Carnegie Mellon University

Robotics Institute

Thesis Proposal

Doctor of Philosophy

Title:

Generating Representations for Action Recognition From Coarsely Labeled and Synthetic Data

Date of Submission: April 26, 2011

Submitted by: Pyry Matikainen

Committee Members: Professor Martial Hebert (co-chair)
Robotics Institute

Professor Rahul Sukthankar (co-chair)
Robotics Institute, Intel Research Pittsburgh

Yaser Sheikh
Robotics Institute

Ivan Laptev
INRIA

Contents

Abstract

Action recognition techniques rely heavily on well chosen features, such as trajectory-based motion descriptors, to make the most of relatively scarce video training data. Typically these features must be hand-selected because the very paucity of suitably annotated data that makes the selection of features critical also restricts the degree to which those features can be directly learned. However, low-quality coarsely annotated data is readily available in the form of tagged videos on websites such as YouTube.com, and public motion capture databases along with simple graphics techniques make possible the generation of vast amounts of plausible (in terms of motion) synthetic videos of human actions. The difficulty lies in taking advantage of these types of data: YouTube clips are coarsely and inconsistently annotated, while synthetic data differs from real data in many way, both overt and subtle. These data sources should not be considered as substitute data for any *particular* action recognition task, but instead as sources of *related* tasks in the broader domain of action recognition. This proposal seeks to take advantage of these plentiful related action recognition tasks to improve performance on a target task, by using them to select or generate good feature representations.

While many approaches have attempted to learn or select 'generally good' features that perform well when shared across many tasks, we propose instead to use collaborative filtering techniques to use these related tasks to recommend features tailored specifically for the target task. These recommendations are made by evaluating, or rating, a small subset of features on the target task, and then using that small set of ratings along with the ratings of features on a large set of synthetic or coarsely labeled real tasks to predict the ratings of the unevaluated features on the target task. Additionally, we propose a layered bag-of-words representation that enables us to exploit the detailed annotations (pixel-level body part labels) available in synthetic human action data.

1 Introduction

A human researcher who designs a vision algorithm has an almost insurmountable advantage over a learning algorithm: they can appeal to an intuition built from direct experience with the world to decide which parts of the visual experience are important to consider and which are distractions. Is it any surprise then that progress in action recognition has largely been driven by human engineered features rather than by unconstrained machine learning?

That is to say, while in principle even simple learning algorithms such k-nearest-neighbor can learn to solve arbitrary problems, to do so they may need impractical amounts of data. Without prior knowledge of the underlying structure of a problem, learning algorithms are reduced to memorizing the answers to previous inputs, which requires spanning the space of inputs. In the case of images or videos, spanning the space of inputs with any density requires an almost unimaginably vast number of samples.

The key to solving this dilemma lies in *features*: a researcher might look at an action classification task (by task we mean the combination of a goal, such as "classify videos as walking or not walking" and a dataset, such as UCF-YT) and decide that the *motion* is important, and then design a motion-based features to *describe* the motion content of a video. Likewise, a researcher in scene understanding may conclude that it is the statistical distribution of edges that matters, and thereby produce the GIST descriptor. Features, whether they be histograms of motions in a scene or statistical descriptions of edges, transform the high-dimensional input into a more manageable lower dimensional one, concentrating and revealing the important information of the input in the process.

This proposal does not seek to introduce yet another feature. Instead, the heart of this proposal is to answer the question "how can we decide which features are useful for a particular task"? Human researchers appeal to their intuition and experience when deciding which features to apply to a given task: they may look at an action classification task with significant camera motion and decide that motion based features are likely to be confused, or they may look at a new image classification task and decide to use SIFT features, simply because in their experience SIFT features have always performed reasonably well. Or consider a related scenario: a researcher is faced with a new video understanding task, so they try the set of features they are most familiar with (*e.g.*, HOG, SIFT,

STIP, *etc.*), and then based on how well those reference or *probe* features perform, decide which new features to apply to the problem. Hypothetically, the researcher might find that the image-based features (HOG, SIFT) perform poorly, and thus decide to invest their effort in motion-based features (trajectories, motion-image histograms,*etc.*). As a third scenario, a researcher faced with an outdoor/indoor scene classification task may decide to use a per-pixel grass detector (trained from a different dataset) as a feature that evaluates how "grassy" a scene is by counting the pixels classified as grass.

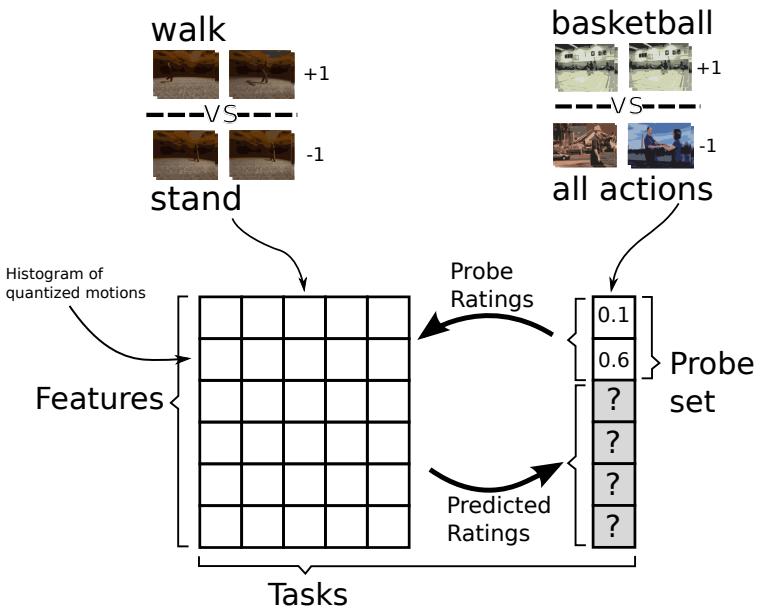


Figure 1: The performance of a small set of features (the probe set) is evaluated (rated) on a target task, and collaborative filtering is used to predict the ratings of other features on the target task, using a database (store) of how features performed (were rated) on other tasks.

In the first scenario, a researcher uses their experience with features to decide which features to use on a new task. In the second, the researcher experimentally evaluates how well some features perform, and then uses the performance of those features along with experience to decide which features to apply. In the third, the researcher applies features trained for a task with more detailed annotations (the grass classifier trained on data with per-pixel labels) to a task with coarser annotation (the indoor/outdoor task which only has per-image labels).

The goal of this proposal is to automate these processes, to build a kind of "algorithmic intuition"

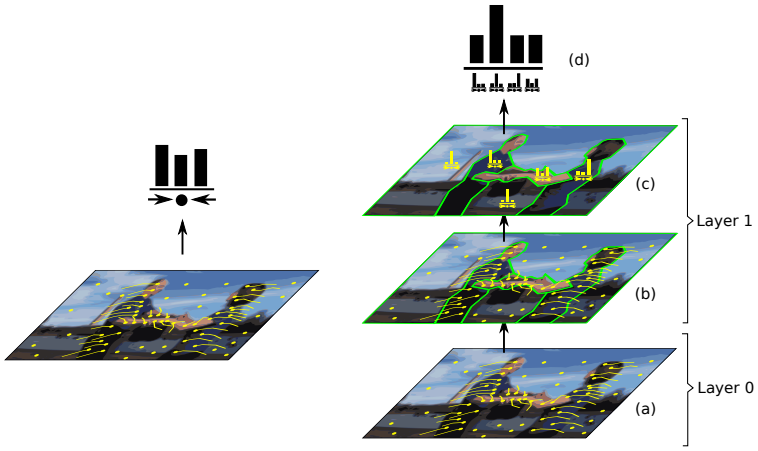


Figure 2: Layered bag-of-words builds upon the popular bag-of-words family of techniques. On the left, a simple bag-of-words technique quantizes descriptors (motions in this case) in a video and computes a histogram. On the right, layered bag-of-words (LBOW) adds an intermediate layer, in which first the base descriptors (a) are segregated into a number of groups (b), and then each group becomes a new descriptor whose value is a histogram (c). In the last stage (d), a standard bag-of-words approach is applied to these new descriptors, so that the result is a histogram of quantized histograms.

that will recommend features for a particular task based on the performance of a small subset of features, rather than having to directly evaluate the entire space of available features. Additionally, our goal is to take advantage of more detailed annotations available for some tasks (*e.g.*, per-pixel labels in some datasets, per-limb labels in human action datasets) to build or recommend features for others. We propose to use collaborative filtering techniques to perform the former (Fig. 1), and to use a layered bag-of-words framework for the latter (Fig. 2). In addition, we seek to expand on these processes, so that rather than recommending features from a small pool of named algorithms, we might make recommendations from a pool of potentially hundreds of thousands or even millions of *randomly generated* features, using the performance of those features on a set of thousands of different tasks to guide the recommendation process.

1.1 Overview

The proposed work is divided into two main parts,

- **Collaborative filtering for feature recommendation** (Fig. 1): Given some formalized notion of how well a feature performs on a task (the feature's *rating* on that task), we plan to use collaborative filtering techniques to recommend features for a particular action recognition or vision task, by predicting the ratings of the unrated features on that task using the ratings of a small probe set of features.

  To make these predictions through collaborative filtering, we will need to have evaluated the features on a large set of tasks. Since there are relatively few action recognition datasets, we consider using synthetic data to create the ratings database, or ratings *store*. Alternatively, the ratings store can be populated using real-data tasks from poorly-labeled video clips (*e.g.*, predict tags for videos on YouTube). One strength of this collaborative filtering approach is that both synthetic and real tasks can be readily included in the ratings store.

  If features are independently recommended, then there is a very close parallel to typical recommender systems, and collaborative filtering techniques can be used with only minor adaptations. However, there is the danger that such an approach may select a large number of redundant features. We will investigate the problem of *jointly* recommending a *set* of features.

- **Layered bag of words** (Fig. 2): In the second part, we propose to add hierarchy to collections of features by successive layers of aggregation and description. That is, in an aggregation layer, current features are merged, or aggregated, into collections. In the following description layer, each aggregation is reduced to a fixed-size descriptor, or described. In this way, the combined effect of an aggregation followed by a description is to transform a set of sparse features into a new, reduced set of sparse features. For example, an aggregation rule might group together nearby trajectory descriptors which appear to describe rotational motion about the same point, and in this way produce another level of descriptors which represent rotating groups of trajectories.

  Additionally, the use of layered representations allows us to exploit the finer detail annotation in synthetic data, by using those finer annotations to guide the selection of the middle layers of the structures. For example, synthetic data labels might be used to learn which types of motions are 'arm-like', so that in a grouping layer nearby 'arm-like' motion descriptors are grouped together into new sparse descriptors that represent 'arm-like' groups of motions.

1.2 Contributions

The expected contributions of this proposal are methodological, rather than theoretical. That is to say, while theoretical contributions will certainly arise in the exploration of the topics of this proposal, the primary contribution is to demonstrate how large numbers of heterogeneous tasks might be combined to build an algorithmic intuition for action recognition, and perhaps more general vision problems. We identify three main contributions:

- **Scaling to large numbers features and tasks**: The *scale* of our approach distinguishes it from the bulk of transfer learning, as we propose to take advantage of thousands of tasks (synthetic *and* real) and hundreds of thousands of candidate features. In comparison, even the random screening approach of [1] only considers on the order of thousands of models.

  The focus on collaborative filtering highlights approaches that can scale to millions of features and thousands of datasets. In contrast to other techniques (*e.g.*, multi-task learning), our proposed feature recommendation paradigm allows features to be recommended very quickly, even from a vast pool. Unlike multi-task learning, where the introduction of badly-matched tasks easily degrades performance [2, 3], there is little penalty, and great potential gain, to expanding the corpus of source datasets.

- **Handling heterogeneous features and tasks**: Our collaborative approach is distinguished not only by its scale, but also by its diversity. By concentrating on the *performance* (rating) of features, rather than their *content*, we can easily incorporate a large variety of features. Furthermore, since features are treated as opaque functions, new features can be incorporated into the method without any modification at all.

  Likewise, tasks and datasets of wildly varying content can be included, provided that there is still a sensible way to rate the features. We are able to take advantage of plentiful synthetic data without having to carefully model the specific target task. Collaborative filtering techniques were developed to deal with missing ratings, and so it is even possible to mix multi-modal datasets, where some features are only applicable on some datasets.

  Our layered bag-of-words approach also handles heterogeneous features well, as the layered representations we propose are to be built on generic video or image descriptors.

- **Exploiting variation in annotation detail**:

  Rather than considering only lowest-common-denominator annotations between different datasets, we take advantage of these differences in annotation detail. For example, some datasets (*e.g.*, those produced synthetically) may have very detailed annotations, such as per-pixel labels or detailed limb positions for human actions, while others have only coarse annotations only at the level of per-clip action labels. We hope to exploit these differences using our layered bag-of-words technique, by using the more detailed annotations to guide the selection or generation of the middle layers of our proposed layered representations. The later layers of these representations can then be learned on the more coarsely annotated data.

1.3 Organization

We explore two main ideas, both related to how to select and represent features. This proposal is organized into three parts,

- Problem setting and general notation: We describe the general setting of the problems we explore (action recognition), and describe common notation. This appears in Section 3.1.
- Collaborative filtering for feature recommendation: We propose to use collaborative filtering techniques in order to recommend features that are likely to perform well on a new task, given the past performance of a candidate pool of features across a large number of diverse tasks (Section 4).
- Layered bag-of-words: We propose to build layered structures on top of features commonly used for bag of words techniques, and to take advantage of detailed annotations available in synthetic data to help build the intermediate layers of these representations (Section 5).

2 Literature Review

2.1 Motion Descriptors

Perhaps the most intuitive representation of motion with a video is optical flow, which has been a very common representation of motion [4, 5, 6, 7]. Despite its intuitive appeal, the actual calculation

of optical flow from sequential frames is itself a difficult problem and even the best algorithms are often afflicted by artifacts and noise. In an attempt to sidestep this problem, many techniques avoid the actual calculation of optical flow, either by implicitly encoding motion through temporal derivatives [8, 9, 10], or by producing the information that would be required to compute optical flow but refraining from the final step [7]. Motion consistency with a template video has frequently been used in order to detect [11, 12, 13, 14] or classify [15, 16] actions in video.

Since most applications are interested in classifying the motion of actors distinct from their backgrounds, an intuitive and frequently used approach is to consider the evolution of the action's silhouette over time [17, 18, 6, 19]. However, silhouettes cannot represent motion that occurs within the silhouette boundary (like a person clapping with both hands in front of her body), so some techniques have considered superpixel regions that can capture internal structure as well [20, 21].

As foreground-background segmentation is a difficult problem in its own right, other approaches detect interest points of various flavors and compute descriptors around them [10, 17, 4, 22].

When these locations extend in time they become trajectories, most frequently arising from tracked features. Often it is assumed that these trajectories are from known, fixed points, such as particular landmarks (*e.g.*, elbows, hands) on a human body [23, 24, 25]. If the trajectories are not on known features, then if they are very long and robust, it is potentially possible to extract full 3D information even from a single view [26]. Other techniques treat trajectories as another type of interest point and make effort to explicitly match trajectories to body parts or landmarks [27, 28, 29].

2.2 Bag-of-words Techniques

Perhaps the most well-known bag-of-words technique is that of textons, or quantized filter bank responses, which originated with texture classification but quickly grew to be applied to object and scene recognition in static images [30, 31]. By analogy to the idea that textual works are composed from a set of discrete repeating elements (words), techniques that model data as a histogram quantized features are generally known as bags-of-words techniques. These approaches are backed by some degree of psychological research suggesting that even human vision may employ unstructured statistical approaches at early stages [32], and can scale well even to large datasets

when used with sparse features [33]. These techniques often outperform more structured approaches (such as parts based models), especially on difficult datasets [34].

Recently bag of words techniques have gained significant popularity in video interpretation [4, 35, 27, 22]. In many cases, the same features that work for static images still apply to video (e.g., histograms of oriented gradients, filter responses) [22].

2.3 Relationships and Hierarchies

Pairwise spatial relationships, in the form of star topologies, fans, constellations, and parts models, have seen frequent use in static image analysis [36, 37, 38]. Practical limitations have made transitioning these methods to video difficult. Sparse pairwise topologies (stars, fans, parts) often suffer from a lack of appropriately annotated training data, as they often require annotations that specify the topology for training [39, 6]. Alternatively, there are structured methods which can operate without such annotations, but at the cost of significantly more complicated and computationally expensive training or testing [40, 25]. In the special limited case of a fixed camera, the entire topology can be fixed relative to the frame by simply using the absolute positions of features [41, 42]. When image regions are considered, often Markov random fields (MRFs) or conditional random fields (CRFs) are used to cluster regions together or assign region labels, taking into account pairwise interactions [43].

Our proposed layered bag-of-words might also be called a hierarchical bag-of-words, but the latter term is occasionally employed [44] to refer to pyramid match kernels built on top of histograms [45]. The difference between our proposed representation and a histogram pyramid is that in our approach sparse descriptors are merged in a data-driven fashion to build our hierarchy, rather than the fixed spatial merging of a pyramid. In this sense, our layered bag of words bears more relation to MRF and CRF based image segmentation and labeling methods [43, 46, 47].

2.4 Domain Adaptation, Feature Selection

Domain adaptation techniques can be powerful across limited and well-characterized domains, such as in [48]. However, the gains are often modest, and as the aptly titled work "frustratingly simple domain adaptation" by Daumé [49] shows, even simple techniques can outperform sophisticated do-

main adaptation methods. Likewise, transfer learning methods such as transductive SVMs [50] can provide modest benefits, but are often computationally expensive and often restricted to datasets with common classes. Formally, our proposed work in collaborative filtering for feature recommendation can be considered a type of inductive transfer learning [51], where the goal is to transfer knowledge when both the tasks and domains are related, but not identical.

Feature selection methods can broadly be divided into filter and wrapper methods according to the taxonomy of Guyon and Elissee [52]. In filter methods, features are selected without knowing the target classifier, while wrapper methods more directly optimize for the features that maximize accuracy with the end classifier. Filtering methods are often computationally cheaper but less powerful than wrapper methods, but at the same time there is also a larger risk of overfitting in wrapper methods [52, 53, 54]. Our feature recommendation can be seen as a type of feature ranking technique [54, 52], however unlike typical boosting-based methods [55, 56, 54] we rank features according to their predicted performance through collaborative filtering.

Common domain adaptation techniques assume that the specific task is the same between the source and target domains [48, 57, 49], and this assumption is the most common one in transfer learning as well [56, 50, 58]. That is to say, if the problem were action recognition, then these techniques would need the specific actions to be matched between the domains. For example, Cao *et al.*perform cross-dataset action detection, using one dataset (KTH) [59] to improve performance on a related one (MSR Actions Dataset II) [60]. However, the particular actions that are detected are present in both datasets, and indeed MSR was constructed to share those actions with KTH. Likewise, Lai and Fox [48] use the Google Sketchup database of 3D models to match laser scans using domain adaptation, but this technique is limited to those objects that are present in the sketchup database.

In contrast, we propose to recommend features for a particular task without requiring that that task is mirrored in our synthetic or coarsely labeled data.

Dictionary learning approaches using supervised or class-aware methods have shown variable improvements, ranging from substantial [61, 62, 63] to modest [64, 65]. As Boureau *et al.* [64] demonstrate, the overall performance of a system can be affected by the individual components in complex ways, accounting for much of this variability. In particular, techniques which rely

more heavily on individual features are more likely to benefit from class-aware dictionaries. For example, Brendel *et al.* [62] associate a single feature with each frame in a video, and they find that producing a separate dictionary for each action gives a substantial improvement over a shared dictionary. However, the difficulty is that a dictionary tuned to one class may actually degrade performance on others [66]; this need to avoid overfitting to particular classes, as well as the lack of forward knowledge about classes, leads us to our relatively simple selection method, which trades off power on any specific class for robustness across classes.

We propose to recommend features from a randomly generated pool, as random classifiers and features have, by themselves, shown benefits over other representations [67, 68, 1].

Blitzer *et al.*use a set of 'pivot features' [69] to perform domain adaptation, by learning the relationships between a small number of labeled pivot features and a larger number of unlabeled features in both domains. Through the pivot features they attempt to transfer models built on the unlabeled features between domains.

Multi-task learning is a sub-domain of transfer learning [51, 2] in which the goal is to learn common features or representations for a number of related tasks. Typical approaches formulate the problem as feature selection in which a shared sparse set of features must be chosen for all tasks [70, 71] or as an SVM-type problem that enforces an analogous sparsity on support vector weights [72]. Other work in multi-task learning uses neural network architectures to structurally force all the tasks to share intermediate representations [2, 73]. An interesting approach by Ruckert and Kramer [74] first learns a good kernel for each task, using standard kernel learning techniques. Then, using those learned kernels and tasks as training samples, they learn how to predict the kernel for a task. This is similar to our proposed collaborative filtering feature recommendation, except that we are predicting the usefulness of features rather than the classifier in which those features will be used. Additionally, Ruckert and Kramer rely on a "meta-kernel" of heuristics to compare how similar datasets are, while in our proposed approach the feature ratings themselves are used to determine the similarity of datasets.

A meta-learning approach by Mierswa and Wurst [75, 76] is similar to our proposed method. Like us, they seek to recommend good features for a given task given a small set of evaluated features. They do this by training a linear SVM on each task using a small set of 'base features'

as inputs; they compute the similarity between tasks according to the SVM weights assigned to the base features. The features recommended for a task are then the features that performed well for the neighbors of that task. Our approach differs in that we do not designate a fixed set of base features, but instead allow all features to potentially be used for that purpose. Our formulation of the problem in this fashion allows us to use essentially any collaborative filtering technique unmodified. Similarly, [77] uses additional information about features in order to learn an association between those 'meta-features' and the usefulness of the underlying features on tasks. Our proposed approach differs in that we do not use additional information about either tasks or features; we treat it as a pure collaborative filtering problem where only the ratings matrix between features and tasks is known.

For collaborative filtering we initially explore the approaches used by the BellKor team to (as a combined effort with two other groups) win the Netflix prize [78, 79]. As the final result was a blend of over one-hundred different approaches, spanning the gamut from neighborhood methods to factorization to regression techniques, their method serves as a fairly comprehensive overview of large-scale collaborative filtering techniques. At first we consider basic neighborhood-based [80, 81, 82] and factorization techniques [83, 84], but later will we include more sophisticated regression methods [85] as well. Interestingly, collaborative filtering itself can be cast as a multi-task learning problem [86], where the goal is to learn common structures that allow item ratings to be predicted from other ratings. It should be stressed, however, that using multi-task learning to do collaborative filtering across a set of tasks is completely different from directly applying multi-task learning to those tasks.

2.5 Synthetic Data

There has been limited work on using synthetic data for action recognition. A number of approaches use synthesized silhouettes from motion capture to match actions (*e.g.*, [87]), but these both require a constrained domain (silhouettes can be extracted) and the action being searched must be present in the data. Another line of work by Qureshi and Terzopoulos [88] uses synthetic crowds to tune sensor networks for surveillance. In still image analysis, Pinto *et al.*use a screening approach to select entire models from synthetic data [89], by simply randomly generating large numbers of candidate models and evaluating (screening for) which ones perform well. The generated models

are convolutional architectures in which the filters for the intermediate layers are learned through unsupervised learning. However, in a later work Pinto and Cox [1] abandon the unsupervised learning of intermediate filters in favor of randomly generated ones, so that the screened models are completely random. Philosophically, our proposed feature recommendation can be seen as an extended type of screening, in which both models (features) *and* tasks are randomly generated and evaluated against one another.

A recent work on pose estimation from depth images [90] uses hundreds of thousands of synthetic depth images of human postures to train randomized decision tree classifiers to label pixels with body parts. While this approach demonstrates the potential power of synthetic data, it takes advantage of a highly constrained task and domain (human body part identification from depth images) where very realistic synthetic data can be generated specifically for the target task. This allows them to use straightforward supervised learning. In contrast, for most domains 'plausible' synthetic data is the best that can be expected.

3 Preliminaries

3.1 General terminology and notation

We begin by describing the general problem setting and related terminology (see Fig. 1).

3.1.1 Tasks and data samples

A task T_j is assumed to be a self-contained forced choice classification problem. Denote a task

$$T_j = \{(x_{j,1}, y_{j,1}), (x_{j,2}, y_{j,2}), \ldots\}, \tag{1}$$

where $x_{j,z}$ is the zth data sample associated with task T_j, and $y_{j,z}$ is the label corresponding to that sample. The data sample $x_{j,z}$ might be some large, complicated representation (*e.g.*, a video clip), while the target label $y_{j,z}$ is a discrete value indicating which class a given data sample belongs to. Note that labels are not consistent across tasks; there is no necessary correlation between the labels in any tasks. Different tasks can have different numbers of class labels. Also note that the data samples associated with a task need not be unique; the same data sample might be shared

across many tasks (for instance, if one dataset is used for many different tasks). For simplicity, the remainder of this proposal assumes that target labels are binary; that is, that $y_{j,z} \in \{-1, 1\}$. In general, our goal is to generate or recommend features or representations that work well on a particular *target task*, using a collection of synthetic or low-quality real *training tasks* or *source tasks* (the transfer learning term [51]) to do so.

3.1.2 Features and feature ratings

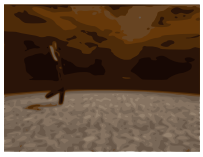
$f_i :$  $\rightarrow \mathbb{R}^{n_i}$

Figure 3: A feature is simply a function that maps from a data sample (typically a video or image) to a real-valued vector.

We use the term feature in a very broad sense, where by 'feature' we simply mean some method or *function* that takes as input a data sample and returns some value as an output. In general this value might take any form, but for simplicity we restrict our attention to functions whose return values are vectors (*i.e.*, we will not discuss features that might return more complicated data structures).

A feature is a mechanistic function from one space (data samples) to another (the feature value; see Fig. 3); there is no intrinsic reason to prefer one feature over another. Indeed, the suitability of a given feature might vary wildly across tasks. We formalize the notion of how well a feature does on a given task as that feature's *rating* on that dataset.

Features are applied to individual data samples, while feature *ratings* are computed over entire tasks. While we were deliberately vague in our description of features, because the underlying details of the feature are largely irrelevant, we must be more precise in our construction of feature ratings. The actual values computed by different features are never compared against one another, so there is no need to enforce consistency among them, but the feature *ratings* themselves must be comparable.

The rating of a feature on a task is denoted by $R(f_i, T_j) = r_{i,j}$. Ratings are restricted to the

range $[0, 1]$, where a higher rating is meant to convey that a feature is in some sense better on a task. If $r_{i,j} > r_{i',j}$, then feature f_i should give better discriminative performance on task T_j than feature $f_{i'}$.

In practice, this rating function might be implemented in many ways. One approach would be to use one of many information-theoretic measures, such as mutual information or the "relief" measure [91], to gauge how much information the feature values give about the data sample labels. However, in preliminary work with simple features we found that the stump classifier accuracy for a given feature gave better performance than mutual information. Thus, we take the straightforward approach of computing the rating as the accuracy of a *reference learner* (we use a linear SVM) using a given feature's output to predict a dataset's labels, normalized so that chance accuracy maps to a rating of 0.0, and perfect accuracy to a rating of 1.0. For the special case of a feature which produces a scalar output, this reduces to the accuracy of a stump classifier on that value.

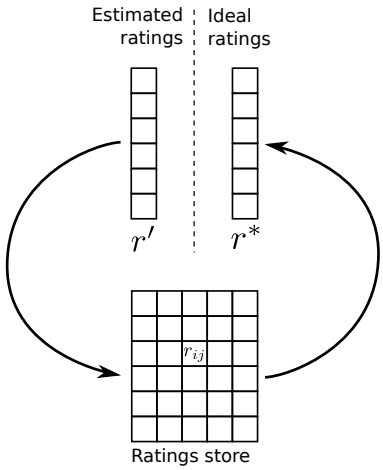


Figure 4: The goal is to predict the ideal ratings of features (*i.e.*, the ratings of features on the testing portion of the target dataset), using the noisy estimated ratings evaluated on the training portion of the target dataset.

3.1.3 Estimated vs. ideal ratings

One of the fundamental difficulties facing any feature selection or recommendation system is that the performance of features, as measured on a *training* set of data, may differ, perhaps significantly, from the performance of those features on the *test* data. For example, a set of features might be

chosen that (through overfitting) perform perfectly on the training data; however, such features are unlikely to preserve that performance on the test data. This is especially likely when the number of training samples is small. We term the ratings of features on the training data *estimated* ratings, since they are noisy estimates of how we expect those features to perform on the actual test data. Likewise, we denote the actual performance of features on the test data to be the *ideal* ratings. If features were all totally independent, then the best estimate of the ideal ratings would be produced by the estimated ratings from the training data. However, if features are related, then potentially a model of feature relationships could be used, along with the estimated ratings, to better predict the ideal ratings. For example, if the performance of features were related by a lower-dimensional linear model, then knowledge of that model combined could be used to reduce the error of the estimated ratings by constraining them to that linear model.

This distinction distinguishes our feature recommendation problem from more typical recommender systems with explicit feedback, where ratings (generally given by human users) are taken as ground truth. That is to say, in a typical recommender system, the system cannot know better than the user what his or her rating of an item is; if the system and the user disagree, the system is in error.

Beyond the computational benefit, we hypothesize that the predicted ratings from collaborative filtering could be closer to the true ratings than even the estimated ratings for the same features on a given dataset, so that $|r' - r| < |\hat{r} - r|$. See Fig. 4.

3.2 Datasets

The primary type of task addressed in this proposal is action recognition, which is typically formulated as a forced choice classification problem from relatively short video clips to a limited number (on the order of tens) of action classes. Action recognition can also be formulated as a detection problem, in which the goal is to locate (in space and time) actions of particular classes in longer video clips, where each clip may contain many instances of the actions. However, there are comparatively few datasets that support this task, as it requires a much greater level of data annotation than classification– with classification, each clip needs only one label, while for detection, the actions must be localized in the clip as well, requiring per-frame annotations.

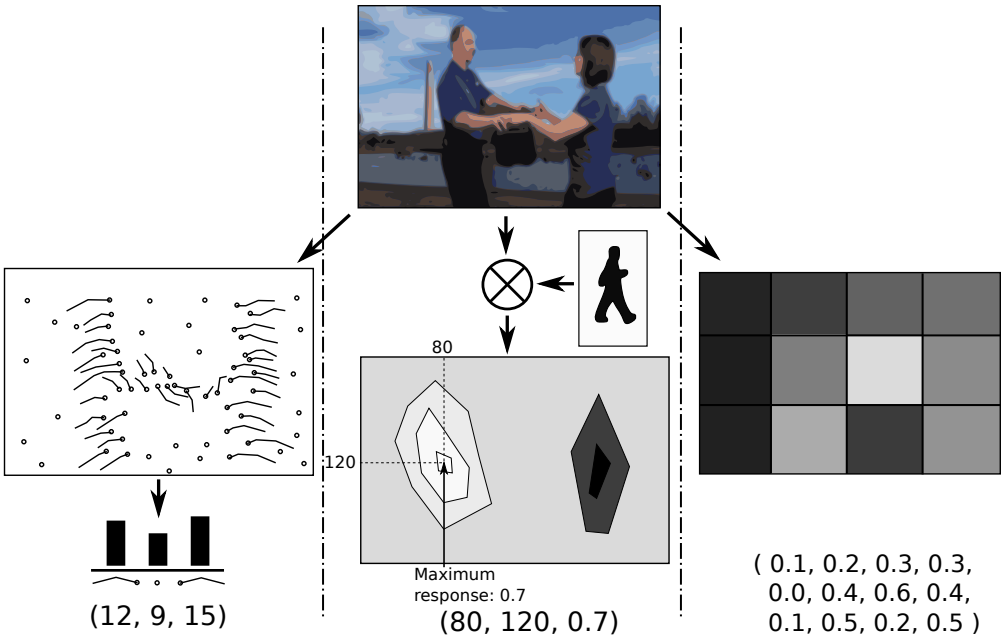


Figure 5: Examples of what can be considered *features* in our framework. On the left, a histogram of motions is computed from the video. In the center, the maximum response and location of that response for a motion template comparison is computed. On the right, the video is gridded, and the magnitude of the motion in each cell is computed. A feature is simply a function computed from a video, and so can represent a large range of operations.

Additionally, in order to test the feasibility of the proposed approaches on appearance (rather than motion) data, we consider the problem of determining whether the faces in two images are of the same person. We choose this category of task due to its utility, the availability of considerable amounts of data, and the body of literature and results against which we can compare.

Table 1 lists the datasets we consider to test our tasks, as well as some of their properties. Datasets which have an asterisk next to whether they are publicly available are available in the sense that similar samples be collected or generated by anyone, but the particular samples used are not. Note that the synthetic data and the random YouTube clips are not suitable for evaluation because the former is not a real-world task (and hence it is unclear that results on the synthetic data are directly meaningful), and the latter because the annotations (tags) of random YouTube clips are so noisy that it would be difficult to interpret the results.

UCF-YT is the UCF YouTube "Actions in the Wild" dataset (UCF-YT) [92]. The UCF-YT

dataset is a straightforward forced-choice classification problem between 11 action classes (mostly various sports). The dataset contains 1600 videos of approximately 150 frames each, and we divide these videos into a training set of 1222 videos and a testing set of 378 videos. The UCF-YT dataset is further sub-divided into subsets of related videos (*e.g.*, all from the same sports match, or sharing the same background); in order to avoid training/testing contamination from closely-related videos, we employ a stratified training/testing split that places each subset either entirely in the training or the testing set. UCF-YT-50 is the recently released 50 class version of the UCF-YT dataset. It is similar to UCF-YT, other than containing more video clips and action classes.

Rochester Activities of Daily Living dataset consists of 150 videos of five individuals performing a series of scripted tasks in a kitchen environment, acquired using a stationary camera. Due to the limited pool of available data, we evaluate using 5-fold cross-validation, using videos from four individuals for training and the fifth for testing, in each fold.

The Mind's Eye dataset is the dataset for the DARPA Mind's Eye project. However, this dataset has not yet been finalized, so the detail of annotations, as well as the number of clips, are not known at this time. The motion capture databases listed are used to generate the motions within the synthetic data; they have no direct task.

3.3 Synthetic data

This section describes the method by which we generate synthetic data to support action recognition tasks.

Our proposed work relies on being able to generate relatively large amounts of synthetic data. Since it is difficult to produce synthetic data that is comparable to real-world data in terms of raw pixel-level appearance, we concentrate on the simpler task of generating synthetic data that matches real-world data in terms of *motion*. We make no attempt to mimic real-world appearance: the human model in our synthetic data is a abstract armature (Fig. 6). However, in terms of motion it is a reasonable analog, since its motion is derived from human motion capture data (Fig. 8).

3.3.1 Motivation for synthetic data

The motivation for using synthetic data is threefold:

Table 1: Datasets and their properties

Dataset	Annotations	Size	Public?
UCF-YT [92]	Coarse	Large	Yes
UCF50 [93]	Coarse	Large	Yes
Rochester Daily Living [29]	Coarse	Small	Yes
Mind's Eye	Unknown	Unknown	Yes
VIRAT [94]	Medium	Medium	Yes
CAVIAR [95]	Fine	Small	Yes
TRECVID [96]	Fine	Large	No
Synthetic	Very fine	Very large	Yes*
Random YouTube clips	Coarse	Very large	Yes*
CMU Motion capture database [97]	Fine	Large	Yes
Facebook faces	Coarse	Very large	Yes*

- Size: More synthetic data can be generated than is feasible to gather from real-life sources with any sort of annotation. For example, the largest video datasets may have approximately 5,000 clips, a number which can be synthetically generated in less than an hour. Since the synthetic data is produced by a program, distributing the synthetic data is only a matter of distributing the program and data files, which together will be a fraction of the size of even a small video dataset. Furthermore, the statistical significance of an algorithm can be easily tested by drawing more samples from the same synthetic distribution; this eliminates the worry that an algorithm might be indirectly overfitting to a particular set of samples.

- Detail: Available video datasets are very poorly annotated. The best case annotation is usually only a single class label for the entire clip, and in some datasets (Hollywood), even that is uncertain. In contrast, synthetic data can be generated with rich annotations, including per-pixel assignment to rigid bodies, foreground-background segmentation, and depth, in addition to the parameters (*e.g.*, motion capture file, position, and viewing angle) that were used to generate the motion and appearance.

- Variation: Bias is inherent in any dataset, simply because the 'world distribution' of possible video clips is impossible to sample from. Any dataset will be biased by the methodology used to collect it, and furthermore, even a supposedly ideal, unbiased dataset would not necessarily be the best in practice, since real world action recognition tasks are themselves 'biased'. But while eradicating bias is an unachievable goal, it is still worthwhile to build large and varied datasets, and synthetic data allows for automated variation generation in ways that are difficult to achieve with real datasets (for an analysis of dataset bias in real-world datasets, see [98]).

  For example, varying the viewpoint is simple with synthetic data, but in a real dataset collecting multiple viewpoints (especially large scale differences and extreme views) can be difficult. Likewise, real datasets often have relatively few actors because it is difficult to find enough people willing to take the time to record actions, but with synthetic data actor appearance and proportions can be easily varied. Synthetic data also allows a degree of *control* over variation that is not possible with real data, such as drawing synthetic samples over a large range of viewpoints in order to quantify the effect of viewpoint variation on an algorithm.

3.3.2 Synthetic data organization

The synthetic data is organized into groups of clips, or tasks. Each tasks consists of a number of *positive* samples all generated from a single motion capture sequence, and a number of *negative* samples randomly drawn from the entire motion capture dataset. In this way, each synthetic task represents an independent binary classification task where the goal is to decide which clips belong to the action vs. a background of all other actions. The actions used in the synthetic data do *not* necessarily correspond to the actions used in any final classification task on real data. Since the synthetic actions are randomly chosen out of motion capture sequences, they may not correspond to easily named actions at all. A typical synthetic clip might be 90 frames long, and a typical synthetic task might be associated with 100 such clips, corresponding to 50 positive samples and 50 negative samples. An example frame from a synthetically generated video can be seen in Fig. 6.

A clip is produced by moving a simple articulated human model according to the motion capture

Figure 6: Example frame from synthetic data. The synthetic video is abstract in appearance, but the movement of the armature man is derived from motion capture data. The texture does not match real data in appearance and serves only to provide descriptor extractors (*e.g.*, trajectories derived from optical flow) with stable inputs.

sequence, with some added distortions. The synthetic data is rendered at a resolution of 320×240 and a nominal framerate of 30fps in order to match the MSR and UCF-YT datasets (see Sec. 3.2). The generation of this synthetic data is quite efficient, as actual rendering time is approximately 6× faster than realtime, so that two synthetic clips can be rendered every second. At this unoptimized rate over 170,000 clips could be generated each day on a single computer; as such, the bottleneck with synthetic data lies in the processing of the resulting video.

3.3.3 Motion generation

The motion of the human model in the synthetic videos is produced by taking motion capture sequences from the CMU motion capture database or other motion capture databases listed in Table 1. and adding temporal distortions and time varying noise to the joint angles.

For each clip a motion capture file is chosen from the 2500 clips in the CMU motion capture database. If the clip is meant to be a positive example, then the motion capture file and approximate location within that file is given, and the starting frame is perturbed by approximately ±1s. If the

clip is meant to be a negative example, a motion capture file is randomly chosen from the entire database, and a starting position within that file is randomly chosen.

Next, temporal distortion is added by introducing a temporal scaling factor (*e.g.*, if the factor is 2.0, then the motion is sped up by a factor of two). Non-integral scaling factors are implemented by interpolating between frames of the motion capture file. Then, a random piece-wise linear function is used to dynamically adjust the temporal scaling factor of the rendered clip. In practice, we limit the random scaling factor to drift between a value of 0.1 and 2.0. Consequently, the timing of a rendered clip differs from that of the base motion capture file in a complicated and nonlinear fashion.

A similar approach is used to add time-varying distortion to the joint angles. A random piece-wise linear function is generated for every degree of freedom for every joint in the armature, and this function is simply added to the joint angles obtained from the motion capture sequence. The magnitude of this distortion is ± 0.3 radians.

We add several other distortions and randomizations to the synthetic data. The viewing angle is randomly chosen for each clip, as is the viewing distance. Additionally, the position of the actor/armature is randomized for each clip. The lighting is also randomized between clips, because the effects and positions of shadows can have a significant effect on the extraction of feature trajectories.

3.4 Feature types

The following list of possible feature types is not meant to be comprehensive; one of the advantages of the method is that it is largely feature agnostic, so it is easy to add many different types of features to the candidate pool. For examples of what can be considered features, see Fig. 5.

3.4.1 RBF classifier features

A type of feature that can easily be built on top of virtually any descriptor is to count how many descriptors a binary classifier (in this case, a Gaussian weighted nearest neighbor classifier) classifies as the positive class. This type of feature can be seen as the generalization of a histogram bin, where the associated classifier is a membership function that tests whether a given descriptor belongs to

the implicit 'label set' of that classifier. Unlike a traditional histogram, however, this type of feature allows a descriptor to belong to multiple labels.

Such a feature is defined by a weighted nearest neighbor classifier, with samples weighted according to a Gaussian radial basis functions, so that the value of the classifier c_j is given by

$$c_i(d) = \text{clip}(\sum_h w_{i,h} \cdot e^{-\beta_{i,h}||d-v_{i,h}||^2}), \tag{2}$$

where $v_{i,h}$ is one sample vector for classifier i_j, $w_{i,h}$ and $\beta_{i,h}$ are the weight and beta for that sample vector, and d is a descriptor. The clip(.) function clips the value to the range $[0, 1]$, so that values less than zero (rejections) are thresholded to zero, while values above one (accepts) are clipped to a maximum of 1.0 and all other values are unchanged. We use RBF classifiers as features due to their generality, but other classifiers could be considered.

A set of such classifier can also be seen as computing an intermediate representation q_w corresponding to each descriptor d_w, so that

$$q_w = (c_1(d_w), c_2(d_w), \ldots, c_n(d_w)), \tag{3}$$

where n is the number of classifiers in the set.

Following this intuition, a set of n such RBF classifiers can be seen as computing a generalized histogram, where each RBF classifier feature computes one bin of the "histogram" according to

$$f_i(x) = \sum_{d \in x} c_i(d), \tag{4}$$

where x is a data sample with associated descriptors d_w (*e.g.*, all the descriptors computed from a given video). The entire histogram g for data sample x is expressed as

$$g = (f_1, f_2, \ldots, f_n) = \sum_{d_w \in x} q_w, \tag{5}$$

which is to say that the classifier c_i is treated as an indicator function for whether a descriptor belongs to label i, where a descriptor might have multiple labels, and where the labels a descriptor d_w takes on are given in the vector q_w.

Note that under our definition of feature, we have a choice of whether to consider the entire histogram g to be a *single* feature (with output dimension n), or whether to partition it into n

separate features. The performance difference between these two options remains to be tested experimentally (see Section 4.5.2).

For the purpose of populating the candidate pool, random RBF classifier features can be generated by selecting the sample vectors at random from a sampling of descriptors from a training set, and then setting the remaining parameters at random (*e.g.*, from uniform or normal distributions).

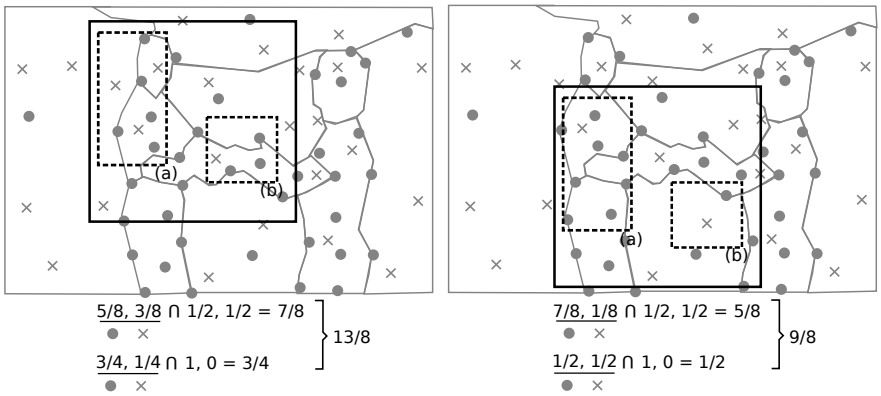


Figure 7: The weighted window feature consists of a number of offset windows within a central window, each of which computes a histogram intersection between a stored histogram and the feature histogram of its sub-window. In this case, there are two subwindows (a) and (b), with stored histograms $(1/2, 1/2)$ and $(1, 0)$ respectively. The symbol $\cap$ denotes the histogram intersection. The window is shown here evaluated at two positions; when used as a feature, the value is the maximum response over all possible positions.

3.4.2 Weighted window features

Since the previous features we have been discussing have largely been sparse features that discard what spatial location information they hold, we now introduce another feature type that takes advantage of that spatial distribution. The feature consists of a number of windows $W_i = (px_i, py_i, sx_i, sy_i, q_i)$ where each window is associated with a position (x_i, y_i) relative to a sliding window center, a size (sx_i, sy_i), and a histogram vector q_i. Then, the value of the feature at a given location is computed by taking the histogram intersection each window's histogram vector and the sparse feature label histogram within that window. That is, let $H(F, cx, cy, w, h)$ be the histogram of feature labels for the features in F in the window defined by the position and size (cx, cy, w, h). The weighted window feature's value at a given point is

$$v(W_i, cx, cy) = \sum_n H(F, cx + px_i, cy + py_i, sx_i, sy_i) \cap q_i, \tag{6}$$

where the intersection symbol $\cap$ denotes the histogram intersection (that is, $a \cap b = \sum_i \min(a_i, b_i)$). The value of the window feature over an entire clip is simply the maximum response over all possible center locations

$$V(W_i) = \max_{cx} \max_{cy} v(W_i, cx, cy). \tag{7}$$

The weighted window feature is illustrated in Fig. 7. Note that this is similar to the type of histogram window that is used for image classification [99].

As a practical detail, the computation time of the histogram $H(.)$ can be significantly reduced by pre-computing integral histograms, so that each sub-window histogram can be computed in constant time. To further reduce the running time we can also only compute feature values at coarse locations in the frame, rather than at every possible pixel-location.

These features can also be randomly generated, but care may need to be taken as to how the weight vectors q_i are generated. While it is always possible to simply randomly generate these weights as well (*e.g.*, as from a normal distribution), this may result in an especially large proportion of useless features. An alternative approach is to sample the weights from training data, so that each q_i is taken as the histogram of a randomly chosen window from a randomly chosen data sample. Thus, we construct the windows so that each window computes a histogram correlation between a (randomly sampled) training patch and the test patch.

4 Proposed Work: Feature Recommendation

The following section details our proposed work on feature recommendation. After an introduction to the work, we describe our preliminary work and established methods, followed by an experimental plan. We conclude with an exploration of interesting avenues of research in this direction, as well as an assessment of risks involved with this work.

4.1 Problem Formulation

We propose to learn from large heterogeneous collections of tasks by introducing an approach we term *feature recommendation*. The overall setup of the problem is as follows: we are given a collection of tasks (since we concentrate on action recognition in this proposal, we can assume these are *video classification* tasks) with associated tasks; we refer to this collection as the *source corpus*. Additionally, we are given a collection of *features* which have been *rated* on some or all of the tasks in the source corpus; we refer to this set of features as the *candidate pool*, and the matrix of the candidate pool's ratings on the source corpus as the *ratings store*. The rating of a feature on a task is meant to correlate with its performance on that task.

Now, given a new task (likely associated with a new dataset as well), the *target task*, we rate some or all of the features on the target task; we denote the set of features rated on the target task as the *probe set*. Then, using the ratings of the probe set on the target task (the *probe ratings*), the objective is to recommend, or predict, which features from the candidate pool will be highly rated on the target task. Alternatively, the objective is to recommend features which, when used jointly, maximize performance on the target task.

In this problem formulation the features and tasks are opaque; that is, we can rate a feature (or jointly rate a *set* of features) on a task, but the process by which this is accomplished is feature and task specific, and not necessarily open for inspection. The problem is to predict, based only on the structure of the correlations between feature ratings in the source corpus and the limited set of ratings on the target task, which features are likely to perform well on the target task.

4.1.1 Analogy to Recommender Systems

Recommender systems are employed in many contexts, such as video rental (Netflix) and product recommendation (Amazon.com). Generally these systems assume that users explicitly rate items; these are referred to as *recommender systems with explicit feedback*. In these systems, the goal is to recommend, based on users' previous ratings, items that are likely to be highly rated by those users. For example, Netflix will attempt to recommend movies that a person is likely to enjoy based on his or her ratings of other movies. Because these systems use explicit feedback, they do not need to know about the particular content of videos/items; they can make their recommendations

based solely on correlations between ratings.

The analogy to feature recommendation is direct: in feature recommendation, the goal is likewise to recommend features (items) that are likely to perform well (be highly rated) on a target dataset (specific user). And, just as recommender systems for products do not need to know about those products, only about their ratings, the feature recommendation approach does not constrain or examine the internals of features, beyond the requirement that features can be evaluated and rated on tasks. Furthermore, recommender systems do not make general recommendations, but rather make recommendations for specific individuals, based on their past ratings. Similarly, in contrast to multi-task learning, our goal in feature recommendation is not to select generally good features, but rather to find specifically good features for a particular target task.

The approaches employed for recommender systems broadly fall under the heading of *collaborative filtering*, and it is these approaches that we propose to employ for our feature recommendation task as well. The following sections formalize the descriptions of features, tasks, and ratings.

4.2 Preliminary work: feature recommendation by simple aggregate performance

Many popular bag of visual words (BoW) techniques rely on quantizing descriptors (where a descriptor might be a trajectory fragment, or a HOG descriptor computed around an interest point, or many other possibilities) computed from video; generally either simple unsupervised techniques such as k-means clustering [59, 10, 4, 35] or hand-crafted quantization strategies (such as our preliminary work on trajectons) are used. The end result of these quantization schemes is a histogram that counts how frequently features are quantized into particular quantization bins. We generalize this notion of quantization by suggesting that each bin of the resulting histogram can be considered an independent feature defined by a classifier that decides whether a given descriptor should be counted by that bin. Then the problem of designing a *quantization scheme* can be seen as the problem of recommending a set of such histogram count classifier features.

While the goal of this proposal is to be able to efficiently recommend features tailored specifically to a new target dataset, as preliminary work we consider the problem of recommending *generally good* features using synthetic data. More concretely, in this preliminary work we consider the

problem of recommending a quantization scheme for a bag-of-words technique, using a corpus of synthetic data as the source from which to recommend the quantization scheme. We use a simple form of feature selection in which we independently recommend features by rating them on all the synthetic datasets, and then assigning them aggregate ratings from the ratings on individual synthetic datasets.

Figure 8: Our synthetic data looks nothing like real data, but in terms of motion they are similar.

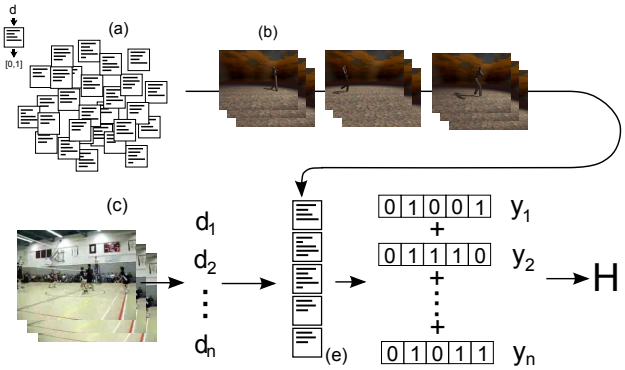


Figure 9: System overview: a pool of randomly generated features (a) is filtered, or seeded, on synthetic data (b) to produce a greatly reduced number of features (e) that are likely to be informative. Real data (c) has descriptors (*e.g.*, trajectories) extracted, and these descriptors are fed through the seeded feature set to produce label vectors y_i, one per descriptor. These label vectors are then accumulated into the histogram H, which represents the video clip.

4.2.1 Feature seeding overview

The basic organization of our method can be seen in Figure 9. First, a set of synthetic video clips is generated using motion capture data. These clips are generated in groups (datasets), where each group is an independent binary classification problem; this is described in detail in Section 3.3.

Next, raw motion descriptors are extracted from the synthetic data pool in the form of trajectory snippets [28, 29] (see Appendix A) and histogram of optical flow (HOF) descriptors around space-time interest points (STIP) [4]. We use the unquantized form of the trajectory descriptors, since our purpose is to replace the quantization schemes outlines in Appendix A with one recommended from synthetic data. Likewise, we use the 90-dimensional STIP-HOF descriptors without any pre-emptive quantization.

Each clip produces many descriptors– trajectory descriptors produce on the order of 300 descriptors per frame of video, while STIP-HOF produces closer to 100 descriptors per frame. These descriptors are sampled to produce a candidate pool of features, where each feature is associated with radial basis function classifier (see Section 3.4.1 whose sample vectors are randomly drawn from the descriptors. Then the synthetic data is used to rank features based on their aggregate classification performance across many groups of synthetic data. We denote the highly-ranked features selected in this way as the *seeded* features. The seeded features can then be applied to real data and used as input to conventional machine learning techniques. For evaluation, we consider the seeded features in a basic BoW framework, using linear SVMs as classifiers.

Note that for this preliminary work we are not proposing a complete action recognition system, but concentrate only on motion features in order to evaluate the potential of such a screening based approach from synthetic data.

4.2.2 Feature pool generation and evaluation

For this preliminary work we consider random RBF features of the form described in Section 3.4.1. We generate a pool (in our case, of size 10,000) of such features by randomly selecting sample vectors from the synthetic dataset's descriptors. The weight associated with each sample vector is chosen from a normal distribution $N(0, 1)$, and the β associated with each sample vector from a uniform distribution over the range $[0, 10]$. These parameters were chosen arbitrarily to generate a large range of variation in the classifiers. Example trajectory descriptors that might be accepted by these types of features can be seen in Fig. 10.

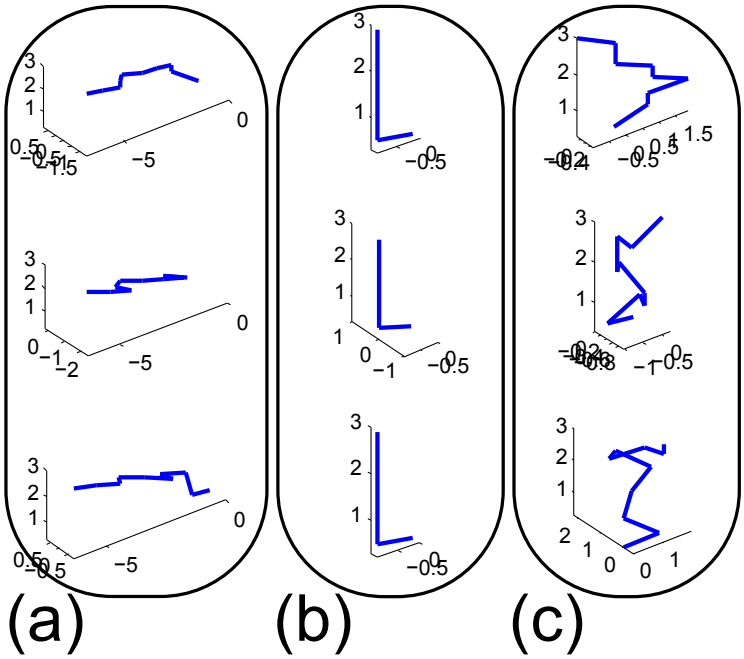


Figure 10: Examples of trajectory descriptors accepted by different classifier features. Some features represent simple concepts, such as leftward movement (a), or a quick jerk (b), while others do not correspond to anything intuitive (c). Given limited labeled data (c) could be indicative of overfitting. Feature seeding allows us to confidently determine that the chosen features generalize well.

4.2.3 Feature seeding/filtering

Given the pool of features, we select for, or seed, a good set of features from the pool by rating them on a set of synthetic data. In practice, the seeding is similar to a single iteration of boosting, with the important difference that the seeding attempts to find features that work well across many different problems, rather than a single one.

Let P_n and N_n correspond to the sets of descriptor sets (videos) in the positive and negative sample sets, respectively, of a synthetic group n. Then we can express the rating $r_{k,n}$ of a feature k on group n ($n = 1, \ldots N$) as

$$r_{k,n} = \max_t \frac{\sum_{D \in N_n} I(b_k(D) \leq t) + \sum_{D \in P_n} I(b_k(D) > t)}{||N_n|| + ||P_n||}, \tag{8}$$

where $b_k(D)$ is the result of evaluating feature k on descriptor set (video) D, and $I(.)$ denotes the

indicator function.

Note that this is just the accuracy of a decision stump on the $b_k(D)$ values. We have also considered mutual information based rating, but we find that it has slightly worse performance, probably because the stump-classifier rating we use here is a better match for the final SVM classification. However, our method does not depend on any single rating metric, and it is straightforward to swap this metric for another.

Now, we express the aggregate accuracy of a feature over all groups as

$$A_k = g(\{r_{k,n} | n = 1, \ldots, N\}), \tag{9}$$

where $g(.)$ is a function that operates on a set. In our case, we consider three possible aggregation functions g: $g_{\min}(X) = \min(X)$, $g_{\max}(X) = \max(X)$, and $g_{\text{avg}}(X) = \text{mean}(X)$. Intuitively, $g_{\min}$ takes the worst-case performance of a feature against a collection of problems, $g_{\max}$ takes the best-case performance, and g_{avg} takes the average case. Note that because the evaluation problems are randomly generated from a large motion capture database (see Section 3.3), it is unlikely that they will share any action classes in common with the target task. The goal is to select features that perform well against a variety of action recognition tasks (*i.e.*, , that can discriminate between different human actions).

Then we simply rank the features according to their A_k values and select the top s ranked ones. In practice, we use seeding to select the top $s = 50$ features from a pool of 10,000.

Given a set of training and test videos on real data, we compute histograms h_D, where each histogram is computed according to (Eqn. 5) over the reduced set of s features. Then we simply train a linear SVM as the classifier.

4.2.4 Datasets

We primarily evaluate on the UCF-YT dataset as described in Section 3.2. In addition, we evaluate this work on the Microsoft Research Action Dataset (MSR) [100], which consists of of sixteen relatively long (approximately 1000 frames per video) videos in crowded environments. The videos are taken from relatively stationary cameras (there is some camera shake). The dataset only has three actions — clap, wave, and box, with each action occurring from 15 to 25 times across all

videos. The actions may overlap. For evaluation we consider MSR to be three separate binary classification problems, i.e., clap vs. all, wave vs. all, and box vs. all, rather than a three-way forced choice because the actions overlap in several parts. Each problem has an equal number of negative samples drawn by randomly selecting segments that do not feature the action in question, so for example, the wave vs. all problem is a binary classification between the 24 positive examples of the wave action and 24 negative examples randomly drawn from the videos. Due to the limited amount of data in this set, evaluation is by leave-one-out cross validation.

As described earlier, for feature seeding we use a synthetic dataset, which consists of 2000 short videos; the "actions" in this dataset do not necessarily correspond to any of the action classes in either the UCF-YT or MSR datasets.

4.2.5 Feature statistics

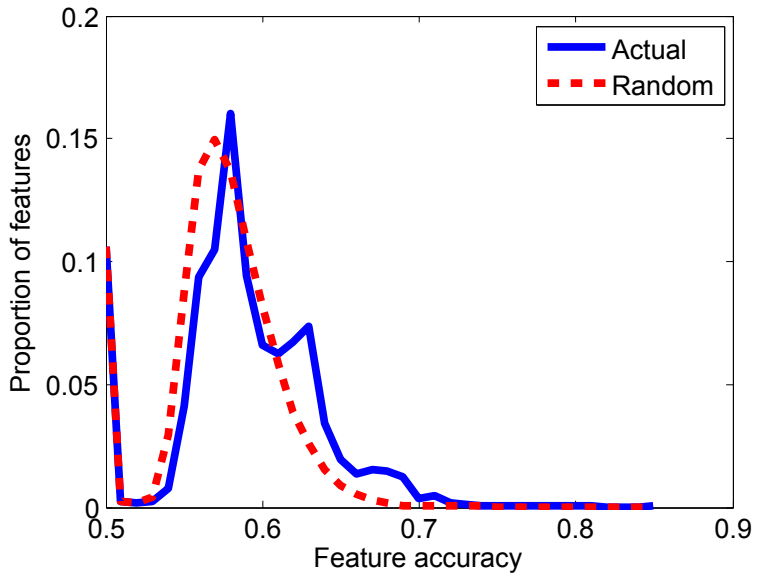


Figure 11: Accuracy distribution of RBF classifier features on synthetic data, compared with the expected number of false positives. Above accuracy 0.61, the majority of features are true positives. The difference between these two distributions is statistically significant to $p < 0.001$ according to the Kolmogorov-Smirnov test.

A natural question to consider is how informative these RBF features are; that is, how likely is our seeding method to find useful features. Because the features are evaluated by treating them

as stump classifiers, the worst an individual feature could do is 0.5 accuracy; any lower, and the classifier simply flips direction. Since there is noise in the data, a classifier that is uncorrelated with video content can still vary in value across videos, and this means that it is possible for it to obtain an accuracy better than 0.5 on the limited data simply by chance. If we were considering a single feature, then we could ignore this unlikely possibility, but with a pool of 10,000, statistically we can expect to see several such false positives.

It is easy to empirically estimate the false positive distribution by simply randomly permuting the labels of all of the test videos; in this way, a classifier cannot be legitimately correlated with the video labels, and the resulting distribution must be entirely due to false positives.

As can be seen in Fig. 11, the accuracy distribution of the real features is quite different from the false positive distribution. In particular, the real feature distribution is shifted to the right of the random distribution, indicating that there are more high-accuracy features than would be expected by chance, even in the worst-case scenario that the vast majority of the features are uninformative. Note that the false positive distribution takes on a log-normal type distribution, albeit with a spike at 0.5 corresponding to zero-variance features (in practice, it is easy to reject these features, even on real data, since they are features which return the same output for every input). The same test performed with the aggregation techniques produces similar results, indicating that the aggregation techniques also reveal informative features.

4.2.6 Comparison with other quantization methods

Since the goal of the proposed technique is to improve on the early quantization and accumulation steps of the bag-of-words model, a natural baseline against which to compare is the standard bag-of-words model consisting of k-means clustering followed by nearest neighbor quantization and histogram accumulation. Additionally, for the UCF-YT dataset we compare against the somewhat more sophisticated quantization technique proposed by Matikainen *et al.* [28].

Our results on the UCF-YT dataset are shown in Table 2. Here the feature shows large gains over both k-means and random feature subsets, at 46.0% to 37.0% and 40.2% respectively. Additionally, our feature selection improves upon by Matikainen *et al.*'s quantization technique, which obtains an accuracy of 42.2%. The power of our technique is further emphasized by the fact that our

Table 2: Results on the UCF YouTube dataset (motion features only).

Method	Total accuracy
Seeded RBF [STIP] ($g_{\max}$)	34.4
k-means [STIP]	36.6
k-means [Traj] (MSR centers)	36.6
k-means [Traj] (synth centers)	37.0
Seeded RBF [STIP] (g_{min})	38.6
Seeded RBF [Traj] ($g_{\max}$)	38.9
Seeded RBF [Traj] (g_{avg})	39.4
Unseeded RBF [Traj]	40.2, $\sigma = 1.9$
Trajectons [Matikainen *et al.* [28]]	42.2
All 10,000 RBF [Traj]	46.0
Seeded RBF [Traj] (g_{min})	46.0

Table 3: Comparison of seeding source on UCF YouTube.

Method / Source	UCF-YT	Synthetic
Seeded RBF [Traj] ($g_{\max}$)	38.6	38.9
Seeded RBF [Traj] (g_{avg})	34.7	39.4
Seeded RBF [Traj] (g_{min})	41.0	46.0

technique uses only 50 features compared to the 216 of Matikainen *et al.*'s trajectons. Even with a pairwise spatial relationship coding scheme, their technique achieves 47.7%, which is only slightly better than the performance of our independent features without any spatial information.

Note that our performance with 50 seeded features matches that of running the entire candidate set of 10,000 features. Beyond the obvious computational and storage benefits of processing only 50 features instead of 10,000, methods that build on top of these quantized features will likely benefit from the reduced dimensionality (*e.g.*, , if pairwise relationships are considered, it is better to consider 50×50 rather than 10000×10000). While the "kitchen sink" approach of feeding all 10,000 classifiers into an SVM worked in this case (likely due to the resilience of linear SVMs against overfitting), other classifiers (*e.g.*, , decision trees, randomized forests) may not be as robust.

The results of this comparison on the MSR dataset are shown in Table 4. Overall, the feature selection posts relatively large gains in the $g_{\max}$ and g_{avg} selection methods, while $g_{\min}$ remains largely the same as for k-means. For the individual classes, the selection method improves performance on the clap and box categories, while performance on wave is largely similar.

It is interesting that the selection techniques that perform well are exactly inverted between MSR and UCF-YT, with $g_{\max}$ and g_{avg} performing well on MSR, while $g_{\min}$ performs well on UCF-YT. In practice, g_{avg} works like a weaker $g_{\max}$, so it is unsurprising that its performance is similar to that of $g_{\max}$ on both datasets. Between $g_{\min}$ and $g_{\max}$, however, we suspect the difference is due to how similar the datasets are to the synthetic data that was used for feature selection. The MSR dataset is much more similar to the synthetic data than the UCF-YT dataset, which may explain why the more aggressive $g_{\max}$ selection performs better on the former while the more robust $g_{\min}$ selection performs best on the latter. More specifically, the MSR dataset has a fixed camera and simple human motions, which matches the cinematography of the synthetic data (albeit varying in the specific actions). By contrast, UCF-YT exhibits highly variable cinematography and includes non-human actions (*e.g.*, , horses and dogs) as well as actions with props (*e.g.*, , basketballs and bicycles).

Table 4: Seeding outperforms k-means, unseeded RBF, and boosting on MSR.

Method	Clap	Wave	Box	Total
Boosting (50/500)	65.0	60.0	57.0	60.7
Unseeded RBF	60.0	61.0	61.7	61.0
Seeded RBF ($g_{\min}$)	60.7	62.5	60.4	61.2
k-means (MSR centers)	53.6	62.5	68.7	61.6
k-means (synth centers)	53.6	62.5	68.7	61.6
Boosting (50/10000)	75.0	64.6	52.0	63.9
Seeded RBF (g_{avg})	71.4	62.5	66.7	66.9
Seeded RBF ($g_{\max}$)	75.0	58.3	70.8	68.0

4.2.7 Comparison of base descriptors

The results of a comparison of base descriptors (trajectories vs. STIP-HOF) is shown in Table 2. Overall, the performance of STIP-HOF features is worse than that of trajectory-based ones. However, note that the best selection method ($g_{\min}$) outperforms k-means for both STIP and trajectory features, and that $g_{\min}$ outperforms the other two methods for both features.

4.2.8 Comparison to unseeded RBF features

As an additional baseline we compare the performance of the features seeded from the synthetic data to that of random sets of features. The purpose of this baseline is to establish whether the gains seen with the classifier sets over k-means are due to the selection process, or whether the classifier-based features are inherently more informative than k-means histogram counts. As can be seen in Tables 2 and 4, the performance of random classifier sets is very similar to that of codebooks produced by k-means, indicating that random classifier sets are by themselves about only as powerful as k-means codebooks. It is only after selection (either on the data itself, if there is enough, or on synthetic data) that significant gains are seen over the k-means baseline.

However, despite the similar performance of unselected classifier features to k-means features, there is reason to believe that the classifiers themselves are a better choice for selection than k-

means based features. On both MSR and on UCF-YT the performance of k-means features derived from vastly different datasets (synthetic data vs. MSR) is very similar. Furthermore, experiments on synthetic data have suggested that the actual choice of k-means centers has only a small effect on performance. As a result, it is difficult to produce large gains by optimizing for k-means centers.

4.2.9 Comparison with feature selection on real data

We perform experiments using AdaBoost for feature selection on the MSR dataset (see Table 4). While boosting on the data itself improves performance on the clap action, the overall performance increase is modest, suggesting that when features are selected from the entire pool of 10,000 classifiers, boosting overfits. When the features are boosted from smaller subsets chosen at random, the overall performance is closer to that of unseeded features. However, the average performance of boosting on the real data is not much better than that of random subsets, and lower than that of seeded features.

Next, we evaluate the contribution of the synthetic data itself, in order to rule out the possibility that it is only the seeding technique (*i.e.*, , randomly partitioning the data into groups and then evaluating aggregate performance) that produces performance gains. We perform our feature seeding using the real training data as the seeding source. In order to mimic the structure of the synthetic data groups (one action class vs. everything else), we partition the UCF-YT training data into groups, where each consists of one action class vs. the remaining 10. We further randomly partition each group into five, for a total of 55 groups. We then perform the feature seeding. These results are shown in Table 3. Note that for every selection method (*e.g.*, $g_{\min}$), the seeding from synthetic data outperforms the seeding from the real data. Additionally, the selection method $g_{\min}$ is the best regardless of the seeding source. Thus, the synthetic data itself plays an important role.

4.2.10 Conclusion

This preliminary work demonstrates that even simple selection/recommendation mechnaisms can give a boost in performance from a corpus of dissimilar, but related, datasets. More tellingly, features selected from synthetic data have better performance than those selected from the real data, despite the similar sizes of the datasets, indicating that the synthetic data itself contributes

to the success of the technique. There are many possible reasons for this: the synthetic data may have more variation, it might help prevent overfitting, or it may simply be cleaner (*i.e.*, , there is no clutter or camera motion in the synthetic data, so it may select features that better represent the core human motions than those selected from the noisy real data). These results are promising, since they suggest that using a corpus of even curdely related labeled data can help on action recognition tasks.

The main drawback of this method is that including more synthetic data does not improve performance. The likely reason is that the simple aggregate statistics by which the features are selected (*i.e.*, , mean performance) converge quickly, and so past the point of convergence, additional data cannot offer an improvement. Another possible reason is that additional synthetic starts to overfit the features to the particular *global* biases of the synthetic data. The discovery of this limitation is one of the main motivations for the proposed feature recommendation framework: rather than attempting to find features that are generally good across *all* the synthetic data, we propose to use the corpus of datasets to tailor features *specifically* for the target dataset.

4.3 Proposed Methods

In order to predict the ratings of features on a new target task, we must first build a model of the relationships between the ratings of features on tasks in the ratings store. Our preliminary work on feature seeding can be seen as assuming a simple model where each feature has a true rating which is estimated with noise on each task, so that the predicted rating for a feature is simply its average (or other aggregate statistic) rating across all the tasks in the store.

In this section, we introduce some basic collaborative filtering techniques for modeling ratings.

4.3.1 Baseline estimation

We start with a simple additive model suggested by Koren [83], in which a feature's rating on a task is modeled as the sum of a global mean rating, a feature factor, and a task factor. Intuitively, the feature factor approximately corresponds to the feature's mean deviation from the global mean across all tasks, and likewise the task factor is approximately the average deviation of all features rated on that task from the global mean.

Formally, we model a rating $r_{i,j}$ as

$$r_{i,j} = \mu + \phi_i + \psi_j, \tag{10}$$

where μ is a global mean rating, ϕ_i is a feature-specific factor, and ψ_j is a task-specific factor. Let m be the number of features, and n be the number of tasks, so that the number of ratings is $m \cdot n$.

Koren [83] suggests that these factors can be found via a least-squares minimization. While true, for a dense ratings matrix (as is the case with our synthetic data, where every feature is rated on every synthetic task) this produces an inconveniently large number of terms (*i.e.*, one for each rating, which might be on the order of millions). Instead, we propose the following iterative method, which is simply the iterated version of the approximation suggested by Koren:

- 1. Estimate global mean: $\mu = \frac{\sum_i \sum_j r_{i,j}}{mn}$
- 2. Estimate initial feature and task factors: $\phi_i = \frac{\sum_j (r_{i,j} - \mu)}{n}$, $\psi_j = \frac{\sum_i (r_{i,j} - \mu)}{m}$.
- 3. Estimate feature factors, holding task factors constant: $\phi_i = \frac{\sum_j (r_{i,j} - \mu - \psi_j)}{n}$
- 4. Estimate task factors, holding feature factors constant: $\psi_j = \frac{\sum_i (r_{i,j} - \mu - \phi_i)}{m}$
- 5. Iterate: Repeat steps 3 and 4 until convergence.

In practice, we find that the factors converge very quickly, often in one or two iterations, and that the factors do not differ much from the initial independent estimates. Note that each step is simply the computation of a mean residual.

For the target task, we hold the previously computed feature factors fixed, and estimate only the target task's factor, according to

$$\psi_{j^*} = \frac{\sum_i (r_{i,j^*} - \mu - \phi_i)}{m}, \tag{11}$$

where j^* is the index denoting the target task.

4.3.2 Factorization methods

First, let us define the residuals $\bar{r}$ that remain after the baseline estimation by

$$\bar{r}_{i,j} = r_{i,j} - (\mu + \phi_i + \psi_j). \tag{12}$$

Then let $\bar{R}$ represent the entire $(m \times n)$ residuals matrix for the source tasks.

The general goal of factorization methods is to represent the rating (or residual thereof) of a feature on a task as the dot product between a feature factors vector and a task factors vector, where the dimensionality of these factors vectors corresponds to a chosen number of k latent factors. Formally,

$$\bar{R} = F^T D, \tag{13}$$

where F^T is a $(m \times k)$ matrix of feature factors, and D is a $(k \times n)$ matrix of task factors. While there are many factorization schemes, a simple and popular choice is to use the singular value decomposition, in which

$$\bar{R} = USV^T, \tag{14}$$

where U and V are orthonormal matrices, and S is a diagonal matrix of singular values. Then, supposing that k latent factors are sought, S_k is the $k \times k$ upper left sub-matrix of S, and U_k is the first k columns of U. We construct the feature factors matrix according to

$$F^T = U_k S_k, \tag{15}$$

and the task factors matrix can be recovered by solving the linear problem $\bar{R} = F^T D$ for D.

Now, given a target task's ratings of p probe features (without loss of generality assume they are the first p features), denoted r_p, we estimate the target task's factor vector by solving the linear least-squares problem

$$(\hat{F}^T)x = r_p \tag{16}$$

for x, where x is a $(k \times 1)$ vector of the target task's factors, and $\hat{F}^T$ is a $(p \times k)$ matrix of the first p rows of F^T. Note that in order to keep the problem overconstrained (so that the least-squares minimization does something useful) $p \geq k$, which is to say that more probe features should be evaluated than latent factors. Finally, we can predict the target task's ratings for all the features according to

$$r' = F^T x, \tag{17}$$

where r' are the predicted ratings.

4.3.3 Neighborhood methods

Neighborhood models form a large class of methods for collaborative filtering; here we briefly describe a simple method invented by Bell and Koren [81].

The goal is to predict the rating r_{q,j^*} of feature q on the target task. Let U be the number of *neighboring* features to the query feature. These neighbors might be found by, for example, the correlation coefficient between task ratings vectors [82]. Note that here we are considering features to be neighbors to features, rather than tasks to tasks.

Then, let S_p be the matrix of ratings of the neighbor set of features over the tasks, and S_q be the ratings of query feature on the other tasks. Likewise, let R_p be the vector of ratings of the probe set of features on the target task.

The simplest neighborhood method predicts the rating of a query feature q on the target task as the mean of the neighbors' ratings:

$$R_q = \frac{\sum_i S_{i,q}}{|S_q|}. \tag{18}$$

Note that this does not use the relative distances of the neighbors. A more sophisticated approach, as suggested by Bell and Koren [81], is to calculate interpolation weights w_i per neighbor, so that the predicted rating is expressed as

$$R_q = w^T R_p, \tag{19}$$

which is simply a weighted sum of the neighboring features' ratings.

They calculate the interpolation weights w by constructing matrices

$$\hat{A} = \frac{S_p S_p^T}{U}, \tag{20}$$

and

$$\hat{b} = \frac{S_p S_q^T}{U}, \tag{21}$$

and solving the linear system

$$\hat{A} w = \hat{b} \tag{22}$$

for the interpolation weights w.

The predicted rating of the query feature on the target task is then computed according to Eq. 19.

Set-centric vs. Feature center neighborhoods:

An important question for neighborhood methods is whether the neighborhood is over tasks, or over features. That is, to predict an unknown rating,

we could either find similar features to the feature in question, or we could find similar tasks to the task on which the feature is evaluated. The former is a *feature-centric* approach, while the latter is *task-centric*. In the collaborative filtering world, the analogous distinction is between user-centric and item-centric approaches, and generally in the literature item-centric approaches have performed better. A possible explanation is that in standard recommender systems, there are vastly more users than items, making item-item comparisons more robust than user-user comparisons. While the intuitive description of our problem metaphorically treats tasks as users, in practice our situation is the opposite of that in typical recommender systems: we have vastly more features (items) than tasks (users), and so we might expect that task-centric approaches would fare better. However, there is a vast difference between consumer items and vision tasks, and so this question must be answered experimentally (see Section 4.5.3).

4.3.4 Metrics

RMSE:

The simplest way to evaluate the feature recommendation is to directly compared the predicted feature ratings against the 'ground truth' feature ratings using a root-mean-square error metric. For early evaluation purposes this metric can work well as a rough guide to which collaborative filtering techniques are likely to be the most productive. However, the RMSE may not accurately reflect the final accuracy of the predicted features, and so is unsuitable as a final evaluation metric.

Set intersection:

If the number of features to be recommended is known, then another simple evaluation metric is to compare the set intersection (number of features in common) between the recommended feature set from the collaborative filtering, and the set chosen on the real data, either from the ground-truth

feature ratings, or using another feature selection method directly on the real data. The difficulty with this metric is evaluating the ground-truth ideal feature set– for a finite task, the set of features that maximizes classification accuracy is almost certainly overfitting, while for an infinite task (*e.g.*, a synthetic set), finding the ideal feature selection is itself a difficult search problem.

Classification accuracy:

Rather than attempting to measure the quality of a feature selection directly, the final accuracy on a test classification problem can be measured. This has the benefit of measuring the actual performance of interest (the end classification accuracy). However, this requires cleanly separated testing and training tasks, and so requires more data than other evaluation metrics.

4.4 Preliminary Experiments

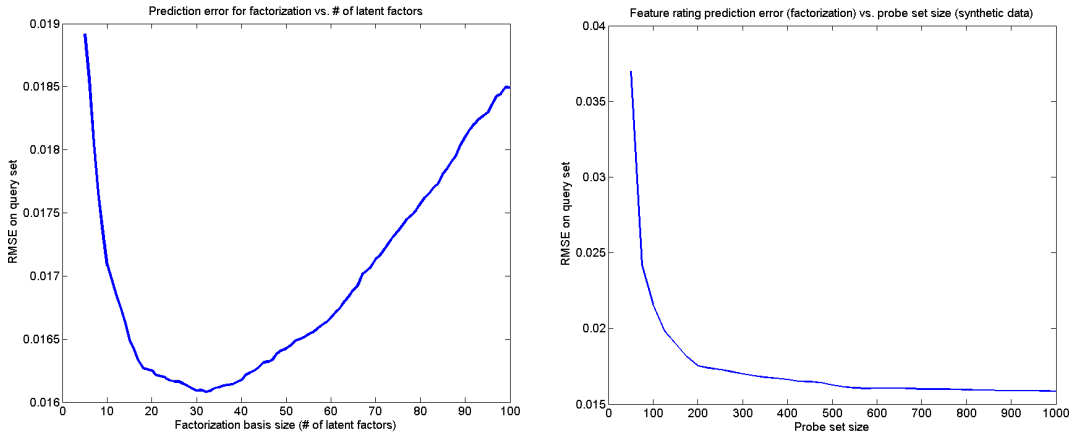


Figure 12: Preliminary experiments with collaborative filtering. Results for predicting ratings of features on synthetic tasks using a ratings store of other synthetic tasks.

As simple preliminary experiments we consider the ability to predict the ratings of features on synthetic tasks from a corpus of similar (but of course, not identical) synthetic data. For these experiments, we use a corpus of 100 synthetic tasks, and evaluate over 100 target (synthetic) tasks. We consider a candidate pool of 10,000 features of the RBF histogram classifier count variety, randomly generated. We evaluate using the RMSE metric on the features outside of the probe set. These results can be seen in Table 5. Note that the best performance is produced by the

Table 5: RMSE error for predicted feature ratings evaluated on synthetic data

Method	RMSE
Mean only	0.0558
Baseline estimation	0.0326
Nearest neighbor	0.0167
Factorization	0.0163

factorization approach, which slightly outperforms the neighborhood method in terms of RMSE and also has a greatly reduced computation time. As further experiments, we vary the probe set size and number of bases (latent factors) chosen for the factorization approach; these graphs can be seen in Fig. 12. The RMSE exhibits an apparent exponential decay with the probe set size, with the majority of the reduction having occurred with approximately 500 probe features. In contrast, the RMSE reaches a minimum with 30 latent factors, with the curve exhibiting a striking 'U' shape. Note that 100 latent factors is the most that could be chosen, since there are only 100 tasks in the corpus.

4.5 Experimental Plan

For experimental evaluation, we initially consider the action recognition task, on any of the appropriate datasets. We build the ratings store from synthetic data, generated according to Section 3.3.

4.5.1 High vs. Low variation synthetic data

An important question is how the amount and types of variation in the synthetic data affect the quality of the recommendations. In some cases the answer should be obvious, for example, that recommendations from synthetic data that includes running actions are likely to be better on a real task that involves running. However, some cases, such as camera motion, are not as clear-cut. These types of questions are easily evaluated by restricting recommendations to being computed from synthetic tasks with varying amounts of variation, and by evaluating on datasets that contain elements (such as vehicles that are not present in the synthetic data.

4.5.2 Lightweight vs. Heavyweight features

The features we have a wide range of granularities, in that they can span the range from very specific (the count in a single histogram bin for a particular quantization scheme) to very broad (the entire histogram). An interesting question is whether it is better to recommend from a large pool of very granular (lightweight) features, or from a smaller pool of more powerful (heavyweight) features.

4.5.3 Task-centric vs. Feature-centric approaches

Neighborhood methods might use neighborhoods of tasks (task-centric approaches), or they might compute neighborhoods around features (feature-centric approaches). The question of which is preferable remains to be answered experimentally.

4.5.4 Content vs. Performance matching on Similar vs. Dissimilar data

As a baseline we can compare the recommendations based on collaborative filtering on *ratings* to features recommended from neighboring tasks based on *content*. That is, collaborative filtering finds neighbors where the probe set of features performs similarly in terms of classification accuracy, but we can also consider finding neighbors where the probe set performs similarly in terms of actual computed values. Note that in the former (accuracy based) case, the neighbors are entire *tasks*, while in the latter (content based) case, the neighbors are individual *data samples* (video clips). This is because we evaluate the accuracy rating of a feature across an entire set of samples, while the actual value of a feature is computed for every individual data sample.

To make such recommendations based on content, each real training sample would find its nearest k neighbors in the synthetic data, and those found neighbors would take on the class label of the real training sample. Then these found neighbors are pooled together into a new, larger dataset which is used to either recommend features, or to directly learn a classifier. Alternatively, these found samples could be used as an additional task for multi-task learning, so that the goal is to find features that perform well on both the real data, and on the neighbors to the real data.

4.5.5 Ideal rating estimation on synthetic data

We have already discussed how the actual measured feature ratings on a real-data task are actually estimated ratings of true or ideal feature ratings. Furthermore, we hypothesized that the collaborative filtering predicted ratings may be closer to the ideal ratings than the estimated ratings, and gave preliminary results tentatively supporting this hypothesis. Of course, as the size of the dataset associated with a task grows, the estimated ratings converge to the ideal ratings, by definition. Thus, we can directly test this hypothesis by estimating the ratings twice: once using a smaller training partition, and a second time using as much data as is available. If the hypothesis is true, then the ratings predicted by collaborative filtering using the training partition ratings should be closer to the full-dataset ratings. For synthetic data we can generate enough data to estimate the ideal ratings to any arbitrary degree of precision.

4.5.6 Effect of store size

While collaborative filtering techniques can easily scale to hundreds of thousands of tasks, the benefit of doing so remains to be evaluated. Will performance continue to rise indefinitely, or will it plateau and then degrade as the number of tasks in the store increases?

4.5.7 Recommendations from limited target data

A potential benefit of the feature recommendation approach is that it may be able to make recommendations even when the size of the target dataset is very small (*i.e.*, the estimated ratings on the target are very noisy). An interesting question is how small the target dataset can get, and how the performance of the recommended features at these small dataset sizes compares to other feature selection techniques, or even hand-picked features.

4.5.8 Evaluating the impact of redundant features

A major concern is that independently recommending features will produce a feature set populated by near-duplicates of the best features, rather than a more varied (and better performing) set. There are two avenues for testing whether this is the case. The first is to compare against traditional

feature selection mechanisms (*e.g.*, boosting) that select feature sets taking into account interdependencies. The second is to generate near-duplicate features to evaluate the costs of redundancy.

4.6 Random candidate features vs. Learned candidate features

Our preliminary work uses a randomly generated candidate pool of features, but another possibility is to use the detailed annotations of synthetic data to populate the pool with features that have been learned to try to detect or classify elements of that annotation. For example, if the synthetic data has annotations on the individual positions of limbs, then features could be learned to try to detect those limbs. These features would be learned from random samples of the given limbs, so that many different learned features could be generated for the same limb. While a feature pool generated in this way may be less likely to contain useless features, it may also overfit to the synthetic data, and that is a question that must be answered experimentally.

4.7 Research Topics

We have identified the following key research topics:

- Set recommendation: Rather than recommending features individually, an interesting question is whether entire *sets* of features can be recommended jointly. Such a system would likely need to model the dependencies between features included in a set, and potentially result in interesting new theoretical models. If feature redundancy is experimentally found to be detrimental, we will need to investigate set recommendation in order to fulfill our proposal contribution of scaling to hundreds of thousands of features

- Missing ratings: Our described approach assumes complete ratings for all the features on all the tasks in the store. However, collaborative filtering has developed many techniques to deal specifically with missing ratings. We may be able to avoid rating all features on all items in the store, potentially allowing for a larger store by decreasing the computational cost of each feature and task. More intriguingly, we could consider mixed-modes tasks, where some features are invalid on some tasks (*e.g.*, image features are invalid on a motion capture task). Another interesting question is, given that we can rate features on tasks on demand,

whether we can intelligently choose, on-line, which features to rate on which tasks next, in order to produce the most useful ratings store. If we find that the computational cost of fully evaluating all features on all tasks is too large, we will need to pursue this research topic in order to be able to scale to the promised large numbers of features and tasks.

- On-line recommendations: We have already suggested that it may be possible to make recommendations for very small target datasets. Additionally, collaborative filtering techniques, built for retail recommendation to end users, are often designed to be very fast. The combination of these two traits suggests that we may be able to make on-line recommendations for features. For example, given just a few frames of a person's face from a live camera, could we recommend a set of features that would then track that person well? Given a few instances of a person performing an arbitrary gesture live, could we learn features to subsequently detect that feature?

- Time-constrained feature recommendation: We can consider adding constraints, such as run-time constraints, to the recommendation system, so that it can produce recommendations fulfilling a certain *time-budget*. All too often vision algorithms are proposed with little concern for their run-times, beyond the researchers being able to run sufficient experiments to populate their papers. This means that computationally efficient features are frequently compared against expensive ones, to the detriment of the faster features. Yet in the same time it takes to evaluate an expensive feature, many cheap ones could be evaluated, and this raises the question of whether it is better to build an ensemble classifier of cheap features rather than using a monolithic expensive feature. Our proposed feature recommendation framework gives us an algorithmic way to answer that question, since we can use time-constrained feature recommendation to recommend the best set of features which jointly meets some time constraint.

4.7.1 Active task generation

So far we have discussed the tasks used for filtering as if they are fixed. However, since the tasks used for filtering are synthetically generated, we can consider actively generating more tasks to improve the predictions for a given real task. Suppose that each synthetic task was generated by

a universal generator parametrized by θ, so that each generated task T_j is associated with the parameter vector θ_j used to generate it. Then, given a real dataset T', the goal is to produce more datasets in the neighborhood of T'.

The simplest approach is to find neighbors of T' in performance space, and then generate more datasets by randomly perturbing the θ vectors of those neighbors.

Assume that T' corresponds to some parameter vector θ' that would produce a dataset with the performance characteristics closest to D'. Then, suppose that the performance distance between D' and a given dataset T_j is given by

$$d_p(T', T_j) = (\theta' - \theta_j)^T Q(\theta' - \theta_j) + C, \tag{23}$$

where Q is a symmetric positive semi-definite matrix and C is a constant. This model assumes that the distance in performance space between two datasets is their distance in parameter space after some non-uniform scaling. The parameter vector θ' is recovered by fitting a quadratic. After θ' has been estimated, more datasets can be generated using that parameter vector, and the process can be iterated if desired.

4.7.2 Probe set selection

Up to this point we have simply assumed that on the target dataset some features have been rated, without paying particular attention to how that set of rated features was chosen. In recommender systems for humans, this is a reasonable assumption, because the systems have little control over what people choose to rate. However, in our case we do not have to passively accept some externally driven set of ratings, but rather we can actively choose which features are chosen to be in the probe set. An interesting problem is then how to choose our probe set to maximize the accuracy of our predicted ratings on the remaining items.

4.7.3 Constrained feature selection

Suppose that we wish to select the best set of features under some computational constraint (*e.g.*, limited computation time). We model the cost of our feature selection as the sum of individual,

or marginal, feature costs c_i, and group, or fixed, costs g_k. The marginal cost is incurred on an individual feature basis, while the group cost is only incurred once if any of the features in that group are chosen. For example, if we have a mix of candidate features, so that some are features computed from STIP descriptors, while others are features computed from trajectories, then all the STIP features would belong to a group whose cost was the time to process video for STIP descriptors, and likewise for the trajectory based features. The individual cost of each feature would be the marginal time to compute that feature from the pre-computed descriptors.

Then, we can formulate the problem of choosing features as a mixed integer linear program. Let the variable $s_i = 0, 1$ indicate whether feature i is chosen, and r_i correspond to the predicted rating of that feature. Then, we introduce intermediate variables z_k indicating whether a group k has any features chosen from it, which is expressed through constraints as

$$\forall i \in g_k z_k \geq s_i. \tag{24}$$

The objective function is simply to maximize the sum of the ratings under the constraint that the total costs are less than a cost budget C, so that

$$\max \sum_i r_i s_i, \tag{25}$$

and

$$\sum_k z_k g_k + \sum_i s_i c_i \leq C. \tag{26}$$

Note that this is a variant of the knapsack problem, and is NP-complete to solve exactly. However, there are reasonable approximation algorithms, and modern integer program solvers can often deal with problems of this size despite their worst-case complexity.

4.7.4 Feature set recommendation

In order to recommend sets of features rather than features independently, some model must be assumed for how features affect each other's accuracies. For example, we might consider using a pairwise model in which the total accuracy is some function of the sum of the individual feature accuracies, plus terms accounting for pairwise interactions between features. The downside to such a model, and indeed to most models, is that the number of parameters to be estimated is very large,

on the order of the square of the number of features. Since the proposed work considers candidate pools of 10,000+ features, this is a prohibitively large number of parameters to estimate. Simpler models will need to be sought.

One straightforward simplification is to consider pairwise effects between *groups* of features. Features would be clustered into a greatly reduced number of clusters based on their individual ratings, and pairwise effects estimated only between clusters.

Group accuracy model:

In the absence of any more compelling theoretical model, we propose to model the accuracy of a set of features be modeled as

$$a_i(F) = 1 - e^{-\alpha_i Z(F)}, \tag{27}$$

where $Z(F) = \sum_{f \in F} a(f)$, and α_i is a task-specific decay parameter. This models simple exponentially decaying marginal returns from additional features. The immediate consequence of this simple model is that, ignoring feature synergies and redundancies, the best set of k features to select is simply the top k rated. It is simple to experimentally validate this model.

4.8 Risks

Here we identify major risks and how we plan to mitigate them.

- Feature correlations only valid on tasks in store: A possibility is that the tasks in the store are simply too distant from the target task, and so the feature performance correlations in the store do not hold for the target. Possible ways to mitigate this are to seek out store data that more closely resembles the target, such as by using real data (YouTube clips), or using active synthetic generation to try to tune the type of generated synthetic data to the target.

- Feature correlations are weak: A more likely scenario is that the performance correlations between features are weak. This is more likely using lightweight features which already have poor individual performance. Ways to mitigate this include considering heavyweight features, which are more powerful, and more likely to correlate, and using more sophisticated machine learning at the end combined with larger recommendation sets. For example, a large set of

features could be recommended, and then an additional layer of feature selection or multi-task learning used to further prune that feature set on the real data.

- High computational costs: A practical concern is that evaluating hundreds of thousands of features on thousands of tasks may be too computationally expensive to be practical. To mitigate this, computational cost must be a priority in feature implementation. Additionally, we can consider using and designing features to take advantage of shared computation wherever possible (such as motion features all using the same base optical flow calculation results).

5 Proposed Work: Layered Bag-Of-Words

In the second part of our proposed work we suggest how hierarchy might be added on top of generic features by using data sources with more detailed annotations (*e.g.*, synthetic data) to guide the generation of mid-level features. In particular, we introduce a framework that builds a layered representation on top of standard bag-of-words techniques; we refer to this method as *layered bag of words* (LBOW). The organization of this section is the same as that of the previous: we begin with an overview and introduction, proceed to established methods, follow with an experimental plan, and conclude with directions for research and risks.

In the previous part we described how a store of information (feature ratings) built from heterogeneous tasks could be used to help select or recommend features for a vision task. However, a system built from simple features recommended in such a fashion is still fundamentally 'flat', that is, that features do not really build on one another, but are instead independently evaluated and only combined at the end. It is our suspicion that such flat systems will be always be limited in their predictive power, and that eventually truly layered systems will be necessary to tackle the most challenging vision tasks. In this section we describe a framework which augments the traditional bag-of-words framework into a hierarchy. We suggest that randomly generated layered bag of words classifiers could themselves be used as input to our previous proposed work of feature recommendation; when used in this manner, many such models would be recommended and then combined into an ensemble classifier, similar to the screening approach of Pinto *et al.* [89]. Additionally, we suggest that the intermediate layers might be learned as features that target more detailed annotations (*e.g.*, part labels) available in some datasets, but not the target task. For

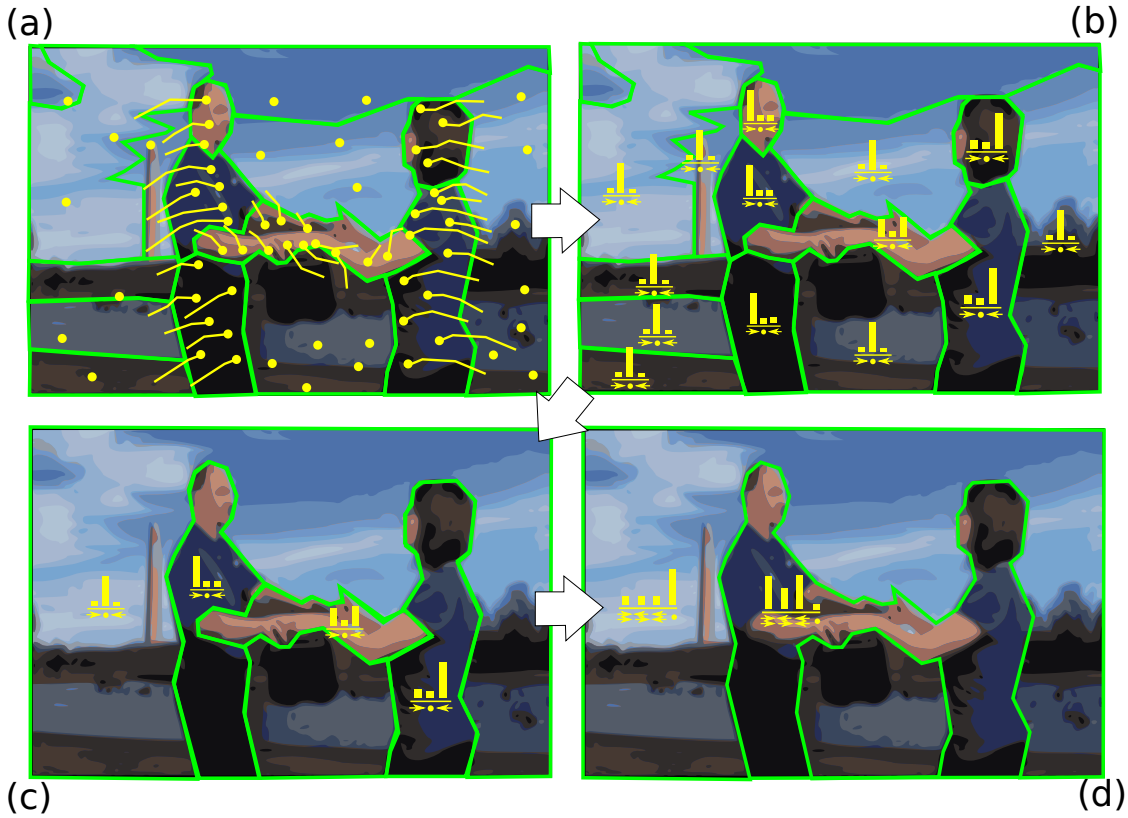


Figure 13: Each layer of layered bag-of-words reduces the set of descriptors by first partitioning the set of descriptors into groups, and the producing one descriptor from each group which summarizes the content of that group. Here descriptors (in this case, image region based descriptors) are successively merged and then described with histograms.

example, the detailed, pixel-level annotations of synthetic data might be used to learn mid-level features in a layered representation. In this way we can fulfill our stated contribution of being able to take advantage of differing levels of annotation detail.

5.1 Notation

In this section we refer to *descriptors*, which are distinguished from the *features* of the previous part (feature recommendation) chiefly by being variable in number. That is, each feature of the previous section is an opaque function that computes some value from a data sample (video); the internal workings of the feature are hidden, but it has known, fixed input and output dimensions.

In contrast, a descriptor is also produced from a data sample, but data samples may produce variable numbers of descriptors. For example, a corner detected by a Harris corner detector is (in this terminology) a descriptor, while the *count* of corners in a video is a feature. The Harris corner detector itself might be termed a *descriptor extractor*, but we will often refer to descriptor extractors simply as descriptors, letting the part stand in for the whole. While the number of descriptors produced by the extractor may be variable, each individual descriptor is still a fixed size vector.

A frequently occurring case is that descriptors contain two parts: a spatio-temporal location in a video, and some other, descriptor-specific, part. For example, a corner detector produces descriptors which are described by their location and (potentially) their strength. STIP [4] interest points likewise have a location, and in addition a HOG/HOF descriptor of the video around that location. Although the location is simple a tuple of numbers, and could thus be considered part of a generic descriptor, this type of localized descriptor occurs sufficiently frequently that we give it special treatment. We refer to these types of descriptors as *local descriptors*.

Note that a descriptor (that is, a descriptor extractor) may be easily turned into a feature by simply counting the number of descriptors produced on a video. Less easily, but still generically, a descriptor may be turned into a feature by first quantizing the produced descriptors, and then producing a histogram of those quantized descriptors. The inverse process of turning features into descriptors is not as straightforward.

As another special case, often unsupervised image segmentation methods (for example, segmenting the video into superpixels/supervoxels) are used to produce image regions. The image data in the region can be considered in any function that takes as input a descriptor, and likewise, when descriptors are grouped together, the resulting descriptors region should be the union of all its component descriptors. For image/video regions, adjacency is often an important concern (*e.g.*, only adjacent regions should be able to merge), and we denote the set of adjacent regions by N, so that $\{i, j\} \in N$ if d_i and d_j have regions adjacent to one another. In the case of descriptors without associated regions, N can be assumed to be fully populated by all pairs.

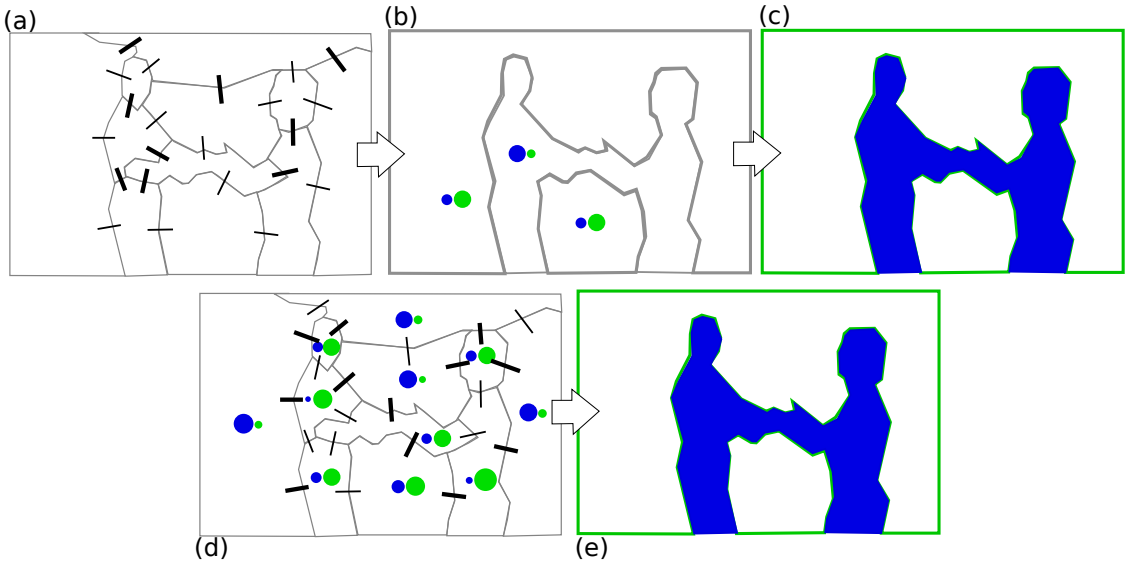


Figure 14: Independent vs. joint grouping and description. In the top row, descriptors (here, image regions) are first grouped according to similarity between adjacent descriptor regions (thicker lines indicate higher similarity). Each merged region is then described with the best label for that region, each label represented by a colored dot, its affinity for that region represented by its size. In the bottom row, descriptors are grouped and described jointly, using an energy minimization optimization. Line thickness indicates the *cost* of grouping descriptors together (assigning them the same label), while dot sizes indicate costs for assigning labels to descriptors. The label assignment minimizes the sum of the costs, or the energy. Note that the pairwise costs (potentials) can depend on the labels as well, however this is difficult to illustrate.

5.2 Methods

We introduce a framework we denote layered bag of words. This framework is very generic, capable of representing a vast range of possible layered techniques. As such, it is not meant to be taken as a practical suggestion for how a system might be constructed, but rather as a theoretical guide in which to consider various possibilities for the inclusion of hierarchy.

Traditional one-layer bag of words can be seen as a two-stage process, in which first all the descriptors are grouped into one set, or collection, and then that collection is described using k-means to summarize its contents. Layered bag of words extends this model to multiple layers, in which each layer consists of a grouping stage, in which the current descriptors are grouped into a variable number of collections, and a description stage, in which each collection reduced to a

descriptor (see Figure 13). Each such layer serves to aggregate and transform the information in the scene, until an optional final layer collapses all the remaining descriptors down into a single feature which can be used in many traditional machine learning techniques.

As a simplified example, consider Figure 13. The base descriptors are trajectory snippets extracted from the scene with an image patch tracker (KLT tracker [101]), as in Appendix A. In the first grouping stage, a superpixel segmentation on the image is used to group nearby trajectories together. These resulting collections are then described by quantizing the trajectories into *stationary*, *moving left*, or *moving right*, and then computing a histogram. Thus, after the first layer the scene is composed of a number of descriptors, where each descriptor is itself a histogram. In the next layer, descriptors are first grouped by merging adjacent descriptors with similar histograms. Then, the collections produced by this merging process are described by simply summing their histograms. In the third layer, descriptors are merged according to a more complicated rule which allows descriptors representing motion to merge, while discouraging merges between descriptors containing motion and those without. These collections are then described by quantizing the histograms into four categories: *stationary*, *leftward motion dominant*, *rightward motion dominant*, *inconsistent motions*.

In order to constrain the flexibility of this framework into a form that may be more manageably investigated, we consider two possible approaches to building grouping and description stages– independent grouping and description, in which the grouping stage and description stage proceed consecutively, governed by independent mechanisms, and joint grouping and description, in which optimization is used to jointly group and describe the descriptors.

5.2.1 Independent Grouping and Description

We treat the problem of grouping the descriptors in the scene, considered independently of describing them, as a clustering problem. As such, many different techniques can be used. In the simplest case, we can use k-means or mean-shift clustering on the descriptors (if the descriptors are local descriptors, then this will take into account spatio-temporal proximity). However, in a more general case we can define an arbitrary distance function $g(d_i, d_j) : \mathbb{R}^{\ltimes} \times \mathbb{R}^{\ltimes} \to \mathbb{R}$ between two descriptors and then; we can cluster the descriptors in this case using spectral clustering methods [102]. Note

that since spectral clustering techniques cluster nodes (descriptors) with stronger *similarity*, rather than lower distance, we must transform our distance metric into an affinity, for instance using a Gaussian:

$$s(d_i, d_j) = \exp g(d_i, d_j)^2/(2\sigma^2), \tag{28}$$

where σ is a parameter controlling the relationship between similarity and distance. Of course, we can learn or randomly generate the similarity function $s(d_i, d_j) : \mathbb{R}^{\ltimes} \times \mathbb{R}^{\ltimes} \to \mathbb{R}$ itself.

For the description stage, initially we propose to either again use clustering or vector quantization techniques on the descriptors, and from the quantized descriptors produce histograms, or to use the generalized histograms we introduced in Section 3.4.1. The former (histogram generation from clustering) has the benefit of reducing the need for annotated data, while the latter (generalized histograms) is more flexible. The latter technique can benefit from detailed annotations in training data, since the screening/filtering process used in the preliminary work (Section 4.2) can be applied to select a quantization scheme for the generalized histogram that best allows the prediction of the detailed annotations. This process is illustrated in the top row of Fig. 14.

Using unsupervised learning for the intermediate layers is not without precedent, as Pinto *et al.*take this approach in [89], using unsupervised techniques to learn the middle layers of models. Interestingly, in a later work [1] they abandon unsupervised learning in favor of *random* intermediate features.

5.2.2 Joint Grouping and Description

We can combine the grouping and description stages by formulating them as an energy minimization problem. Energy minimization methods are a popular choice for image segmentation [47]. In this formulation, the goal is to assign a label l_i to each descriptor $d_i \in D$, such that the total energy is minimized, where that energy is formulated as

$$E = \sum_{\{i,j\} \in N} V_{i,j}(l_i, l_j) + \sum_{d_i \in D} U_i(l_i), \tag{29}$$

and N is the set of interacting descriptors, $V_{i,j}$ is a pairwise penalty, and U_i is a unary penalty. If we have no supervised data for intermediate features, we can simply consider randomly generated

penalty functions. That is to say, we can consider the unary penalties to be functions of the form

$$U_i(l) = f_l(d_i), \tag{30}$$

which is to say that each *label* is associated with a function f_l that computes the penalty of assigning that label to a given descriptor. This function might be randomly generated as a RBF classifier, as per Section 3.4.1. This energy minimization approach is illustrated in the bottom row of Fig. 14.

We could use the same strategy for the pairwise penalties, creating one penalty function for each possible pair of labels, however that is likely to result in an excessive number of functions (*e.g.*, with 100 labels, this would require 10,000 functions). Instead, we might consider using only one universal pairwise penalty function,

$$V_{i,j}(l, m) = (g(d_i, d_j) \cdot I(l \neq m)) + v_{l,m}, \tag{31}$$

where $g_(d_i, d_j)$ is a penalty function for assigning different labels to the descriptors d_i and d_j, and $v_{l,m}$ is a penalty term for assigning two neighboring descriptors the labels l and m. In this formulation, it is only necessary to generate (or learn) the single penalty function g, and generate (or learn) the pairwise label penalty matrix V, where $V_{ij} = v_{l,m}$. The function g can be randomly generated as an RBF classifier, and the matrix V can simply be generated as a random symmetric matrix with all positive entries.

With detailed annotations, it is also possible to learn the pairwise and unary penalties, for example, by treating the problem as a conditional random field (CRF). CRFs are frequently used for such purposes [43], where the goal is to label pixels of the image from a set of labels, and where supervised training data is available to learn the probabilistic association between local/region descriptors and labels.

A further possibility is to learn the pairwise costs between labels using a formulation similar to that of the pairwise descriptors we considered in [28]; in this case,

$$V_{i,j}(l, m) = -\log(P(d_i = l, d_j = m, x_i, x_j)), \tag{32}$$

where x_i and x_j are the locations of the local descriptors d_i and d_j, respectively, and the the probability term $P(.)$ can be represented and learned in a manner similar to that of our previous work [28].

5.2.3 Feature distance or similarity from synthetic data

Either of the two previous approaches can take advantage of a distance or similarity function between descriptors. With sufficiently detailed annotated data, these types of functions can be learned directly. Suppose the synthetic data has parts level annotation, so that it is possible to tell whether two local descriptors belong on the same part. Then, given input local descriptors, we can simply use supervised learning to learn a classifier that determines whether those local descriptors are present on the same part. With appropriate transformations (*e.g.*, inverted in some way for distance), the output of this classifier can be used as a distance or similarity function.

This method is intuitively appealing because it tries to group features according to the parts-level semantics of the training data. Additionally, it is relatively simple, and the training data is readily available. However, care must be taken in the actual clustering from this affinity, since the learned classifier is likely to produce very noisy results. Additionally, the relative inflexibility of this method may be detrimental for the lower layers, where there might not actually be sufficient information in the descriptors to learn such distance / similarity classifiers with any accuracy.

5.2.4 Iterative Screening

In the previous two sections we suggested that the grouping and description layers could either be randomly generated, and then the whole randomly generated layered representation could be treated as a feature for feature recommendation, or that layers could be learned using detailed annotations (for instance, from synthetic data) as a guide, using standard learning techniques. As a third possibility, we suggest an approach we call *iterative screening*, continuing in the line of [89].

In iterative screening, at each iteration a number of random new candidate layers are generated. Each candidate is evaluated by constructing a layered representation consisting of the previous layer stack, followed by the candidate layer, followed by a stage that groups all the descriptors together and then describes them using histograms of k-means labels. The best new candidate layer, that is, the one with the highest aggregate performance on the evaluation data (synthetic, or real but plentiful) as per Section 4.2, is accepted, and the total representation becomes the previous layer stack combined with the new candidate layer.

In this way, the layered representation grows by one layer each iteration, where that layer is

greedily chosen from a random candidate pool to maximize performance on a number of training tasks. Growth can be terminated when either a desired number of layers is reached, or when performance has peaked when measured on separate validation data.

5.3 Experimental Plan

For evaluation of these layered methods we focus on the task of action recognition, since compared to the face recognition task we have better synthetic data from which to learn intermediate features, and more diversity in tasks. The metric is simply performance on the target task; unlike feature recommendation, there is no middle result (like feature ratings) to evaluate. The evaluation datasets can be UCF-YT, MindsEye, and potentially Rochester Daily Living, although the standard caveats about that dataset apply.

5.3.1 Random model generation vs. Guided learning vs. Iterative screening

A main experimental goal is to determine which method of intermediate layer generation produces the best results when built on synthetic data and then applied to real evaluation tasks. In random generation, full LBOW models are randomly generated and then used as features in feature recommendation to produce an ensemble classifier. In guided learning, the LBOW models are generated using supervised learning to produce intermediate layers that attempt to find the parts present in the synthetic data annotations. In order to build a candidate pool of models, this procedure can be repeated many times, using different sets of synthetic data to guide the intermediate layers. For initial experiments, these parts will simply be limbs. Iterative screening attempts to iteratively screen for LBOW models that perform well on the synthetic data; since the screening operates on random layers, this procedure can be repeated to build a candidate pool of distinct iteratively screened models.

As a practical matter, the latter two techniques require building candidate pools by repeated evaluation, which in the case of iterative screening could be costly, and will likely limit the size of the candidate pool that can be produced in such a manner. To mitigate that, the candidate pool for iterative screening could be produced by including all failed candidates at each level of iteration: this will produce a pool containing LBOW models of various depths, but will require no

additional evaluation. However, if iterative screening is too slow to build a viable candidate pool, and the larger pools of the other techniques outperform iterative screening, then that would be a valuable result in itself.

5.3.2 Effect of number of layers

We suspect that layered methods should outperform flat (single layer) approaches, but this remains to be tested empirically. Furthermore, there is no theoretical reason to expect that any particular number of layers should produce the best performance; that is a question that must be answered experimentally as well. It remains to be seen whether the best performance is obtained with relatively few or many layers; indeed, it is entirely possible that performance continually improves, limiting to but never reaching some final maximum.

5.3.3 Random description vs. unsupervised description

Another question that remains to be answered is whether random description functions are preferable to unsupervised learning (clustering/quantization); Pinto *et al.* [89] favor the former, with the claim that randomized intermediate features perform approximately as well as ones learned through unsupervised techniques.

5.3.4 Joint vs. independent grouping and description

Given the popularity of CRF inspired segmentation and labeling techniques, it will be interesting to know whether joint grouping and description outperforms independent grouping and description. These methods can be compared in multiple ways. First, it should be tested whether randomized grouping and description demonstrates any difference when performed jointly or independently. Second, the more important experiment is to compare the performance of a joint grouping and segmentation learned (for example, by CRF techniques) from the detailed annotations to the performance of random independent grouping and description, and guided independent grouping and description.

5.4 Research Topics

We have identified three main research topics in LBOW:

- Layer representations: We have suggested a few possibilities for grouping and description functions, however in the general framework these are much less restricted. An area of research is considering alternative representations for these operations.

- Incorporating external descriptors at higher levels: In the proposed framework, external descriptors (*e.g.*, STIPs, trajectory fragments) are only used as the base descriptors at the first level; at higher levels, descriptors are produced purely by merging and describing existing descriptors. This means that the layered representation only leverages existing features at the very bottom. Ideally, it would be possible to incorporate other descriptors at all levels of the layered representation; investigating ways to do so is a research area.

- Recommending layered representations: The proposed LBOW incorporates into the proposed feature recommendation only by considering each LBOW model to be a feature in itself. However, an interesting, but as yet uninvestigated, possibility is to be able to assemble a layered representation through recommendation, so that rather than recommending an entire layered representation as a monolithic feature, separate layers are assembled together into a hierarchy by recommendation. This is a challenge because the feature recommendation assumes that features can be evaluated independently, whereas the proposed layers in the representation are more tightly coupled.

5.5 Risks

The two major risks in the proposed LBOW framework are that the LBOW models generated through supervision of various sorts will overfit to the training data, and hence perform poorly on the real data, and that the randomly generated models will simply perform poorly on everything, and hence even clever ensembles of recommended layered representations will fail. To mitigate the first, we can consider a wide variety of synthetic data, or, in the supervision models that require less detailed annotation (iterative screening), real data can be used (*e.g.*, coarsely tagged YouTube clips). In the worst case, and as a sanity check, we can consider the Facebook data, where

there is sufficient real data that we have sufficient data that even half is enough to screen on. To mitigate the second risk, that randomized LBOW models will have overall poor performance, we can potentially consider search based methods rather than screening (*e.g.*, randomized gradient descent, genetic/evolutionary algorithms) to focus the effort of the random generation on models that are more likely to perform well.

6 Timeline

- Summer 2011: Basic feature recommendation, infrastructure development.
- Fall 2011: Feature recommendation.
- Winter 2011: Layered BOW.
- Spring 2012: Layered BOW, begin writing thesis.
- Summer 2012: Write thesis. Defense.

7 Statement of Work

We will undertake the following work (optional goals and areas of research in parenthesis):

- Feature recommendation through collaborative filtering
- Feature set recommendation
- (Time-budget constrained feature recommendation)
- (On-line feature recommendation)
- Layered bag-of-words (LBOW)
- (Incorporating external descriptors into LBOW)
- (Multi-level recommendations)

8 Conclusion

We have proposed how synthetic data, or other heterogeneous labeled data, might be used in a feature recommendation framework in order to efficiently select, or recommend, features that are likely to perform well on a target task. In addition, we have described how layered representation might be built on top of the standard bag of words framework, and we have described how these hierarchies might be selected using synthetic data, or have their intermediate layers learned by targeting detailed annotations present in the synthetic data. We suggest that these layered bag of words models could themselves be considered features for feature recommendation, so that collections of such models could be recommended and assembled into ensemble classifiers. Ultimately, it is unlikely that the hard problems of vision will be solved by any one method on any one task, but instead by a robust collection of methods built across many tasks. This work takes crucial steps in that direction by exploring how large amounts of mixed data might be used to build a store of algorithmic intuition.

References

[1] N. Pinto and D. Cox, "Beyond Simple Features: A Large-Scale Feature Search Approach to Unconstrained Face Recognition," in *International Conference on Automatic Face and Gesture Recognition (FG)*, 2011.

[2] Rich Caruana, "Multitask learning," in *Machine Learning*, 1997, pp. 41–75.

[3] Brendan Juba, "Estimating relatedness via data compression," in *In Proceedings of the 23 rd International Conference on Machine Learning*, 2006, pp. 441–448.

[4] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *CVPR*, 2008.

[5] A. Fathi and G. Mori, "Action recognition by learning mid-level motion features," in *CVPR*, 2008.

[6] Y. Ke, R. Sukthankar, and M. Hebert, "Event detection in crowded videos," in *ICCV*, 2007.

[7] E. Shechtman and M. Irani, "Space-time behavior-based correlation—or—how to tell if two underlying motion fields are similar without computing them?," *PAMI*, vol. 29, no. 11, 2007.

[8] M. D. Rodriguez, J. Ahmed, and M. Shah, "Action MACH: a spatio-temporal maximum average correlation height filter for action recognition," in *CVPR*, 2008.

[9] Osama Masoud and Nikos Papanikolopoulos, "A method for human action recognition," *Image and Vision Comp.*, vol. 21, 2003.

[10] Piotr Dollar, Vincent Rabaud, Garrison Cottrell, and Serge Belongie, "Behavior recognition via sparse spatio-temporal features," in *VS-PETS*, 2005.

[11] Y. Ke, R. Sukthankar, and M. Hebert, "Event detection in crowded videos," in *ICCV*, 2007.

[12] Y. Ke, R. Sukthankar, and M. Hebert, "Efficient visual event detection using volumetric features," in *ICCV*, 2005.

[13] E. Shechtman and M. Irani, "Space-time behavior based correlation -or- how to tell if two underlying motion fields are similar without computing them?," *PAMI*, vol. 29, no. 11, 2007.

[14] I. Laptev and P. Perez, "Retrieving actions in movies," in *ICCV*, 2007.

[15] A. Efros, A. Berg, G. Mori, and J. Malik, "Recognizing action at a distance," in *ICCV*, 2003.

[16] A. F. Bobick and J. W. Davis, "The recognition of human movement using temporal templates," *PAMI*, vol. 23, no. 3, 2001.

[17] Alper Yilmaz and Mubarak Shah, "Actions sketch: A novel action representation," in *CVPR*, 2005.

[18] Moshe Blank, Lena Gorelick, Eli Shechtman, Michal Irani, and Ronen Basri, "Actions as space-time shapes," in *ICCV*, 2005.

[19] S. N. Vitaladevuni, V. Kellokumpu, and L. S. Davis, "Action recognition using ballistic dynamics," in *CVPR*, 2008.

[20] Pingkun Yan, S. M. Khan, and M. Shah, "Learning 4D action feature models for arbitrary view action recognition," in *CVPR*, 2008.

[21] Daniel Weinland, Remi Ronfard, and Edmond Boyer, "Free viewpoint action recognition using motion history volumes," *Computer Vision and Image Understanding*, vol. 104, no. 2, 2006.

[22] Heng Wang, Muhammad M. Ullah, Alexander Kläser, Ivan Laptev, and Cordelia Schmid, "Evaluation of local spatio-temporal features for action recognition," in *British Machine Vision Conference*, Sept. 2009, p. 127.

[23] S. Ali, A. Basharat, and M. Shah, "Chaotic invariants for human action recognition," in *CVPR*, 2007.

[24] Cen Rao, Alper Yilmaz, and Mubarak Shah, "View-invariant representation and recognition of actions," *IJCV*, vol. 50, no. 2, 2002.

[25] Imran N. Junejo, Emilie Dexter, Ivan Laptev, and Patrick Pérez, "Cross-view action recognition from temporal self-similarities," in *ECCV*, 2008.

[26] V. Rabaud and S. Belongie, "Re-thinking non-rigid structure from motion," in *CVPR*, 2008.

[27] P. Matikainen, M. Hebert, and R. Sukthankar, "Trajectons: Action recognition through the motion analysis of tracked features," in *ICCV workshop on Video-oriented Object and Event Classification*, 2009.

[28] Pyry Matikainen, Martial Hebert, and Rahul Sukthankar, "Representing pairwise spatial and temporal relations for action recognition," in *ECCV*, 2010.

[29] R. Messing, C. Pal, and H. Kautz, "Activity recognition using the velocity histories of tracked keypoints," in *ICCV*, 2009.

[30] J. Winn, A. Criminisi, and T. Minka, "Object categorization by learned universal visual dictionary," in *ICCV*, 2005.

[31] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray, "Visual categorization with bags of keypoints," in *ECCV Workshop on Statistical Learning*, 2004.

[32] Laura Walker Renninger and Jitendra Malik, "When is scene identification just texture recognition?," *Vision Research*, vol. 44, no. 19, 2004.

[33] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman, "Total Recall: Automatic query expansion with a generative feature model for object retrieval," in *ICCV*, 2007.

[34] Vincent Delaitre, Ivan Laptev, and Josef Sivic, "Recognizing human actions in still images: a study of bag-of-features and part-based representations," in *Proc. BMVC*, 2010, pp. 97.1–11, doi:10.5244/C.24.97.

[35] J.C. Niebles, H. Wang, and L. Fei-Fei, "Unsupervised learning of human action categories using spatial-temporal words," *IJCV*, vol. 79, no. 3, 2008.

[36] Gustavo Carneiro and David Lowe, "Sparse flexible models of local features," in *ECCV*, 2006.

[37] D. J. Crandall and D. P. Huttenlocher, "Weakly supervised learning of part-based spatial models for visual object recognition," in *ECCV*, 2006.

[38] M. Leordeanu, M. Hebert, and R. Sukthankar, "Beyond local appearance: Category recognition from pairwise interactions of simple features," in *CVPR*, June 2007.

[39] Z. M. Zhang, Y. Q. Hu, S. Chan, and L. T. Chia, "Motion context: A new representation for human action recognition," in *ECCV*, 2008.

[40] Hao Jiang and David R. Martin, "Finding actions using shape flows," in *ECCV*, 2008.

[41] Neil Johnson and David Hogg, "Learning the distribution of object trajectories for event recognition," *Image and Vision Computing*, vol. 14, 1996.

[42] Dimitrios Makris and Tim Ellis, "Spatial and probabilistic modelling of pedestrian behaviour," in *BMVC*, 2002.

[43] L. Ladicky, C. Russell, P. Kohli, and P.H.S. Torr, "Associative hierarchical crfs for object class image segmentation," in *ICCV*, 2009, pp. 739–746.

[44] P. Schnitzspan, M. Fritz, S. Roth, and B. Schiele, "Discriminative structure learning of hierarchical representations for object detection," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, june 2009, pp. 2238 –2245.

[45] Cordelia Schmid, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *In CVPR*, 2006, pp. 2169–2178.

[46] Huawu Deng and David A. Clausi, "Unsupervised image segmentation using a simple mrf model with a new implementation scheme," *Pattern Recognition*, vol. 37, no. 12, pp. 2323 – 2335, 2004.

[47] Yuri Boykov, Olga Veksler, and Ramin Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, pp. 2001, 2001.

[48] Kevin Lai and Dieter Fox, "Object recognition in 3D point clouds using web data and domain adaptation," *International Journal of Robotics Research*, vol. 29, pp. 1019–1037, 2010.

[49] H. Daumé III, "Frustratingly easy domain adaptation," in *Proceedings of Association of Computational Linguistics*, 2007.

[50] R. Collobert, F. Sinz, J. Weston, L. Bottou, and T. Joachims, "Large scale transductive SVMs," *Journal of Machine Learning Research*, vol. 7, 2006.

[51] Sinno Jialin Pan and Qiang Yang, "A survey on transfer learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 10, pp. 1345 –1359, oct. 2010.

[52] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[53] J. Loughrey and P. Cunningham, "Overfitting in wrapper-based feature subset selection: The harder you try the worse it gets," in *Research and Development in Intelligent Systems XXI*, pp. 33–43. Springer, 2005.

[54] T. Windeatt and K. Dias, "Feature ranking ensembles for facial action unit classification," in *Proceedings Artificial Neural Networks in Pattern Recognition*, 2008.

[55] X. Wang, C. Zhang, and Z. Zhang, "Boosted multi-task learning for face verification with applications to web image and video search," in *CVPR*, 2009.

[56] W. Dai, Q. Yang, G. Xue, and Y. Yu, "Boosting for transfer learning," in *Proceedings of International Conference on Machine Learning*, 2007.

[57] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, "Analysis of representations for domain adaptation," in *NIPS*, 2007.

[58] L. Duan, D. Xu, I. Tsang, and J. Luo, "Visual event recognition in videos by learning from web data," in *CVPR*, 2010.

[59] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local SVM approach," in *ICPR*, 2004.

[60] L. Cao, Z. Liu, and T. Huang, "Cross-dataset action detection," in *CVPR*, 2010.

[61] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Supervised dictionary learning," in *NIPS*, 2008.

[62] W. Brendel and S. Todorovic, "Activities as time series of human postures," in *ECCV*, 2010.

[63] L. Yang, R. Jin, C. Pantofaru, and R. Sukthankar, "Discriminative cluster refinement: Improving object category recognition given limited training data," in *CVPR*, 2007.

[64] Y. Boureau, F. Bach, Y. Le Cun, and J. Ponce, "Learning mid-level features for recognition," in *CVPR*, 2010.

[65] X. Lian, Z. Li, C. Wang, B. Lu, and L. Zhang, "Probabilistic models for supervised dictionary learning," in *CVPR*, 2010.

[66] D. Stavens and S. Thrun, "Unsupervised learning of invariant features using video," in *CVPR*, 2010.

[67] G. Carneiro, "The automatic design of feature spaces for local image descriptors using an ensemble of non-linear feature extractors," in *CVPR*, 2010.

[68] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NIPS*, 2007.

[69] John Blitzer, Ryan Mcdonald, and Fernando Pereira, "Domain adaptation with structural correspondence learning," in *In EMNLP*, 2006.

[70] Guillaume Obozinski and Ben Taskar, "Multi-task feature selection," Tech. Rep., In the workshop of structural Knowledge Transfer for Machine Learning in the 23rd International Conference on Machine Learning (ICML, 2006.

[71] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil, "Multi-task feature learning," in *Advances in Neural Information Processing Systems 19*. 2007, MIT Press.

[72] Theodoros Evgeniou, Charles A. Micchelli, and Massimiliano Pontil, "Learning multiple tasks with kernel methods," *Journal of Machine Learning Research*, vol. 6, pp. 615–637, 2005.

[73] Amr Ahmed, Kai Yu, Wei Xu, Yihong Gong, and Eric Xing, "Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks," in *ECCV*, 2008.

[74] Ulrich Rckert and Stefan Kramer, "Kernel-based inductive transfer," in *in Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, ser. Lecture Notes in Computer Science*. 2008, pp. 220–233, Springer.

[75] Ingo Mierswa and Michael Wurst, "Efficient case based feature construction for heterogeneous learning tasks," Tech. Rep., In Proc. of ECML 2005, 2005.

[76] Ingo Mierswa and Michael Wurst, "Efficient feature construction by meta learning - guiding the search in meta hypothesis space," in *In Proc. of the Internation Conference on Machine Learning, Workshop on Meta Learning*, 2005, pp. 84–92.

[77] Su-In Lee, Vassil Chatalbashev, David Vickrey, and Daphne Koller, "Learning a meta-level prior for feature relevance from multiple related tasks," in *Proceedings of the 24th international conference on Machine learning*, New York, NY, USA, 2007, ICML '07, pp. 489–496, ACM.

[78] Y Koren, "The bellkor solution to the netflix grand prize," `http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf`, 2009.

[79] James Bennett, Stan Lanning, and Netflix Netflix, "The netflix prize," in *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[80] Linas Baltrunas and Francesco Ricci, "Locally adaptive neighborhood selection for collaborative filtering recommendations," in *Proceedings of the 5th international conference on Adaptive Hypermedia and Adaptive Web-Based Systems*. 2008, AH '08, pp. 22–31, Springer-Verlag.

[81] Robert M. Bell and Yehuda Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," in *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, Washington, DC, USA, 2007, pp. 43–52, IEEE Computer Society.

[82] Yehuda Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2008, KDD '08, pp. 426–434, ACM.

[83] Yehuda Koren, "Factor in the neighbors: Scalable and accurate collaborative filtering," *ACM Trans. Knowl. Discov. Data*, vol. 4, pp. 1:1–1:24, January 2010.

[84] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman, "Indexing by latent semantic analysis," *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, vol. 41, no. 6, pp. 391–407, 1990.

[85] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton, "Restricted boltzmann machines for collaborative filtering," in *In Machine Learning, Proceedings of the Twenty-fourth International Conference (ICML 2004). ACM*. 2007, pp. 791–798, AAAI Press.

[86] Nguyen Duy Phuong and Tu Minh Phuong, "Collaborative filtering by multi-task learning," in *Research, Innovation and Vision for the Future, 2008. RIVF 2008. IEEE International Conference on*, july 2008, pp. 227 –232.

[87] Yisheng Chen, Rick Parent, Raghu Machiraju, and Jim Davis, "Human activity recognition for synthesis," in *IEEE CVPR Workshop on Learning, Representation and Context for Human Sensing in Video*, 2006.

[88] F. Qureshi and D. Terzopoulos, "Surveillance in virtual reality: System design and multi-camera control," in *CVPR*, 2007.

[89] N. Pinto, D. Doukhan, J. DiCarlo, and D. Cox, "A high-throughput screening approach to discovering good forms of biologically inspired visual representation," *PLoS Computational Biology*, vol. 5, no. 11, 2009.

[90] J. Shotton, A. Fitzgibbon, M. Cook, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *In CVPR*, 2011.

[91] Huan Liu, Hiroshi Motoda, and Lei Yu, "A selective sampling approach to active feature selection," *Artif. Intell.*, vol. 159, pp. 49–74, November 2004.

[92] Jingen Liu, Jiebo Luo, and Mubarak Shah, "Recognizing realistic actions from videos "in the wild"," in *CVPR*, 2009.

[93] University of Central Florida, "Ucf50 action recognition dataset," .

[94] "Virat video datasets release 1.0," `http://viratdata.org`.

[95] "Caviar project," `http://homepages.inf.ed.ac.uk/rbf/CAVIAR/`.

[96] Alan F. Smeaton, Paul Over, and Wessel Kraaij, "Evaluation campaigns and trecvid," in *MIR '06: Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval*, New York, NY, USA, 2006, pp. 321–330, ACM Press.

[97] Carnegie Mellon University Graphics Lab, "CMU graphics lab motion capture database," 2001.

[98] A. Torralba and A. Efros, "Unbiased look at dataset bias," in *CVPR*, 2011.

[99] Ivan Laptev, "Improving object detection with boosted histograms," *Image Vision Comput.*, vol. 27, pp. 535–544, April 2009.

[100] J. Yuan, Z. Liu, and Y. Wu, "Discriminative subvolume search for efficient action detection," in *CVPR*, 2009.

[101] Stan Birchfield, "KLT: An implementation of the Kanade-Lucas-Tomasi feature tracker," 2007.

[102] Ulrike von Luxburg, "A tutorial on spectral clustering," *CoRR*, vol. abs/0711.0189, 2007.

9 Appendix A: Trajecton motion descriptors

Here we describe a simple trajectory-based motion descriptor for video; this descriptors is described more fully in our papers [27, 28]. These descriptors are quantized, local descriptors, so that each descriptor extracted from a video can be expressed as a tuple of (x, y, t, l), where x, y, t specify the spatio-temporal location of the descriptor, and l is the discrete label assigned to the descriptor.

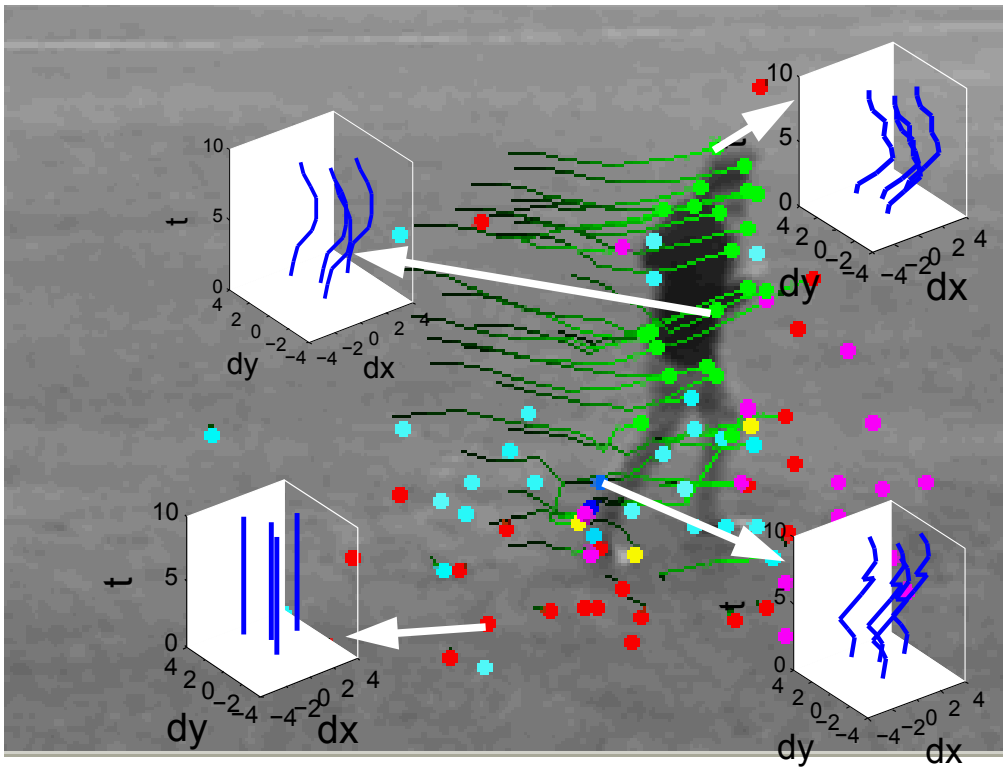


Figure 15: Simple KLT feature tracking is used to track as many features as possible within a video. Each tracked point produces a fixed length trajectory snippet every frame consisting of the last L (usually ten) positions in its trajectory. These snippets are quantized to a library of trajectons.

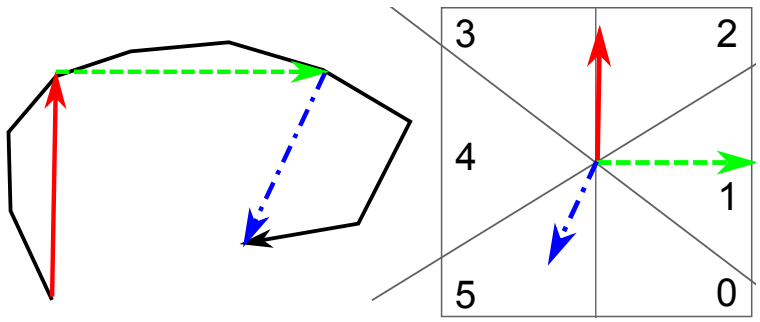


Figure 16: Sequencing code map (SCM) quantization breaks a trajectory fragment into a number of stages (in this case three) that are separately quantized according to a map (in this case a 6-way angular map). These per-stage labels, called sequence codes, are combined into a final label for the fragment, which in this case would be $215_6 = 83_{10}$.

9.1 Patch Tracking and Trajectory Snippet Production

A standard KLT tracker is used to track image patches (using "good features to track") over a video. In our implementation, we track a fixed number of points (typically 100 to 300), with tracks replaced as necessary when lost. The output of this tracking is a trace of (x, y) pairs for each tracked point. For convenience of notation, we can assume that indices are never reused; then we can express a point i's position at time t as $X_i^t = (x_i^t, y_i^t)$.

Then, for each frame, each track that exists during this frame produces a trajectory snippet that consists of the discrete derivatives of the point's locations in time. In other words, given a frame time t and point i, and a maximum snippet length L, the trajectory snippet produced is:

$$T_i^t = \{X_i^t - X_i^{t-1}, X_i^{t-1} - X_i^{t-2}, \ldots, X_i^{t-L+1} - X_i^{t-L}\},$$

where if X_i^j does not exist for a given time any terms containing it are set to zero.

Since X_i includes both x and y position, the full flattened vector will be of length $2L$. If the number of tracked features is fixed at n, and a video has f frames, this means that the total number of trajectory snippets (and hence eventually trajectons) will be nf. Also, note that if a point is tracked for longer than L frames, every window of size L in that trajectory produces its own snippet.

9.2 Trajectory Snippet Clustering and Quantization

These trajectory snippets are quantized to a set of labels. One method is to cluster a training set of snippets using k-means with the standard Euclidean distance metric into k clusters, and then store the cluster centers in a library of k trajectons [27]. Trajectory snippets are then quantized using this library by assigning each trajectory snippet the index of the trajecton to which it has the smallest Euclidean distance.

In a later work [28] we introduced an alternative, fixed quantization scheme, which we called sequencing code map (SCM) quantization (see Fig. 16). In this method, each trajectory fragment is divided into s consecutive stages of length t frames, such that $st \leq T$, where T is the total length of the fragment. The total motion, or summed derivative, of each stage is computed as a $(dx, dy)_k$ pair for each stage. This (dx, dy) vector is quantized according to a lookup table into one of n stage labels, or sequence codes; the s sequence codes are combined to produce a single combined label that can take on $s^n = k$ values. For example, with 3 stages and an SCM containing 8 bins, there would be 8^3 or 512 labels total. Since the time to quantize a stage is a single table lookup regardless of how the table was produced, this method is extremely computationally efficient (*i.e.*, the computation time does not grow with an increasing number of quantization bins).

Formally, we denote by $M(dx, dy)$ the SCM function, which is implemented as a two-dimensional lookup table that maps from a dx, dy to an integer label in the range of 0 to $n - 1$ inclusive. We denote by $(dx, dy)_s$ the derivative pair for stage s. Then the assigned label is given by $l = \sum_{j=0}^{s} n^j \cdot M(dx_j, dy_j)$.

Note that neither of these approaches makes any attempt to explicitly induce scale invariance, either spatial or temporal. If a particular action occurs at different scales or speeds, then instances of that action are initially represented by different sets of trajectons, and it is at the classification stage that these instances are grouped together under a single label. This idea is consistent with typical bag of words approaches and allows the representation to discriminate between similar types of motion when necessary (*e.g.*, , running *vs*.jogging).