the samples it contains. We use these approximate neighborhoods to generate the connectivity graph for LCC, as described below.

7.2.2 Clustering using Connectivity Graph

We examine the leaf nodes of the forest to determine neighborhood relations. The number of trees in the forest where a pair of samples are neighbors (i.e., are in the same leaf node) indicates the similarity of the samples. We build a weighted undirected graph, $G(V,E)$, where the nodes are the data elements and the edge weights, $W_{i,j}$, represent similarity scores between nodes i and j.

Let Φ be the Proximity Forest consisting of p Proximity Trees. Let the function $I_k(X,Y)$ indicate that samples X and Y belong to the same leaf node in tree k (1 if true, 0 if false). Then the weights of $G(V,E)$ are specified as follows.

$$\begin{aligned} &V_i, V_j \in V \\ W_{i,j} = &\sum_{k=1}^{p} I_k(V_i, V_j) \end{aligned} \tag{7.1}$$

The computational complexity of this step is $O(N)$. There are on average N/τ leaf nodes per tree, each of maximum size τ. At each leaf node, all pairs of samples are scanned to update the connectivity graph, which takes a maximum of $\tau \cdot (\tau - 1)/2$ operations. For each tree, we thus have $N \cdot (\tau - 1)/2$ operations. There are p trees in the forest, so the total time to generate the graph is $p \cdot N \cdot (\tau - 1)/2$, which is $O(N)$ since p and τ are small constants.

After generating $G(V,E)$, one can apply any clustering mechanism that works on graph data to the affinity information encoded by the edge weights. We employ hierarchical agglomerative clustering with Ward's linkage, as in previous chapters, and have not significantly explored other options. The connectivity graph has only a fraction of the edges that would exist in a completely connected graph, yet maintains the neighborhood relations necessary to cluster the data.

Figure 7.1 illustrates the LCC process on 1000 points in 2D drawn from four Gaussian distributions. A Proximity Forest is constructed on the input data, and the connectivity graph is created. The middle image in the figure shows the connectivity graph filtered such that edges are shown

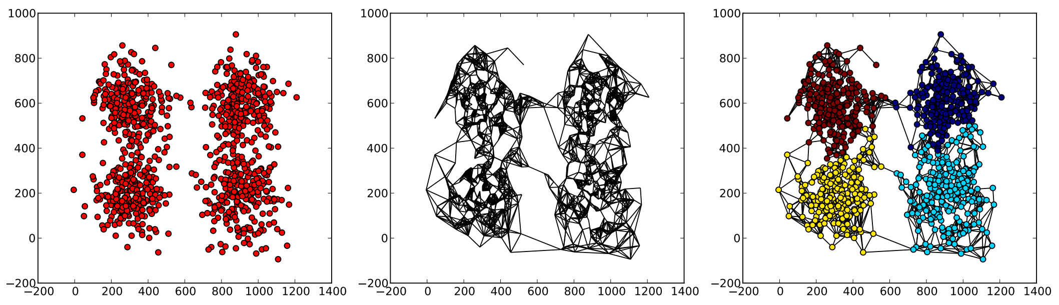


Figure 7.1: LCC stages (from left to right): input data, Proximity Forest connectivity, graph-based clustering.

only between points that co-occur in at least $1/3$ of the trees in the forest. To be clear, this filtering is only for visualization, and is not part of the cluster process. The last image in the figure overlays the points with colors based on cluster label assignment. In this case, hierarchical agglomerative clustering using Ward's Linkage was applied to the full connectivity graph, and the resulting dendrogram was cut to produce four clusters.

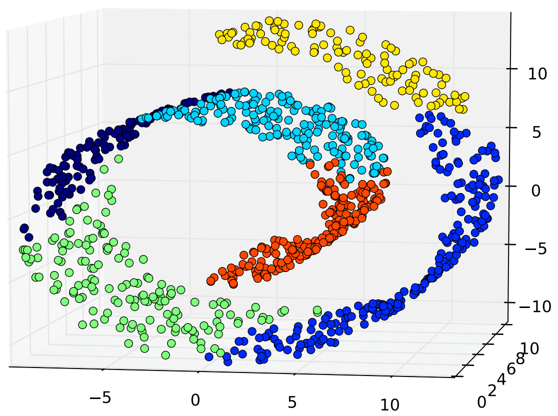


Figure 7.2: LCC applied to Swiss Roll data. As a connectivity-oriented cluster mechanism, Latent Configuration Clustering respects the manifold surface of the data.

Figure 7.2 illustrates clustering performance on the "swiss roll" data. The purpose of this figure is to show that as a connectivity-oriented cluster mechanism, Latent Configuration Clustering respects the manifold surface of the data.

7.2.3 Exemplar Selection

While not required in all applications, it can be convenient to select a representative example, or exemplar, from a cluster. Since LCC assumptions admit no direct computation of the mean of a set of data elements, we must select a medoid instead. This is accomplished via selecting the node with the highest closeness centrality [Sab66] from those in the subgraph of the cluster.

The weighted graph $G(V, E)$, represents the connectivity of the leaves in the forest structure, as described previously. For each cluster C_k, we generate the subgraph $G_k(V_k, E_k) \subset G(V, E)$ consisting of all nodes in C_k and the edges between them. The edge weights, $W_{i,j}$, represent similarity between nodes i and j. We select exemplars, Γ_k, from the subgraphs by computing the closeness centrality, and selecting those with the highest scores. We represent the closeness centrality score of node v in cluster k as $cc_k(v)$, which is computed as follows.

$$
\begin{aligned}
G_k(V_k, E_k) &\subset G(V, E), \quad \text{for cluster } k \\
cc_k(v) &= \sum_{n \in V_k \setminus v} 2^{-d_{G_k}(v,n)} \\
\Gamma_k &= \arg\max_{v \in V_k}(cc_k(v)) \\
\text{where} \quad d_{G_k}(v, n) &= W_{max} - W_{v,n} \\
\text{and} \quad W_{max} &= \max(W_{i,j} \in G)
\end{aligned}
\tag{7.2}
$$

7.3 Application to Discovering Actions in Video

We use a Subspace Forest, as described in Chapter 6, as the Proximity Forest structure in the LCC algorithm when used for clustering actions. We evaluate Latent Configuration Clustering using three data sets, KTH, Gestures, and Tower.

For KTH and Gestures, we use a data cube size of 32 frames at 20×20 pixels for each sample, in agreement with what was done in earlier chapters. For the Tower data set, we use a cube size of 48 frames at 32×32 pixels, because the videos are in lower resolution (so we try to capture more pixels) and some actions take longer to represent (so we extend the temporal duration). For all three data sets, we use a forest of 27 trees with $\tau = 21$. These choices for the Subspace Forest are in agreement with Chapter 6.

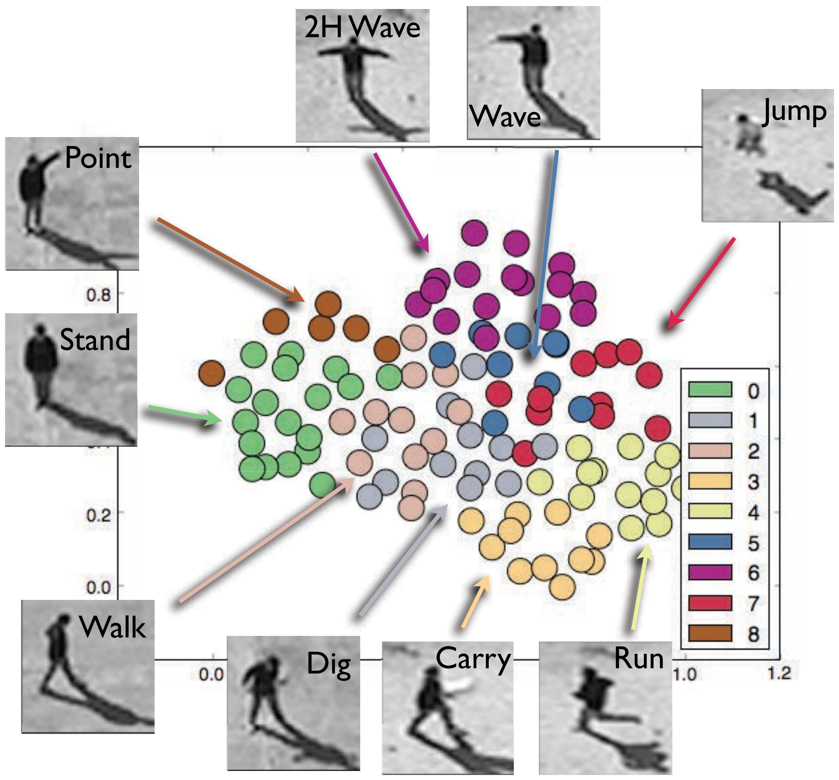


Figure 7.3: Unsupervised exemplar selection on UT Tower data. Color indicates cluster membership. A single exemplar from each class was selected when the number of clusters was limited to the number of classes in the data.

7.3.1 Action Clustering and Unsupervised Exemplar Selection

Figures 7.3, 7.4, and 7.5, illustrate the clusters and associated exemplars on each of the three data sets. LCC perfectly selects an exemplar from each class in the Tower data, even with some apparent cluster overlap. Our method also perfectly generates a single exemplar from each class in KTH, and the classes show minimal cluster overlap. Gestures clusters well, but not perfectly. Of the nine classes, no exemplar for PR (Pair of fingers, Right) was found. Instead a second exemplar for FC (Flat hand, Contracting) was selected.

It is not surprising that Gestures proved the most challenging to cluster, as even supervised classification schemes find the data set relatively challenging in comparison to others. State of the art

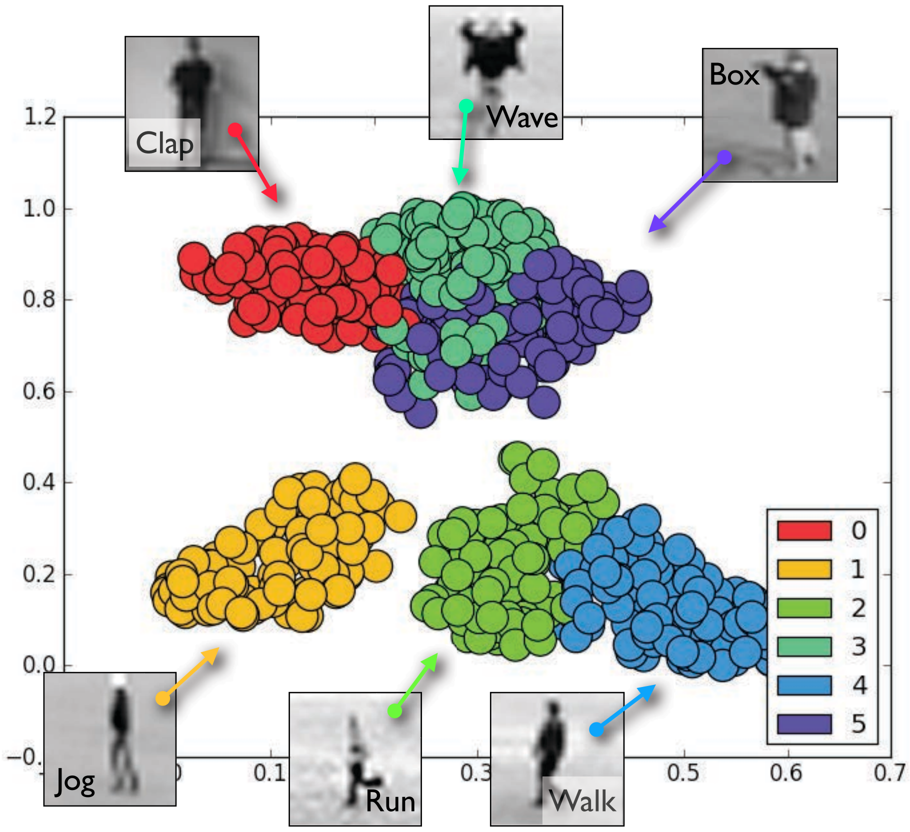


Figure 7.4: Unsupervised exemplar selection on KTH Action data. A single exemplar from each class was selected when the number of clusters was limited to the number of classes in the data.

supervised classification accuracy on KTH is in the upper 90's [LBK10, GIK10], UT Tower is also in the upper 90's [CRA10], while Gestures remains in the upper 80's to lower 90's [LB11]. Each of these data sets defines its own classification protocol and training/testing partitioning, yet the clustering results shown here are indicative that KTH and UT Tower may be intrinsically separable using a Grassmann data representation, while Cambridge Gestures remains more challenging.

7.3.2 Exemplar-based Classification

In order to quantitatively gauge the quality of LCC clustering and exemplar selection, we conducted an evaluation to determine how well we could classify actions with only a few exemplars selected from the training corpus of the KTH Actions data set. We chose KTH Actions because, to the best of our knowledge, it is the only well-known data set upon which unsupervised classification results have been reported, as by Niebles et al. [NWF08].

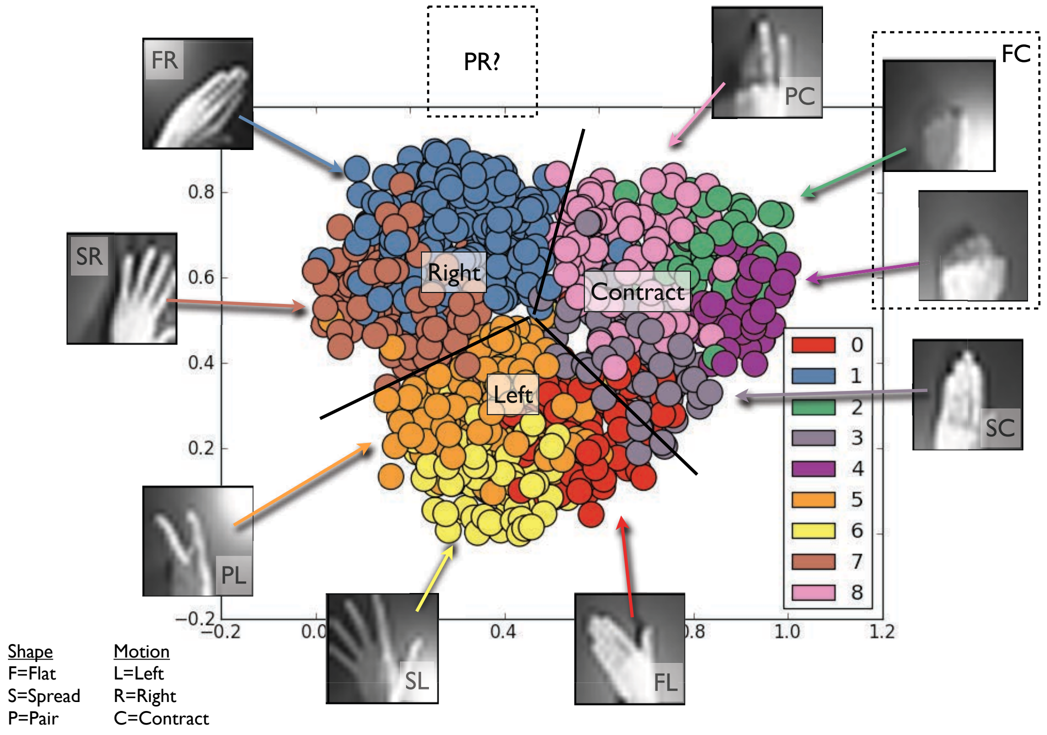


Figure 7.5: Unsupervised exemplar selection on Cambridge Gestures data. A single exemplar from most classes was selected when the number of clusters was limited to the number of classes in the data. No exemplar was automatically selected for the “Pair of fingers, Right” (PR) gesture. Instead two were chosen showing the “Flat hand, Contracting” (FC) gesture.

We apply Latent Configuration Clustering to the KTH training partition of 384 samples, using Schuldt et al.’s recommended partitioning [SLC04]. We used Proximity Forests of three different sizes, 18 trees, 27 trees, and 40 trees. We select from 6 to 24 exemplars from the training data, allowing for 1 to 4 exemplars per class. We use the exemplars to classify the samples from the testing partition (216 samples) using a nearest-exemplar strategy. The label for the nearest exemplar is used to predict the label of the test sample. We repeat each trial 10 times and report the average results.

Classification accuracy is reported in Figure 7.6. When using only 4 exemplars of each cluster (24 total exemplars), classification accuracy approaches 90%, which compares favorably to Niebles et al. [NWF08] who use Probabilistic Latent Semantic Analysis (pLSA) with a bag of words representation to achieve 83.3% unsupervised classification accuracy. Even with only 2 exemplars per cluster (12 total exemplars), performance still compares favorably to pLSA.

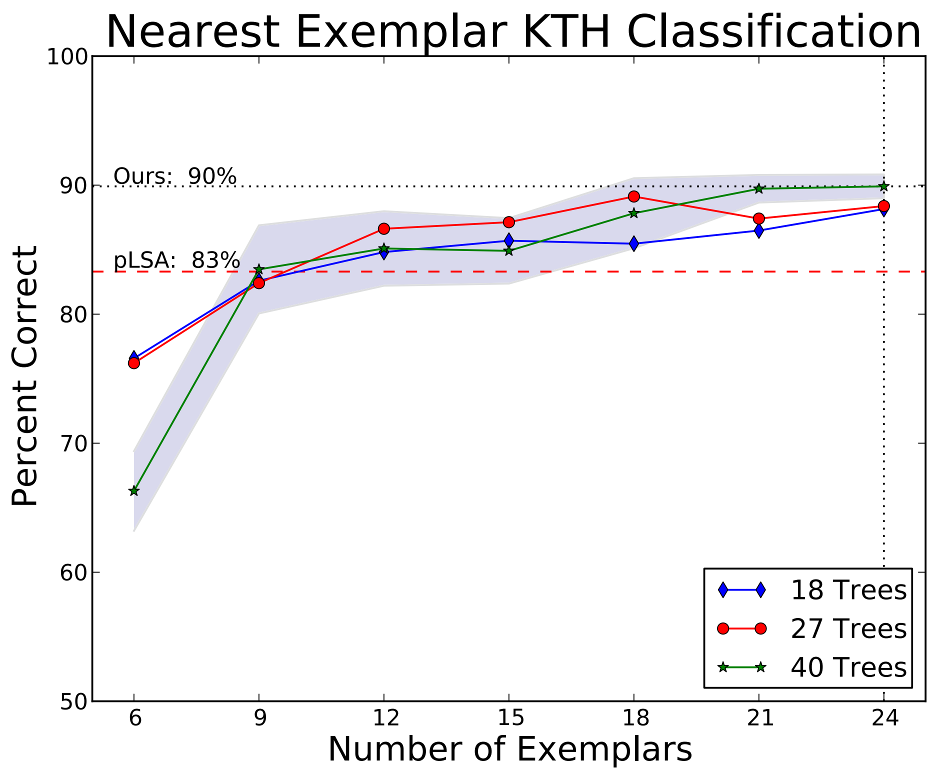


Figure 7.6: Classifying KTH Actions using only a few exemplars per cluster. The x-axis shows the number of exemplars, and the y-axis the nearest-exemplar classification accuracy. Each point represents the average over 10 trials. The blue region indicates the 95% confidence interval around the sample means for the forest of 40 trees. The graph also shows a comparison to the pLSA method from Niebles et al. [NWF08]. Even with only two exemplars per class (12 total), average performance is comparable or superior to pLSA.

In the figure, the shaded region represents the 95% confidence interval around the results for the 40-tree forest. This provides an indicator of the statistical significance between the three forest sizes. There is little difference in significant performance based on the three forest sizes, except for when using only 6 exemplars or when using 21 or more. One would expect a larger forest may produce better results, but it is unclear to us why the largest forest seems to perform much worse when only a single exemplar per cluster is selected. This will be the subject of future evaluation.

7.4 Conclusion

We introduced a method called Latent Configuration Clustering to address the need for efficiently clustering arbitrary metric data, especially in the case where pair-wise comparisons are computationally expensive. Given a set of data elements of arbitrary type and a distance function, LCC generates a sparse connectivity graph to which a standard clustering algorithm can be applied. LCC is efficient because it greatly reduces the number of expensive distance computations that is required for clustering, and replaces them with cheap scalar comparisons.

We applied LCC to unsupervised clustering and exemplar selection of action tracklets. We demonstrated that LCC generates qualitatively good clusters on three data sets, and that it selects exemplars representative of the different action classes. Using exemplars automatically selected from the training partition of the KTH Actions data set, we performed unsupervised action recognition that is superior to existing reported results. Specifically, given only two exemplars per class, our method achieves unsupervised classification accuracies around 85%. Given only four exemplars per class, we show 90% accuracy on KTH.

Chapter 8

Putting It All Together

8.1 Mind's Eye

As mentioned in the introductory chapter, a major motivation for the development of the technology outlined in this thesis was our involvement in the DARPA Mind's Eye program [Gel11]. While the Mind's Eye program is intended to address higher-level human behavior in video, action recognition is a significant component of the effort. This chapter qualitatively describes how the proposed unsupervised action learning and recognition methods performed in the context of this program.

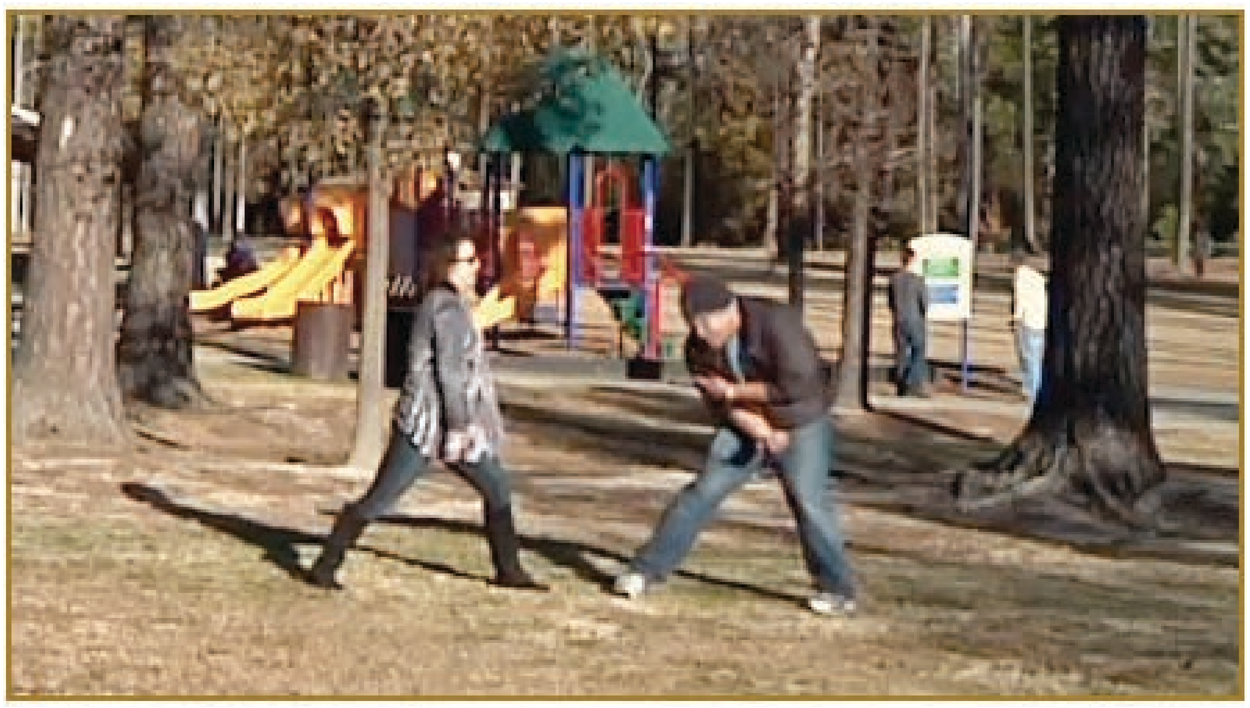


Figure 8.1: Screen capture from a Mind's Eye year 1 video. Unlike the previously discussed benchmark data, Mind's Eye videos feature multiple actors, complex and dynamic backgrounds, and longer durations exhibiting multiple activities.

8.2 Mind's Eye: Year 1 versus Year 2

The first two years of the Mind's Eye program had separate data sets. In the first year, performers were given approximately 3,000 short videos to use for evaluating system performance on four related tasks: 1) verb *recognition* from a set of 48 action verbs, 2) video *description* in 140 characters of text, 3) gap-filling by describing what happens in a section of video that has been blacked-out, and 4) anomaly detection in video. The videos used in the first year for recognition and description are generally short (a minute or less in duration), but contain multiple actors and objects in dynamic settings, including parks, street corners, and parking lots. A screen capture from one of these videos is shown in Figure 8.1.

Many computer vision challenges must be overcome to recognize actions in the Mind's Eye data that are beyond the scope of this thesis. These challenges include detecting and tracking the subjects in the video, under uncontrolled lighting and in dynamic outdoor scenes. The first year data set created additional challenges due to semantic ambiguity and the related issue of high intra-class variability. For example, in the year 1 corpus, some samples for the verb "walk," would best be described as "walking a dog," and "walking on hands."

In the second year of the program, the data was designed to more closely reflect the intended goal of improving automated video surveillance. The year 2 data has approximately one tenth the number of videos, but each is much longer, typically about 15 minutes in length, totalling approximately 4,500 minutes of high-definition video. These videos are designed to simulate realistic operating scenarios, such as monitoring the door to a "safe house," or monitoring a check point on a country road. The videos depict periods of activity as well as long periods where nothing of importance is happening. Unlike in the year 1 corpus, where it was sufficient for the recognition system to label an entire video sample with one or more of the target verbs, in year 2, it was necessary to localize where and when each verb occurs.

In the year 2 challenge, the verbs being recognized were defined with less semantic variation ("walk" means a human walking in a normal upright position), and demonstrated in fewer settings. However, the year 2 videos feature significant variation in camera angle in the data set. Many samples feature actors moving towards or away from a ground-level camera, which induces rapid scale changes in the tracked subjects and frequent occlusions – challenges that were less frequently observed in the year 1 data.

As stated in the introduction, many of the Mind's Eye target verbs are not what we consider actions, but instead require higher level representations that may be built using actions as a component. The action recognition component of our Mind's Eye system learned 66 unique action labels from the unlabeled training corpus, which we present below.

8.3 Learning Actions

During the course of our involvement in the Mind's Eye program, our methods for analyzing the videos evolved, both in response to new capabilities being developed under this research, but also in response to the change in data sets. We employed the action learning and recognition techniques outlined in this thesis on the year 2 data. Because our methods were not fully-developed during the year 1 period of performance, we do not present any results herein on that data set. All further references to Mind's Eye data are to the year 2 corpus.

Tracklets were sampled from the videos using the same technique as outlined in Chapter 4. Tracklets were formed using sliding temporal windows applied over tracked regions in the video (see §4.1.2). Due to the complexity of the videos, the lower level vision system responsible for detecting and tracking the subjects is significantly more complex than that presented on the ETHZ Living Room data set.

A total of 24,800 tracklets were extracted from the training videos, each having dimension 32x32 pixels by 48 frames. A Subspace Forest with 24 trees was constructed to index the tracklets. To speed construction time, one quarter of the forest was built on each of four processing nodes. Total time to build the ANN index was approximately eight hours. Without the Subspace Forest, the estimated time to cluster the tracklets using the PM representation would be 8,500 hours (354 days), without additional parallelism.

Using Latent Configuration Clustering applied to the Subspace Forest, as described in Chapter 7, the tracklets were clustered to produce 2,480 exemplars. Each exemplar was given one or two labels (by human annotation) to describe the action being performed. In some cases the exemplar was not easily described, was ambiguous, or was unidentifiable due to low-level detection or tracking errors. In total 134 of 2480 exemplars were given the label "unknown." The remaining were assigned either one or two labels.

The labeling of exemplars from the Mind's Eye data differed somewhat to that performed in Chapter 4, mainly due to the scale of the challenge. With the ETHZ data, we had a small set of labels learned from only a few hundred training tracklets. At this scale, we could easily construct a bit vector for each sample indicating the presence of each action. With the Mind's Eye corpus, we had 2400 exemplar tracklets to label (reduced from over 24,000 total training tracklets), with no predefined set of labels. In order to make the process of labeling exemplars relatively easy (although it still took many hours to do), we allowed for a maximum of two labels per exemplar.

Although exemplar labeling at this scale still requires significant manual effort, the techniques introduced in this thesis made it possible for one person to do so in one day. With the year 1 corpus, prior to the development of LCC and the exemplar selection strategy, we used about a dozen students over several days time to label approximately 8000 training tracklets. In year 2, we processed three times the number of training tracklets, yet only needed to label 2400 exemplars, which was done by a single person in a single day. Labeling was easier not only because we reduced the number of tracklets, but also because we simplified the label assignment process, as previously discussed.

The set of unique labels discovered from the unsupervised learning process is shown in Table 8.1, which also shows the number of tracklets assigned each label. Since some tracklets received two labels, the sum of the counts is greater than the number of tracklets. There are a total of 66 labels, some of which are related. Labels such as "walk" and "walk-group" differ only in that in the latter case, the tracklet shows two or more people performing the action. The labels of the form "x-motion" indicate that the tracklet was centered not on the full extent of a person's body, but either on a specific body part (leg, e.g.), or other area where there is tracked motion (foliage, shadow, or carried object). The table is provided to show the variety of labels discovered in the videos and the uneven distribution of the label set – two characteristics which are unusual in benchmark data.

8.4 Detecting Actions

Action detection in the Mind's Eye challenge is done using a nearest-exemplar matching strategy similar to that described in Chapter 4 of this thesis. The key differences are: 1) exemplars had a maximum of two labels in the Mind's Eye data, as discussed previously, and 2) nearest exemplar matching was performed using a second Subspace Forest that just indexed the exemplars.

Because over 2000 exemplars were used for action detection in the year 2 corpus, we needed to use a second Subspace Forest to index the exemplars for efficient nearest-exemplar labeling. Thus, the Subspace Forest data structure is used twice in our Mind's Eye action recognition process. Once for clustering the training tracklets to produce the exemplars as part of LCC, and again to index the set of exemplars so that labeling the test data is fast.

As done in Chapter 4, for every tracked subject in the test videos, a sliding temporal window is used to extract the tracklet to be labeled. The label of the nearest exemplar is used to label each test tracklet. The score of the label is proportional to the distance between the test sample and the exemplar, according to distribution statistics collected from the training data. The frames within a tracklet are labeled up to four times, due to overlapping the sliding windows. Thus, at a single point in time (frame number), a tracked entity may have several labels. In addition, as part of the Mind's Eye processing, scores are normalized and smoothed over time by applying a Hidden Markov Model.

Figure 8.2 shows a collage of six sample action detections from the year 2 data. It is worth noting that even though the distribution of labels is heavily weighted towards a few very common actions (walk, carry, turn, loiter), we detect instances of rare actions like collide and greet.

8.5 Integration Test

The Mind's Eye program provided funds not only to research institutions, but also to three corporate technology integrators. One of those was iRobot Corporation, maker of popular home and military robots. Our team partnered with iRobot to port our action recognition technology to their platform.

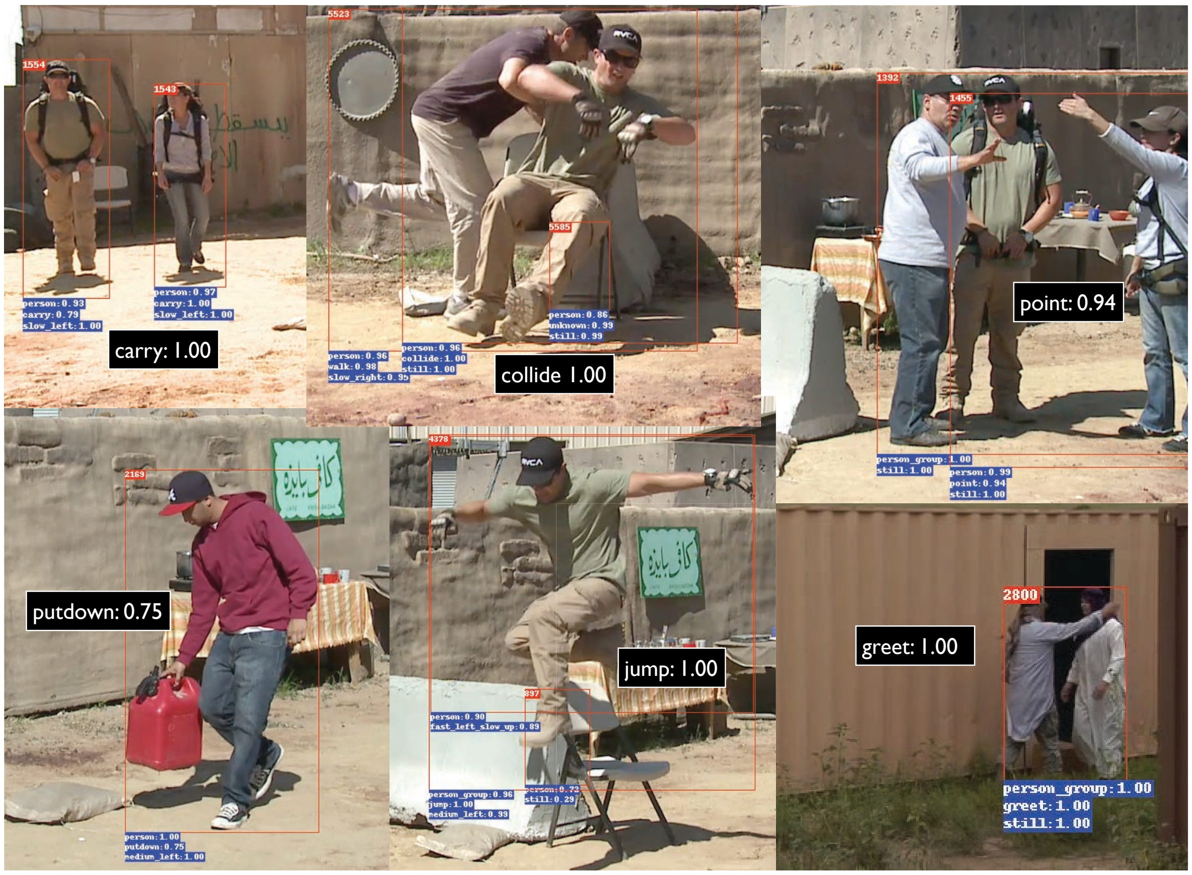


Figure 8.2: Samples of action detections in the Mind's Eye year 2 data. In the figure, clockwise from top left: carry, collide, point, greet, jump, and putdown. The highlighted score indicates the confidence in the label based on nearest neighbor voting from the Subspace Forest.

In June 2012, a team from iRobot demonstrated real-time action recognition to the Mind's Eye program manager and other interested parties using the technology described in this thesis. The iRobot demonstration was the only one able to meet the action detection goals of the integration test. Further, the iRobot team, upon request by the reviewers, was able to incrementally add a new action to the demonstration within a single day, while on-site. This is a significant practical advantage of building a system without a closed-world assumption. Information about the integration test was conveyed in an e-mail from Dr. Christopher Geyer, the Principal Investigator from iRobot, dated June 26, 2012.

Table 8.1: Set of labels learned from Mind's Eye training data.

Label	**Count**	**Label**	**Count**
approach	4	loiter	185
arm-motion	13	loiter-group	117
bend	27	loiter-seated	6
carry	227	no-action	12
cart-motion	9	object-drop	3
chase	6	object-motion	24
clap	11	pass	39
clap-group	41	pass-group	2
climb	1	patdown	7
collide	11	pickup	28
cover	2	point	49
dig	21	point-group	33
dig-group	4	push	25
drag	10	putdown	28
drop	1	ride-bike	17
enter	3	run	24
escort	5	run-group	4
exchange	3	shadow-motion	12
exit	3	shove	1
cloth-flapping	15	sit	8
foliage-motion	28	sit-up	1
follow	32	stand	6
foot-motion	16	stop	21
gesture	39	swing	16
give	4	take	1
greet	9	throw	6
head-motion	77	turn	153
hold-box	3	unknown	134
jump	8	walk	659
kick-ball	19	walk-group	351
lean	1	walk-rifle	7
leave	4	wave	19
leg-motion	141	wave-group	20

Chapter 9

Conclusion

9.1 Summary of Work

This thesis has presented a set of evaluations, algorithms, and data structures that address some of the challenges that stand in the way of being able to recognize human behaviors outside of controlled benchmark data sets. For empirical evaluation, the individual contributions presented in this thesis were tested on controlled data, but the methods were designed out of the need to apply action recognition to more challenging and larger data sets.

In order to learn from a large corpus of unlabeled data, we argued for advancing methods for unsupervised clustering of actions. We performed a comparative analysis of a local-features action representation with one based on manifold geometry and showed that the manifold method may be more amenable to unsupervised learning. We proposed a method for extracting video samples of tracked entities from streaming data, and applying hierarchical clustering on the samples, represented as points on Grassmann manifolds, to derive a set of representative action exemplars. We demonstrated that this method can be used for action recognition based on K-nearest-exemplar matching, and has the advantage of allowing for incremental learning and anomaly detection.

To address the real-time performance requirements of action recognition applications, we developed a scalable data structure, called a Proximity Forest, for indexing general metric data for Approximate Nearest Neighbor (ANN) queries. To evaluate the Proximity Forest, we first compared it to two popular ANN methods, KD-Trees (KDT), and Hierarchical K-Means (HKM), on vector data of interest to the computer vision community. The Proximity Forest was 15 to 20% more accurate than KDT and HKM on a series of studies which controlled for data set size, dimensionality, distance metric, number of neighbors to return, and forest size. In addition to the significant increase in accuracy on the tested data, the Proximity Forest is more general because it can be applied to any metric data, including our manifold representation of actions.

We introduced the Subspace Forest, a variant of a Proximity Forest designed for ANN indexing of fixed-dimensional subspaces (or, equivalently, indexing points on a Grassmann manifold). When used for nearest-neighbor classification of actions, the Subspace Forest yielded state-of-the-art accuracy on three well-known action benchmarks, with the significant advantage of being fast and scalable.

The final contribution of this thesis uses the Proximity Forest indexing structure as a key component of a method for efficiently clustering any metric data. This was the final technical achievement needed in order to address the scalability issues involved in unsupervised learning of actions using our chosen manifold representation. We demonstrated clustering and exemplar selection from three data sets. After post-hoc labeling of the selected exemplars (i.e., labels are required to describe the selected exemplar actions, but not for learning them), we demonstrated unsupervised action recognition on a benchmark data set, exceeding the accuracy of a competing unsupervised method.

9.2 Broader Impact

The methods for scalable action clustering and recognition presented in this thesis were key enablers for our performance in the Mind's Eye program. However, there are contributions of broader interest as well.

The Proximity Forest is a general-purpose data structure for Approximate Nearest Neighbor indexing. Our study shows that it may be more accurate than existing methods popularly used in computer vision applications. While more engineering is required to match the maturity and speed of the implementations of competing methods, there is potential that a "black-box" replacement of the ANN indexing method with a Proximity Forest could substantially improve the accuracy of applications such as large-scale image retrieval.

As we found in action recognition, sometimes a non-vector data representation is beneficial. Anyone wishing to perform large scale similarity search or clustering of such data has far fewer tools to draw from, because most existing methods assume the data resides in a vector space with a global coordinate system. This thesis provides additional tools for those dealing with non-vector metric data elements. The Proximity Forest can be used for ANN indexing of general metric spaces, and Latent Configuration Clustering can be used for efficiently clustering elements of a general metric space.

9.3 Future Work

There are many remaining challenges to be addressed when developing systems to automatically recognize human behaviors in video. These challenges span the range from improving lower-level processing to integrating actions, interactions, and gestures as input to higher-level methods for detecting activities, events, and other complex behaviors.

This thesis focused on methods relating to action recognition. The representation we employed requires that subjects are first detected and tracked by a lower-level video processing module. Accurate human detection and tracking are areas of active research in the computer vision community. There is a concept of a feedback loop between the detection and tracking modules, sometimes phrased as "tracking via detection, and detection via tracking." It may be possible to extend this optimization loop to include the characteristic motion patterns of human subjects. The idea would be to correct the localization of the subject by finding perturbations of the tracked region that yield better similarity scores to known actions.

Along this line of thought, one might be able to extend the action representation from 3rd order tensors to 4th order, representing a *set* of (x,y,t) data cubes. The 4th order representation might be able to account for small spatial or temporal localization errors by extracting a set of tracklets around a given space-time volume, each element of the set generated by one of a fixed-number of transformations. Actions would then be represented as a set of candidate volumes around the subject, and action recognition becomes a set-to-set matching problem on Grassmann manifolds.

In the introductory chapter, we alluded to the fact that interactions and gestures may be represented in a similar fashion to actions. Chapter 3 compared the Product Manifold representation to a Bag of Features representation on actions, hand gestures, and facial expressions. However, this thesis did not explore the application of the proposed methods much beyond action recognition. Given a reliable method for detecting and localizing interactions between two or more people, a Subspace Forest may work as well for interactions as it does for actions.

We showed that a Proximity Forest provides accurate Approximate Nearest Neighbor indexing of vector data, as well as being generalizable to non-Euclidean metric data. Before the wider community would embrace a potential replacement for the popular FLANN library [ML09], more engineering is required to generate an implementation with comparable indexing speed. Speed improvements would likely arise by using a compiled language (C/C++ instead of Python), multi-threading, and employing vectorized operations when the input data is in matrix form.

Finally, more work needs to be done for learning higher-level behaviors by integrating action, interaction, and gestures with additional information about the subjects and objects of interest, as well as the context or setting of the activity. For the Mind's Eye project, we combined this information primarily using intuition and heuristics. In the future, more principled ways of learning higher-level concepts from low and mid-level information may be investigated.

References

[AMN$^+$98] S. Arya, D. M Mount, N. S Netanyahu, R. Silverman, and A. Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[AR11] J. K. Aggarwal and M. S. Ryoo. Human activity analysis: A review. *ACM Computing Surveys (CSUR)*, 43(3):16, 2011.

[ARS09] M. Andriluka, S. Roth, and B. Schiele. Pictorial structures revisited: People detection and articulated pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1014–1021, 2009.

[BHZM07] R. Basri, T. Hassner, and L. Zelnik-Manor. Approximate nearest subspace search with applications to pattern recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[BK08] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Incorporated, 2008.

[BKR10] U. Bonde, T. K Kim, and K. Ramakrishnan. Randomised manifold forests for principal angle-based face recognition. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 228–242, 2010.

[BL97] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1000–1006. IEEE, 1997.

[BLGX10] M. Bregonzio, J. Li, S. Gong, and T. Xiang. Discriminative topics modelling for action feature selection and recognition. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 8.1–8.11, 2010.

[BS02] Mukund Balasubramanian and Eric L Schwartz. The isomap algorithm and topological stability. *Science*, 295(5552):7–7, January 2002.

[CNBYM01] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, 2001.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1997.

[CRA10] Chia-Chih Chen, M. S. Ryoo, and J. K. Aggarwal. UT-Tower Dataset: Aerial View Activity Classification Challenge. http://cvrc.ece.utexas.edu/SDHA2010/Aerial_View_Activity.html, 2010.

[DRCB05] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *Proceedings of the Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (PETS)*, pages 65–72, 2005.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[FBF77] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[FHT09] Jerome H. Friedman, Trevor Hastie, and Robert Tibshirani. *Elements of Statistical Learning*. Springer Series in Statistics, second edition, 2009.

[GBS$^+$07] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2247–2253, 2007.

[Gel11] T. Geller. Seeing is not enough. *Communications of the ACM*, 54(10):15–16, 2011.

[GIB09] A. Gilbert, J. Illingworth, and R. Bowden. Fast realistic multi-action recognition using mined dense spatio-temporal features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 925–931, 2009.

[GIK10] K. Guo, P. Ishwar, and J. Konrad. Action recognition using sparse representation on covariance manifolds of optical flow. In *Proceedings of the IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, pages 188–195, 2010.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 518–529, 1999.

[HN03] X. He and P. Niyogi. Locality preserving projections. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, pages 153–160, 2003.

[JKJ$^+$11] A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. A category-level 3-d object dataset: Putting the kinect to work. In *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1168–1174, 2011.

[JT05] F. Jurie and B. Triggs. Creating efficient codebooks for visual recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 604–610, 2005.

[Kar77] H. Karcher. Riemannian center of mass and mollifier smoothing. *Communications on Pure and Applied Mathematics*, 30(5):509–541, 1977.

[KG10] A. Kovashka and K. Grauman. Learning a hierarchy of discriminative Space-Time neighborhood features for human action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2046–2053, 2010.

[KSH05] Y. Ke, R. Sukthankar, and M. Hebert. Efficient visual event detection using volumetric features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 166–173, 2005.

[KWC07] T. K. Kim, S. F. Wong, and R. Cipolla. Tensor canonical correlation analysis for action classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[Lap05] I. Laptev. On space-time interest points. *International Journal of Computer Vision*, 64(2):107–123, 2005.

[LB11] Y. M. Lui and J. R. Beveridge. Tangent bundle for human action recognition. In *Proceedings of the IEEE Conference on Automatic Face and Gesture Recognition (FG)*, pages 97–102, 2011.

[LBK10] Y. M. Lui, J. R. Beveridge, and M. Kirby. Action classification on product manifolds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 833–839, 2010.

[LJD09] Z. Lin, Z. Jiang, and L. S Davis. Recognizing actions by shape-motion prototype trees. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 444–451, 2009.

[LMGY04] T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, pages 825–832, 2004.

[Low99] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1150–1157, 1999.

[Low04] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[LP07] I. Laptev and P. Perez. Retrieving actions in movies. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1–8, 2007.

[McQ66] L. L. McQuitty. Similarity Analysis by Reciprocal Pairs for Discrete and Continuous Data. *Educational and Psychological Measurement*, 26(4):825–831, December 1966.

[MCUP04] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, 2004.

[ML09] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of the International Conference on Computer Vision Theory and Application (VISSAPP)*, pages 331–340, 2009.

[MTJ06] F. Moosmann, W. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, pages 985–992, 2006.

[MTS$^+$05] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1):43–72, 2005.

[NGV10] F. Nater, H. Grabner, and L. Van Gool. Exploiting simple hierarchies for unsupervised human behavior analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2014–2021, 2010.

[NS06] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2161–2168, 2006.

[NWF08] J. C. Niebles, H. Wang, and L. Fei Fei. Unsupervised learning of human action categories using spatial-temporal words. *International Journal of Computer Vision*, 79(3):299–318, 2008.

[OD11a] Stephen O'Hara and Bruce A Draper. Introduction to the bag of features paradigm for image classification and retrieval. *arXiv:1101.3354*, January 2011.

[OD11b] Stephen O'Hara and Bruce A. Draper. Unsupervised learning of Micro-Action exemplars using a product manifold. In *Proceedings of the IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, pages 206–211, 2011.

[OD12] Stephen O'Hara and Bruce A. Draper. Scalable action recognition using a subspace forest. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1210–1217, 2012.

[OD13] Stephen O'Hara and Bruce A. Draper. Are you using the right approximate nearest neighbor algorithm? In *Proceedings of the IEEE Workshop on the Applications of Computer Vision (WACV)*, pages 9–14, 2013.

[O'H12] Stephen O'Hara. ProximityForest on SourceForge.net. `http://sourceforge.net/projects/proximityforest/`, 2012.

[OLD11] Stephen O'Hara, Yui Man Lui, and Bruce A. Draper. Unsupervised learning of human expressions, gestures, and actions. In *Proceedings of the IEEE Conference on Automatic Face and Gesture Recognition (FG)*, pages 1–8, 2011.

[OLD12] Stephen O'Hara, Yui Man Lui, and Bruce A. Draper. Using a product manifold distance for unsupervised action recognition. *Image and Vision Computing*, 30(3):206–216, March 2012.

[PCI$^+$07] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[PDC09] N. Pinto, J. J. DiCarlo, and D. Cox. How far can you get with a modern face recognition test set using only simple features? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2591–2598, 2009.

[RAK09] K. Rapantzikos, Y. Avrithis, and S. Kollias. Dense saliency-based spatiotemporal feature points for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1454–1461, 2009.

[RAS08] M. D. Rodriguez, J. Ahmed, and M. Shah. Action MACH a spatio-temporal maximum average correlation height filter for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[RCAR10] M. S. Ryoo, C. C. Chen, J. K. Aggarwal, and A. Roy-Chowdhury. An overview of contest on semantic description of human activities (SDHA) 2010. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 270–285, 2010.

[RS00] S. T Roweis and L. K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[Sab66] Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

[SAS07] P. Scovanner, S. Ali, and M. Shah. A 3-dimensional sift descriptor and its application to action recognition. In *Proceedings of the International Conference on Multimedia (MM)*, pages 357–360, 2007.

[SC12] S. Sadanand and J. J. Corso. Action bank: A high-level representation of activity in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1234–1241, June 2012.

[SFC$^+$11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 1297–1304, 2011.

[SH08] C. Silpa-Anan and R. Hartley. Optimised KD-trees for fast image descriptor matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[SLC04] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: A local SVM approach. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 32–36, 2004.

[SZ03] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1470–1477, 2003.

[TC10] P. Turaga and R. Chellappa. Nearest-neighbor search algorithms on non-Euclidean manifolds for computer vision applications. In *Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing*, pages 282–289, 2010.

[TSL00] J. B Tenenbaum, V. Silva, and J. C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[Uhl91] Jeffrey K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, November 1991.

[Uni12] University of Maryland Language and Media Processing Laboratory. ViPER-GT. `http://viper-toolkit.sourceforge.net/products/gt/`, 2012.

[Vai89] P. M Vaidya. An O(NlogN) algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4(1):101–115, 1989.

[VF10] A. Vedaldi and B. Fulkerson. VLFeat: an open and portable library of computer vision algorithms. In *Proceedings of the International Conference on Multimedia (MM)*, pages 1469–1472, 2010.

[War63] Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[WLZY11] X. Wang, Z. Li, L. Zhang, and J. Yuan. Grassmann hashing for approximate nearest neighbor search in high dimensional space. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2011.

[Wol01] David H. Wolpert. The supervised learning no-free-lunch theorems. In *Proceedings of the Online World Conference on Soft Computing in Industrial Applications*, pages 10–24, 2001.

[WUK$^+$09] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid. Evaluation of local spatio-temporal features for action recognition. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 124.1–124.11, 2009.

[WXDL11] X. Wu, D. Xu, L. Duan, and J. Luo. Action recognition using context and appearance distribution features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 489–496, 2011.

[YGV10] A. Yao, J. Gall, and L. Van Gool. A hough transform-based voting framework for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2061–2068, 2010.

[Yia93] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.

[YKC10] T. H. Yu, T. K. Kim, and R. Cipolla. Real-time action recognition by spatiotemporal semantic and structural forests. In *Proceedings of the British Machine Vision Conference (BMVC)*, page 56, 2010.