THESIS

ELEMENT REARRANGEMENT FOR ACTION CLASSIFICATION ON PRODUCT MANIFOLDS

Submitted by

Karthik Kadappan

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2013

Master's Committee:

Advisor: J. Ross Beveridge

Anthony A. Maciejewski
Chris Peterson
Sanjay Rajopadhye

ABSTRACT

ELEMENT REARRANGEMENT FOR ACTION CLASSIFICATION
ON PRODUCT MANIFOLDS

Conventional tensor-based classification algorithms unfold tensors into matrices using the standard mode-k unfoldings and perform classification using established machine learning algorithms. These methods assume that the standard mode-k unfolded matrices are the best 2-dimensional representations of N-dimensional structures. In this thesis, we ask the question: "Is there a better way to unfold a tensor?" To address this question, we design a method to create unfoldings of a tensor by rearranging elements in the original tensor and then applying the standard mode-k unfoldings. The rearrangement of elements in a tensor is formulated as a combinatorial optimization problem and tabu search is adapted in this work to solve it. We study this element rearrangement problem in the context of tensor-based action classification on product manifolds. We assess the proposed methods using a publicly available video data set, namely Cambridge-Gesture data set. We design several neighborhood structures and search strategies for tabu search and analyze their performance. Results reveal that the proposed element rearrangement algorithm developed in this thesis can be employed as a preprocessing step to increase classification accuracy in the context of action classification on product manifolds method.

DEDICATION

I would like to dedicate this thesis to my beloved parents Meenal and Kadappan, my brother Jothi, and the beautiful city of Fort Collins.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my gratitude to the people who made this thesis possible. I deeply thank my advisor Dr. Ross Beveridge for many things. His patience, guidance, and support have made it possible for me to accomplish this work. I have learned from him the importance of designing toy-problems to tackle larger problems. I thank Dr. Anthony A. Maciejewski for providing constant encouragement and support throughout my graduate program. He also gave me the freedom to explore my own research interests. I am also grateful to Dr. Sanjay Rajopadhye and Dr. Chris Peterson for serving on my thesis committee.

A special thanks to Dr. Yui Man Lui for inspiring me use manifolds for visual recognition. Working with him taught me a lot about this field which I knew very little about. His intuition on the matter has strongly influenced and guided my approach.

I would like to express my gratitude towards all my friends and the beautiful city of Fort Collins for making me feel at home and keeping me sane through these years. I especially thank my friends: Bala, Baves, Dosii, Kous, Nyiiks, Sridhar, Tanni, Tika, Tyags, and Vam.

Lastly, I thank my parents and brother for being the wonderful people they are, and for being extremely understanding and encouraging through all the rough times I have had. Without their love, none of this would have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 What is action classification?

Action classification is the task of labeling motions in a video. Given a database of labeled videos and a query video the question to be addressed is, what is the "most probable" label to which the query belongs? More often than not we are interested in human actions because of the number of potential applications. Classification of human motions in a video can be performed at different levels of abstraction. Different taxonomies have been proposed by authors for dealing with movement at various levels of complexity. Terms such as action, gesture, activity, and behavior are often interchangeably used. A review of the different taxonomies is provided in [38].

A commonly used taxonomy (see reviews [38, 55]) is the one given by Moeslund et al. [50]: action primitive, action, and activity. An action primitive is an atomic entity. Actions are built from action primitives and activities are, in turn, built from actions. The granularity often depends on the context of the environment. As an example, in racquetball, action primitives could be, e.g., "run left", "run right", "forehand", and "backhand". In this example the term action is used for a sequence of action primitives needed to successfully return a ball. Actions could be, e.g., "kill shot", "pinch shot", and "ceiling shot". The activity is then "playing racquetball".

The growing interest in action classification is motivated by the number of potential applications in different areas in human-computer interaction [29], robotics [8], security industry, and entertainment industry. Moeslund et al. roughly group the applications under three titles: surveillance, control, and analysis [50]. Surveillance applications cover problems related to detecting abnormal activities in public locations and crowd behavior or congestion

analysis (say, in shopping malls). Control applications cover problems where action is used to control something. This could be used in games (e.g., Microsoft Xbox Kinect, Sony PS3 Move), virtual reality, and other human-computer interface applications. An example of an analysis application is automatic annotation of videos for efficient content-based retrieval of videos.

1.2 Motivation

In recent years, many action classification algorithms have been proposed. However, reliable action classification remains a challenge due in part to the complexity of human movements. Most actions have large variation in performance. For example, running movements can differ in stride length and speed. Issues such as invariance to viewing angle, scale, and subject further add to the complexity of the problem. Environment parameters such as clutter, occlusion, and lighting conditions are an additional source of variation. A good action classification algorithm should be able to generalize the variations within the same action but distinguish the variations between different actions.

Several approaches have been proposed for action classification. These methods can be logically divided into two categories - pixel-based methods [33, 48] and feature-based methods [39, 53, 59, 72]. Feature-based methods are better suited for activity recognition and are not considered in this work. On the other hand, pixel-based methods are better suited for action recognition, in particular actions in controlled environments. They are sensitive to background clutter, nonaligned actions, and large view point changes. Nevertheless, the success of pixel-based methods greatly depends on reducing the correlations within the videos.

Pixel-based methods take advantage of the fact that videos contain a significant amount of redundant information. The redundancy is because of the spatial-temporal coherence and structural commonalities found within videos [61]. Kersten demonstrated this redundancy perceptually by asking human observers to restore missing pixels in an image [32]. It is often useful to pare away with these redundancies so that the intrinsic features of the video

are revealed and used for action classification. The goal of subspace methods (discussed in Chapter 2) and subspace-based methods (discussed in Chapter 3) are to perform classification by removing correlations in the data.

A state-of-the-art example of tensor-based action classification is from a recent paper by Liu et al. [48]. This method is called the action classification on product manifolds method (discussed in Chapter 3). Liu [48] represents a video as a third order tensor and applies modified higher order singular value decomposition to generate three orthogonal matrices, one for each of the three standard unfoldings of the 3-dimensional structure into a 2-dimensional structure. These orthogonal matrices span subspaces that capture the variations of the row space of the three standard unfolded matrices. The success of such tensor-based methods heavily depends on the extent to which discriminating information is captured in the rows of the standard unfolded matrices. The work by Liu [48] assumes that the standard unfolded matrices are the best 2-dimensional representations of 3-dimensional structures for the purpose of classification. This observation gives rise to an important question: "Is there a better way to unfold a tensor?"

In this work, new unfoldings of a tensor are developed by rearranging elements in the original tensor and then applying the standard mode-k unfoldings. The goal of the element rearrangement problem is to reduce the intra-class distance and increase the inter-class distance. The element rearrangement algorithm developed in this thesis can be employed as a preprocessing step to increase classification accuracy in the context of action classification on product manifolds method.

1.3 Previous Work

As mentioned in the previous section, the goal of subspace methods is to perform classification by removing correlations in the data. Sirovich and Kirby [36] and Turk and Pentland [68] introduced the idea of subspace methods for face recognition. Murase and Nayar [51] extended the idea to 3D Object Recognition. In Tensorfaces [70], multilinear analysis of ensembles of facial images is considered. All of the above methods unfold each image into a

vector and reduce the correlations among the different pixels. However, the long column vectors sometimes result in degraded performance due to the curse of dimensionality and the small-sample-size problem.

In recent years, many works [43, 73, 76, 77, 78] represent images as matrices and reduce correlations within the image rows and columns, rather than among all the pixels in the image. This, to an extent, reduces the curse of dimensionality and small-sample-size problem. For image data represented in a matrix form, higher performance has been reported in [43, 78]. Similarly, for image sets represented as tensors, where correlations are removed along the column vectors of the mode-k flattened matrices, higher performance has been reported in [73, 76].

Yan et al. present an element rearrangement method for tensor-based subspace learning in [75]. In this method, the elements within a tensor are rearranged to maximize the correlations along the mode-k flattened matrices, so that existing tensor-based dimensionality reduction techniques can effectively remove the redundancy in the tensor data. Yan formulates the element rearrangement problem as an integer optimization problem with a nonlinear objective function. In addition, the element rearrangement algorithm is extended for improving the data compression performance and classification accuracy. Results are reported for the CMU PIE data set [60].

1.4 Contributions

This section summarizes the main contributions of this thesis.

1. Presents a tutorial on the action classification on the product manifolds method.

2. Formulates the element rearrangement problem as a local search problem.

3. Defines useful search spaces and neighborhood structures.

4. Studies the relationship between the element rearrangement operation and the subspaces spanned by the rows of the mode-k flattened matrices in the context of action classification.

1.5 Overview of Chapters

This thesis is organized into six chapters. Chapter 2 introduces the Cambridge-Gesture data set used in this work and describes the procedure for preprocessing the data set. A thought experiment for action classification using subspace methods is presented in this chapter.

Chapter 3 introduces tensors and Grassmann manifolds as a unifying framework for subspace-based action classification and discusses the action classification on the product manifolds method. Chapter 3 also reviews the necessary mathematical background on tensor algebra and Grassmann manifolds. Mathematics covered in this chapter include tensor unfolding, higher order singular value decomposition, principal angles, Stiefel manifolds, Grassmann manifolds, product manifolds, and geodesic distance. Finally,Chapter 3 provides a tutorial on the action classification on product manifolds method with a simple toy problem.

Chapter 4 defines the element rearrangement problem and formulates the problem as a combinatorial optimization problem. The local search method is proposed to solve the optimization problem and a general algorithm for local search method is discussed. Two variants of local search, hill climb and tabu search, are applied to the element rearrangement problem. Different search spaces and neighborhood structures are proposed for improving the tabu search. The chapter contains further discussion on the different tabu search strategies used.

In Chapter 5, parameters of the tabu search are experimentally determined. Four heuristics are defined based on the tabu search strategies developed in Chapter 4. Experimental results on the Cambridge-Gesture data set are reported for the four heuristics. This chapter ends with a discussion on the relation between the subspaces spanned by mode-k flattened matrices and the element rearrangement operation. Finally, Chapter 6 discusses the conclusions of this thesis and possible future work.

To summarize, the main chapters in this thesis are divided into two parts. Chapters 2 and 3 integrate the known facts and set up the framework for this thesis. Chapters 4 and 5 contain the main proposal, experimental results, and analyses.

Chapter 2

Action Classification

This chapter introduces the data set used in this work and describes the procedure for preprocessing the data set. We discuss a simple thought experiment for action classification which extends the idea of Eigenimages to Eigenvideos. The thought experiment is used to establish a platform and define terminology to be used in the rest of the thesis.

2.1 Data set

All the experiments in this study are performed on the Cambridge-Gesture database [35][1]. The database contains nine gestures generated by the combinations of three primitive hand shapes (flat, spread, and V-shape) and three primitive motions (leftward, rightward, and contract). Each of the nine gestures are repeated in five sets (Set1, Set2, Set3, Set4, and Set5) of varying illuminations with ten motions for each of the two subjects giving 900 videos. Videos are in RGB format with a frame size of 320×240, and number of frames in a video vary. Examples of these gestures from Set1 of the Cambridge-Gesture data set are given in Figure 2.1.

All the videos are converted from the RGB format to the grayscale format and resized to $20 \times 20 \times 32$. Color information is discarded because only the intensity of the pixels matter in the algorithms considered in this work. Video length is fixed at 32 frames for computational convenience, and these 32 frames are collected from the middle of a video sequence. Frame resizing is done using bilinear interpolation. Each video can be represented by a three dimensional array.

[1]The database is publicly available at `ftp://mi.eng.cam.ac.uk/pub/CamGesData`.

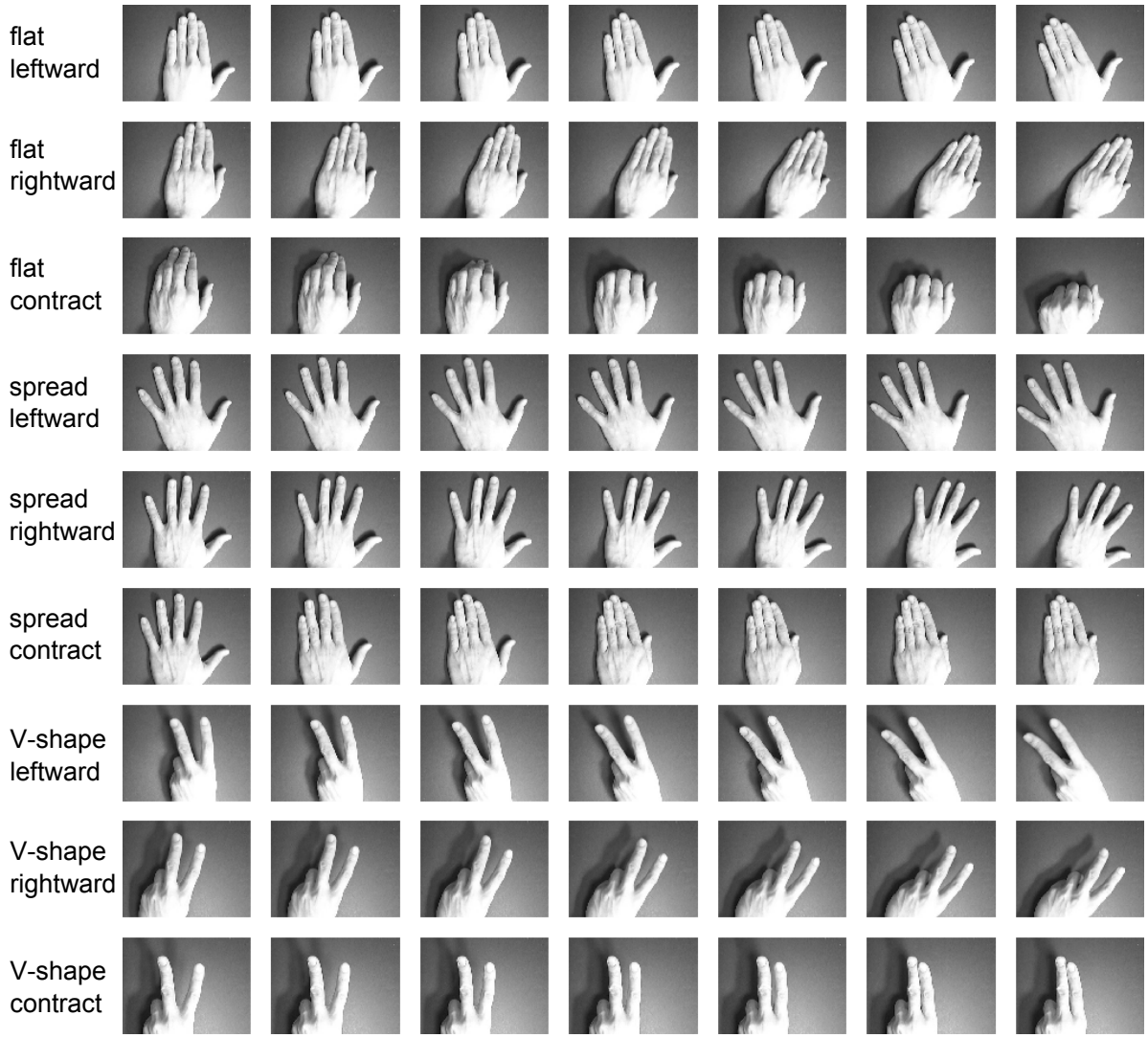


Figure 2.1: Example gestures from Set1 of the Cambridge-Gesture database. Each of the nine rows correspond to a gesture. In each row seven frames out of the 32 frames are presented for each gesture. From top the gestures are: flat leftward, flat rightward, flat contract, spread leftward, spread rightward, spread contract, V-shape leftward, V-shape rightward, and V-shape contract.

2.2 Subspace based methods

Sequence of images are usually captured by a sensor, digitized, and stored in a digital system as 2D arrays of pixels. An image of size $m \times n$ pixels can be unrolled (vectorized) so that it represents a point in an mn dimensional Euclidean space. This space is often referred to as the "image space". Classification in this high dimensional image space can be done

using a simple distance metric such as Euclidean distance. But classification performance in such high dimensional spaces is quite often degraded due to the curse of dimensionality and the small-sample-size problem.

In the context of classification problems, curse of dimensionality refers to the phenomenon of degradation of performance as the dimension of the input data grows. This in part due to the problem of sparse or limited training samples, and the very fundamental relationship between the number of training samples available for the machine learning algorithm and the degrees of freedom in the space in which the learning algorithm must construct a decision criteria. Increase the number of dimensions, and in the most general sense, one needs more training samples. Another aspect of the curse of dimensionality is the problem of data concentration. In [7] Beyer argues that under certain assumptions on the distribution of data the ratio of the distances of the query point to the nearest and farthest neighbors tends to one as dimensionality of the data increases. In such cases, the meaningfulness of nearest neighbors and classification is diminished.

The curse of dimensionality and the ill-posed nature of the the classification problem above can be overcome in subspace methods. Sirovich and Kirby [36] and Turk and Pentland [68] introduced the idea of subspace methods for face recognition. Murase and Nayar [51] extended the idea to 3D Object Recognition. Since then subspace methods have been used to approach problems in the area of visual learning and recognition. The goal of subspace based methods is to reduce the dimensionality of the data by projecting it into subspaces while retaining as much useful information as possible in the original data set. The challenge is to determine under what conditions the subspace projecting an image into a subspace improve performance. There are several choices for subspaces.

A basic algorithm for image classification using Eigenspaces is given here. Although some of the details may vary, image classification using other subspace methods often follow the similar procedure. Compute the Eigenspace using the training images. The basis for the Eigenspace are called Eigenimages. Classification takes three basic steps. First, training

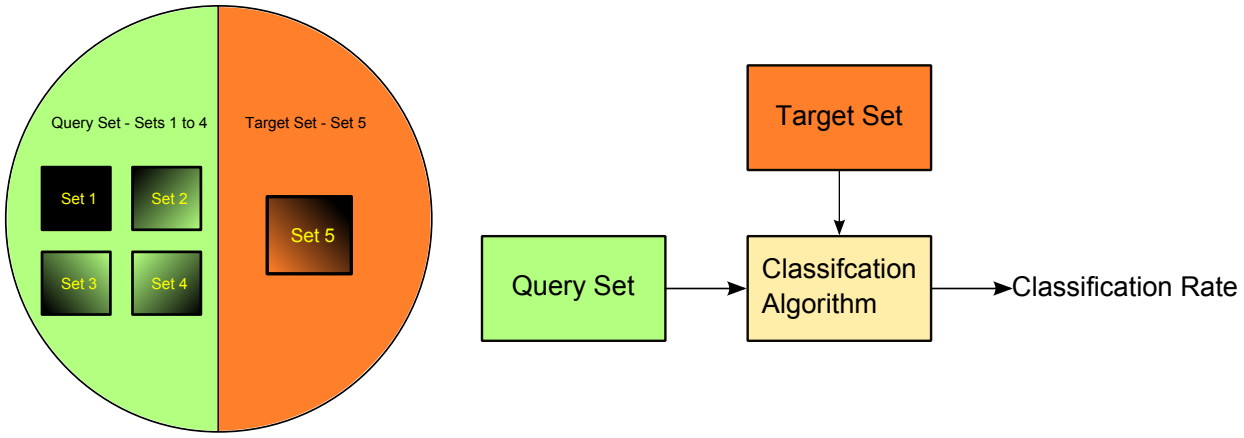


Figure 2.2: Action classification using subspace methods. Cambridge-Gesture data set is partitioned into query set (Set1, Set2, Set3, and Set4) and target set (Set5). Target set is the labeled data set, and query set is identified by the algorithm.

images are projected into the Eigenspace. Next, the query image is also projected into the Eigenspace. Finally, the query image is classified by comparing it to the projected training images. In the following section the idea of Eigenimages is extended to Eigenvideos.

2.3 Action Classification using Subspace Methods

The idea of subspace methods for image classification can be extended for classifying actions. There are several choices for subspaces. The following steps summarize a thought experiment to classify actions using an Eigenspace as the subspace:

1. Partition the data set

   The Cambridge-Gesture data set is divided into query set and target set. Let $\mathcal{A}_{pqr} \in \mathbb{R}^{20\times20\times32}$ represent the r^{th} sample of the q^{th} action from set p. The query set consists of Set1, Set2, Set3, and Set4 and is represented by $D_{query} = \{\mathcal{A}_{pqr} \in \mathbb{R}^{20\times20\times32} \mid 1 \leq p \leq 4,\ 1 \leq q \leq 9,\ 1 \leq r \leq 20\}$. The target set consists of Set5 and is represented by $D_{target} = \{\mathcal{A}_{pqr} \in \mathbb{R}^{20\times20\times32} \mid p = 5,\ 1 \leq q \leq 9,\ 1 \leq r \leq 20\}$. The target set is the labeled set used to identify the videos from the query set. This is shown in Figure 2.2.

2. Compute Eigenspace and Eigenvideos

   Eigenspace and Eigenvideos are computed using the following steps:

   (a) Unroll data: Unroll all the videos in the query set D_{query} by first stacking columns

within the frame and then across frames to obtain a vector. Each of these vectors reside in a 12800-dimensional space, $x^{pqr}_{query} \in \mathbb{R}^{12800\times 1}$.

(b) Center data: Each vector is centered by subtracting the mean of all the vectors from each vector. The mean centered vectors are represented by $\bar{x}^{pqr}_{query}$.

(c) Create video data matrix: All centered vectors are combined into a video data matrix, $\mathbf{D} \in \mathbb{R}^{12800\times 720}$.

(d) Compute Eigenspace and Eigenvideos: The singular value decomposition (SVD) is computed for the the video data matrix as shown in Equation (2.1). Eigenvideos are given by the columns of the left singular matrix, $\mathbf{U}$, and form the basis for the Eigenspace.

$$\mathbf{D} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{V}^T \tag{2.1}$$

3. Classify target videos

Each target video is unrolled and mean centered. Both target videos and query videos are projected into the Eigenspace as shown in Equation (2.2).

$$\begin{aligned} \tilde{x}^{pqr}_{query} &= \mathbf{U}^T \bar{x}^{pqr}_{query}, \\ \tilde{x}^{pqr}_{target} &= \mathbf{U}^T \bar{x}^{pqr}_{target} \end{aligned} \tag{2.2}$$

The projected query and target videos can be compared using different metrics. The most common metric is the L_2 norm. Each projected target video is compared with each of the projected query videos. Action of the query video is given by the action of the target video found closest to that query video.

From this thought experiment the following conclusion can be drawn. Use of subspace methods for image classification helped overcome the problem of curse of dimensionality and small-sample-size problem. Although, the usefulness of subspace methods to action classification problems is reduced due to the size of the videos. For example, an 8×8 image when unrolled resides in a 64-dimensional space, whereas, a $20 \times 20 \times 32$ video resides in a 12800-dimensional space.

Chapter 3

Action Classification using Tensor-Based Subspace Comparison

This chapter discusses the motivation behind tensor-based action classification and illustrates the action classification on the product manifolds method using a toy problem. This chapter also reviews the mathematical background on tensor algebra and Grassmann manifolds.

3.1 Motivation

3.1.1 Tensors

Tensor and tensor decomposition originated in the late 1920, but did not receive much attention until the work done by Tucker [65] in 1960s. Most of the early applications of tensors was in the field of psychometrics and chemometrics. Starting in the early 2000's Tensor-Level thinking made its way into the field of Computer Vision [12]. Some applications include face recognition [70], visual tracking [44], and action classification [33, 48, 69].

Visual data (images and videos) is an ensemble of multiple factors. For example, face images are the composite consequence of multiple factors, including facial geometry, pose, expression, and illumination. Human motion is the confluence of factors such as action performed, speed, scene structure, viewing angle, scale, and so on. Linear algebra based methods such as Eigenspaces (discussed in Section 2.2) are best suited for single factor variations [70]. Multilinear algebra, the algebra of higher order tensors, offers a mathematical framework for capturing multiple factor variations and interactions.

A video may be expressed as 3-dimensional array containing spatial and temporal information and can be represented by a third order tensor. Useful information such as multiple factor variations and interactions in a video can be extracted using tensor decomposi-

tion [33, 48, 69]. A survey paper on general topics of tensor decomposition and applications may be found in [37].

3.1.2 Subspace Comparison

We often encounter linear subspace structure in image-sets and videos. For example, a set of images of an object under varying lighting conditions can be approximated by a low dimensional linear subspace with mild assumptions [4, 6, 56]. Other variations such as pose, view angle, and expression can also be approximated with low dimensional subspaces.

Videos of human actions are more than just a set of images because it contains temporal information. Such spatio-temporal data are often modeled using linear dynamical models, and the parameters of these linear dynamical models are finite dimensional linear subspaces of appropriate dimensions [66]. A general discussion on theoretical and empirical evidence of subspace structure in image-sets and videos can be found in [27, 66].

In subspace methods, as discussed in Section 2.2, classification is performed by projecting a probe vector into the subspace and comparing it with the target vectors in the subspace. In recent years, subspaces are treated as basic elements, and classification is performed by comparing subspaces instead of vectors [19, 28, 34, 42, 74]. Thebenefit of subspace comparison is the focus is on appropriate variations. For example, face images indexed by subject and illumination condition can be modeled as a collection of illumination subspaces for each subject. Non-discriminating information of illumination condition is absorbed as a variability within subspaces, and emphasis is on the discriminating information of subject as variability between the subspaces. Subspace comparison is also robust to missing data since subspaces can to an extent fill-in missing data.

We will refer to this approach of subspace comparison as subspace-based methods. It is also referred to as subspace-based learning [28]. Motivated by this approach, several subspace-based classification methods have been proposed [19, 28, 34, 42, 74].

3.1.3 Grassmann Manifolds

In the previous section the advantages of subspace-based methods is discussed. However, there is the challenge of representing and comparing subspaces appropriately. A more fundamental question is - Are images and videos sampled from a linear or a nonlinear space? An image or a video can be regarded as a collection of numbers representing the pixel intensities, and can be identified as a point in an abstract image space. It is argued in [46, 47, 63] that the image space is generally a non-euclidean space because the law of superposition does not hold in the image space. For example, a simple image translation or rotation cannot be expressed as a linear combination over the field of a Euclidean space.

To understand the challenge of classifying nonlinear data consider the example of data lying on a 2-dimensional "Swiss Roll". The points that are far apart on the swiss roll, as measured by traveling along the curved surface, may appear close in the high dimensional input space, as measured by the straight-line Euclidean distance. Therefore, performing classification in an Euclidean space may result in poor performance.

One approach to account for the geometry of the data is to use manifold learning methods. The goal of manifold learning is to learn a mapping from the high dimensional input space to a low dimensional space while preserving the intrinsic properties of the data. However, they require large amount of densely sampled training data which may not be be available for some real world applications. Two common manifold learning methods are Isomap [63] and Locally Linear Embedding [58].

Another school of thought is to represent visual data in an underlying parametrized space derived from the properties of differential geometry. As one example, the Grassmann manifold is the set of all d-dimensional subspaces in an n-dimensional Euclidean space, and exploits the geometric structure of visual data. This motivates the use of nonlinear surfaces such as Grassmann manifolds. Jihun et al. use Grassmann manifolds as common framework for subspace-based methods [28]. A study by Turga et al. show that inference problems over subspaces can be naturally cast as inference problems on the Grassmann manifold [66].

Grassmann manifolds have been exploited in many computer vision problems such as face recognition [28, 45], action recognition [48, 66, 67], clustering [13, 26], and visual tracking [57, 71].

3.2 Mathematical Background

This section reviews the mathematical background needed in the remainder of the thesis. Subjects include tensor unfolding, Higher Order Singular Value Decomposition (HOSVD), principal angles, Stiefel manifolds, Grassmann manifolds, product manifolds, and geodesic distance.

3.2.1 Tensor

In this section, we review some of the tensor algebra. The exposition is based on [16]. Additional references include [37, 40]. Recent survey papers of this subject can be found in [2, 15, 37]. *MATLAB* Tensor toolbox [3] is used in this work.

A tensor is a multidimensional array. More formally, an order-d tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is a real d-dimensional array $\mathcal{A}(1:n_1, \ldots, 1:n_d)$ where the index range in the k-th mode is from 1 to n_k. Scalar is a order-0 tensor, vector is an order-1 tensor, and matrix is an order-2 tensor. A video sequence can be represented using an order-3 tensor. The three dimensions can be represented by (x, y, t), where (x, y) corresponds to the location of the pixel within a frame and t corresponds to time or frame number.

Example 1: A third order tensor $\mathcal{A} \in \mathbb{R}^{3\times3\times3}$ is shown below. An element (i, j, k) of a third order tensor is denoted by a_{ijk}. For all the examples here it will be useful to think of the tensor $\mathcal{A}$ as video with 3 frames each of size 3×3 pixels.

$$\mathcal{A}(:,:,1) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}; \qquad \mathcal{A}(:,:,2) = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}; \qquad \mathcal{A}(:,:,3) = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix};$$

3.2.1.1 Tensor Unfolding

Tensor unfolding is a process of turning a tensor $\mathcal{A}$ in to a matrix $\mathbf{A}$. More formally, a tensor unfolding can be stated as rewriting a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ as a matrix $\mathbf{A} \in \mathbb{R}^{N_1 \times N_2}$

by arranging $\mathcal{A}$'s entries into the matrix $\mathbf{A}$ where $N_1 \times N_2 = n_1 \times \cdots \times n_d = N$. There are many possible ways to unfold a tensor. In this work, the most commonly used mode-k family of tensor unfoldings are considered. A more detailed discussion on tensor unfolding can be found in [37]. Some tensor algebra terms are discussed below:

1. Fiber: Higher order analogue of matrix rows and columns. A fiber of a tensor $\mathcal{A}$ is a vector a obtained by fixing every index but one. A matrix column is a mode-1 fiber, and a matrix row is a mode-2 fiber. Third order tensors have an additional fiber called a tube (mode-3 fiber). In general a mode-k fiber is obtained by keeping everything except the kth index fixed.

   Example 2: Mode-1, mode-2, and mode-3 fibers of tensor $\mathcal{A}$ given in Example 1 are shown below. Elements of mode-1, mode-2, and mode-3 fibers are denoted by $a_{:jk}$, $a_{i:k}$, and $a_{ij:}$ respectively.

$$\text{Mode 1: } \mathcal{A}(:,1,1) = \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} \quad \text{Mode 2: } \mathcal{A}(1,:,1) = \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \end{pmatrix} \quad \text{Mode 3: } \mathcal{A}(1,1,:) = \begin{pmatrix} a_{11} \\ b_{11} \\ c_{11} \end{pmatrix}$$

2. Slice: Two dimensional section of a tensor. A slice of a tensor $\mathcal{A}$ is a matrix $\mathbf{A}$ obtained by fixing all but two indices. The standard method of extracting slices from a tensor is to keep the first unfixed index of the tensor as the row index of the slice and the second unfixed index of the tensor as the column index of the slice.

   Example 3: Elements of horizontal, lateral, and frontal slices are denoted by $\mathbf{A}_{i::}$, $\mathbf{A}_{:j:}$, and $\mathbf{A}_{::k}$ respectively. Horizontal and lateral slice of tensor $\mathcal{A}$ given in Example 1 is shown below:

$$\text{Horizontal Slice: } \mathcal{A}(1,:,:) = \begin{pmatrix} \mathcal{A}(1,1,1) & \mathcal{A}(1,1,2) & \mathcal{A}(1,1,3) \\ \mathcal{A}(1,2,1) & \mathcal{A}(1,2,2) & \mathcal{A}(1,2,3) \\ \mathcal{A}(1,3,1) & \mathcal{A}(1,3,2) & \mathcal{A}(1,3,3) \end{pmatrix} = \begin{pmatrix} a_{11} & b_{11} & c_{11} \\ a_{12} & b_{12} & c_{12} \\ a_{13} & b_{13} & c_{13} \end{pmatrix}$$

$$\text{Lateral Slice: } \mathcal{A}(:,1,:) = \begin{pmatrix} \mathcal{A}(1,1,1) & \mathcal{A}(1,1,2) & \mathcal{A}(1,1,3) \\ \mathcal{A}(2,1,1) & \mathcal{A}(2,1,2) & \mathcal{A}(2,1,3) \\ \mathcal{A}(3,1,1) & \mathcal{A}(3,1,2) & \mathcal{A}(3,1,3) \end{pmatrix} = \begin{pmatrix} a_{11} & b_{11} & c_{11} \\ a_{21} & b_{21} & c_{21} \\ a_{31} & b_{31} & c_{31} \end{pmatrix}$$

In mode-k unfolding, the mode-k fibers are arranged to produce a mode-k flattened matrix of size $n_k \times \left(\frac{N}{n_k}\right)$. Mode-k unfolding is represented by $\mathcal{A}_{(k)}$. In $\mathcal{A}_{(1)}$, columns of the tensor are arranged as the columns of the matrix. Similarly, for $\mathcal{A}_{(2)}$ and $\mathcal{A}_{(3)}$ the rows and tubes of the tensor are arranged as columns of the matrix. The mode-k fibers of $\mathcal{A}$ can be arranged as columns of a matrix in more than one way. Three common unfoldings are discussed below:

1. Forward-cyclic: For $\mathcal{A}_{(k)}$ unfolding the rows of the matrix are indexed by k and the columns of the matrix are indexed $[k+1:d \quad 1:k-1]$ with the $k+1$ index repeating most frequently and $k-1$ index repeating least frequently [16]. The concept is easier to understand using an example.

   Example 4: Forward-cyclic unfolding of the tensor in Example 1 is given below:

$$\mathcal{A}_{(1)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_{11} & b_{12} & b_{13} & c_{11} & c_{12} & c_{13} \\ a_{21} & a_{22} & a_{23} & b_{21} & b_{22} & b_{23} & c_{21} & c_{22} & c_{23} \\ a_{31} & a_{32} & a_{33} & b_{31} & b_{32} & b_{33} & c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$\mathcal{A}_{(2)} = \begin{pmatrix} a_{11} & b_{11} & c_{11} & a_{21} & b_{21} & c_{21} & a_{31} & b_{31} & c_{31} \\ a_{12} & b_{12} & c_{12} & a_{22} & b_{22} & c_{23} & a_{32} & b_{32} & c_{32} \\ a_{13} & b_{13} & c_{13} & a_{23} & b_{23} & c_{23} & a_{33} & b_{33} & c_{33} \end{pmatrix}$$

$$\mathcal{A}_{(3)} = \begin{pmatrix} a_{11} & a_{21} & a_{31} & a_{12} & a_{22} & a_{32} & a_{13} & a_{23} & a_{33} \\ b_{11} & b_{21} & b_{31} & b_{12} & b_{22} & b_{32} & b_{13} & b_{23} & b_{33} \\ c_{11} & c_{21} & c_{31} & c_{12} & c_{22} & c_{32} & c_{13} & c_{23} & c_{33} \end{pmatrix}$$

   Interpretation: What does unfolding of tensor mean to a video sequence? For mode-1 unfolded tensor $\mathcal{A}_{(1)}$, the columns of $\mathcal{A}_{(1)}$ correspond to the columns within each frame of the video sequence and the rows of $\mathcal{A}_{(1)}$ correspond to the horizontal slice (captures the relation between (x,t)). For mode-2 unfolded tensor $\mathcal{A}_{(2)}$, the columns of $\mathcal{A}_{(2)}$ correspond to the rows within each frame of the video sequence and the rows of $\mathcal{A}_{(2)}$ correspond to the lateral slice (captures the relation between (y,t)). For mode-3 unfolded tensor $\mathcal{A}_{(3)}$, the columns of $\mathcal{A}_{(3)}$ correspond to a particular pixel across all the frames of the video sequence and the rows of $\mathcal{A}_{(3)}$ correspond to the frontal slice (captures the relation between (x,y)). In other words the rows correspond to the frames that are unrolled.

2. Backward-cyclic: Here the rows of the matrix are indexed by k and the columns of the matrix are indexed $[k-1:-1:1 \quad d:-1:k+1]$ with the $k-1$ index repeating most frequently and $k+1$ index repeating least frequently [16]. Similar to the forward-cyclic mode, it will be useful to understand the relation between the rows and columns of the unfolded tensor with the video.

   Example 5: Backward-cyclic unfolding of the tensor in Example 1 is given below.

$$\mathcal{A}_{(1)} = \begin{pmatrix} a_{11} & b_{11} & c_{11} & a_{12} & b_{12} & c_{12} & a_{13} & b_{13} & c_{13} \\ a_{21} & b_{21} & c_{21} & a_{22} & b_{22} & c_{22} & a_{23} & b_{23} & c_{23} \\ a_{31} & b_{31} & c_{31} & a_{32} & b_{32} & c_{32} & a_{33} & b_{33} & c_{33} \end{pmatrix}$$
$$\mathcal{A}_{(2)} = \begin{pmatrix} a_{11} & a_{21} & a_{31} & b_{11} & b_{21} & b_{31} & c_{11} & c_{21} & c_{31} \\ a_{12} & a_{22} & a_{32} & b_{12} & b_{22} & b_{32} & c_{12} & c_{22} & c_{32} \\ a_{13} & a_{23} & a_{33} & b_{13} & b_{23} & b_{33} & c_{13} & c_{23} & c_{33} \end{pmatrix}$$
$$\mathcal{A}_{(3)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{21} & a_{22} & a_{23} & a_{31} & a_{32} & a_{33} \\ b_{11} & b_{12} & b_{13} & b_{21} & b_{22} & b_{23} & b_{31} & b_{32} & b_{33} \\ c_{11} & c_{12} & c_{13} & c_{21} & c_{22} & c_{23} & c_{31} & c_{32} & c_{33} \end{pmatrix}$$

3. Ordered: Here the rows of the matrix are indexed by k and the columns of the matrix are indexed $[1:k-1 \quad k+1:d]$ with the first index repeating most frequently and d index repeating least frequently.

3.2.1.2 Tensor Mode-k Product

Mode-k product is the multiplication of mode-k unfolded tensor with a matrix of appropriate dimension, that is, each mode-k fiber is multiplied by the matrix. Mathematically, if $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is a tensor and $\mathbf{Q} \in \mathbb{R}^{m_k \times n_k}$ is a matrix, then mode-k matrix product is given by $\mathcal{P} = \mathcal{A} \times_k \mathbf{Q}$, where $\mathcal{P} \in \mathbb{R}^{n_1 \times \cdots \times n_{k-1} \times m_k \times n_{k+1} \times \cdots \times n_d}$.

Example 6: Consider the tensor $\mathcal{A}$ given in Example 1 and the matrix $\mathbf{Q}$ given below. Then the mode-2 product is given below:

$$\mathbf{Q} = \begin{pmatrix} q_{11} & q_{21} & q_{31} \\ q_{21} & q_{22} & q_{23} \end{pmatrix}$$
$$\mathcal{P} = \mathcal{A} \times_2 \mathbf{Q}$$
$$\mathcal{P} = \begin{pmatrix} q_{11} & q_{21} & q_{31} \\ q_{21} & q_{22} & q_{23} \end{pmatrix} \times \begin{pmatrix} a_{11} & b_{11} & c_{11} & a_{21} & b_{21} & c_{21} & a_{31} & b_{31} & c_{31} \\ a_{12} & b_{12} & c_{12} & a_{22} & b_{22} & c_{23} & a_{32} & b_{32} & c_{32} \\ a_{13} & b_{13} & c_{13} & a_{23} & b_{23} & c_{23} & a_{33} & b_{33} & c_{33} \end{pmatrix}$$

Another interpretation of mode-k multiplication is a linear combination of the matrices within a tensor. In the above example the mode-k product is a linear combination of the lateral matrices. Row 1 is the first lateral matrix (slice), similarly rows 2 and 3 are the second and third lateral matrix. It is important to note that the order of operands in the actual multiplication is the reverse of the order in which they are represented in the mode-k product formula. One of the reasons to move from normal multiplication notation to mode-k multiplication is to avoid the use of transpose as we move to the generalized Singular Value Decomposition (SVD) case. Below are the two equivalent notations for SVD of a matrix $\mathbf{A}$.

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T = \mathcal{S} \times_1 \mathbf{U} \times_2 \mathbf{V}$$

3.2.1.3 Higher Order Singular Value Decomposition

Higher Order Singular Value Decomposition (HOSVD) is a method for tensor decomposition (factorization). It is analogous to the SVD factorization of a matrix. HOSVD unfolds the tensor to a set of matrices and performs decomposition on all the unfolded matrices.

Given a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ the mode-k unfolding $\mathcal{A}_{(k)} \in \mathbb{R}^{n_k \times m_k}$ is factored using SVD as follows:

$$\mathcal{A}_{(k)} = \mathbf{U}^{(k)} \mathbf{\Sigma}^{(k)} {\mathbf{V}^{(k)}}^T \qquad k \in [1:d] \tag{3.1}$$

where $\mathbf{\Sigma}^{(k)} \in \mathbb{R}^{n_k \times m_k}$ is a diagonal matrix, $\mathbf{U}^{(k)} \in \mathbb{R}^{n_k \times n_k}$ is an orthogonal matrix spanning the column space of $\mathcal{A}_{(k)}$ associated with the non-zero singular values, and $\mathbf{V}^{(k)} \in \mathbb{R}^{m_k \times m_k}$ is an orthogonal matrix spanning the row space of $\mathcal{A}_{(k)}$ associated with the non-zero singular values. Then HOSVD of $\mathcal{A}$ is given by:

$$\mathcal{A} = \mathcal{S} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \cdots \times_d \mathbf{U}^{(d)} \tag{3.2}$$

where $\times_k$ denotes the mode-k product. $\mathcal{S} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is called the core tensor and is given by:

$$\mathcal{S} = \mathcal{A} \times_1 {\mathbf{U}^{(1)}}^T \times_2 {\mathbf{U}^{(2)}}^T \cdots \times_d {\mathbf{U}^{(d)}}^T \tag{3.3}$$

Refer to [40] for a more detailed discussion on HOSVD.

Interpretation: What does HOSVD of a tensor mean to a video sequence? Mode-1 left orthogonal matrix $\mathbf{U}^{(1)}$ provides an orthonormal basis for the column space of mode-1 unfolded tensor $\mathcal{A}_{(1)}$, so this captures the variation in the columns (the (x) dimension) within the frames. Mode-1 right orthogonal matrix $\mathbf{V}^{(1)}$ provides an orthonormal basis for the row space of mode-1 unfolded tensor $\mathcal{A}_{(1)}$, so this captures the variation in the horizontal slices (the (y,t) dimension). Similar interpretations can be made for mode-2 and mode-3 orthogonal matrices.

3.2.2 Principal Angles

The exposition here is based on the work by Golub *et al* in [25]. Principal angles between two linear subspaces gives information on how close the linear subspaces are to each other. Consider the simple case of two 1-D linear subspaces. In this case the principal angle reduces to the acute angle between the two vectors. Mathematically, let $P \in \mathbb{R}^n$ and $Q \in \mathbb{R}^n$ be two one dimensional linear subspaces. Then the principal angle is given by

$$\theta = \cos^{-1}\left(\frac{\left|P^T Q\right|}{||P||\,||Q||}\right) \tag{3.4}$$

If $\theta = 0$ the two 1-dimensional linear subspaces (vectors in this case) are identical, and if $\theta = 1$ the two linear subspaces are orthogonal. Extending this example, consider the linear subspaces spanned by the columns of the two matrices $\mathbf{P} \in \mathbb{R}^{n\times m}$ and $\mathbf{Q} \in \mathbb{R}^{n\times l}$, which are subspaces in $\mathbb{R}^n$. Mathematically, the two linear subspaces are given by $f = span(\mathbf{P})$ and $g = span(\mathbf{Q})$. Assume that $p = rank(\mathbf{P}) \geq rank(\mathbf{Q}) = q \geq 1$. Extending from the 1-dimensional case, the closeness of the linear subspaces f and g can be given by the smallest acute angle between two vectors chosen from f and g. Instead of choosing the minimum principal angle as a measure of closeness or any other single angle, a recursive definition proposed by Hotelling is considered [30]. A vector in f is a linear combination of the columns of matrix $\mathbf{P}$ and can be represented by a matrix-vector product $\mathbf{P}x$ where $x \in \mathbb{R}^m$. Similarly, a vector in g is given by $\mathbf{Q}y$ where $y \in \mathbb{R}^l$. The principal angles $\theta_{(k)}$ are

defined recursively for $k = 1, \ldots, q$ by

$$\theta_{(k)} = \min_{x \in f} \min_{y \in g} \cos^{-1}\left(x^T y\right) = \cos^{-1}\left({x_k}^T y_k\right), \text{ subject to}$$
$$||x|| \quad = \quad ||y|| \quad = \quad 1 \tag{3.5}$$
$$x^T x_i \quad = \quad y^T y_i \quad = \quad 0, \quad i = 1, \ldots, k-1$$

The principal vectors are given by $\{x_1, x_1, \ldots, x_q\}$ and $\{y_1, y_1, \ldots, y_q\}$. An efficient algorithm for computing the principal angles using SVD is given in [9].

Interpretation: How do principal angles for pairs of subspaces mean in the context of video sequence? Consider two tensors $\mathcal{A}_1$ and $\mathcal{A}_2$ representing two video sequences. HOSVD of $\mathcal{A}_1$ and $\mathcal{A}_2$ gives the following mode-k orthogonal matrices: ${\mathbf{U}_1}^{(k)}$, ${\mathbf{V}_1}^{(k)}$, ${\mathbf{U}_2}^{(k)}$, and ${\mathbf{V}_2}^{(k)}$. Each of these orthogonal matrices span subspaces that capture the different variations in the spatial and temporal dimensions of the video. Therefore, smaller principal angles $\theta_{(k)}$ between a pair of subspaces (${\mathbf{U}_1}^{(k)}$ and ${\mathbf{U}_2}^{(k)}$ or ${\mathbf{V}_1}^{(k)}$ and ${\mathbf{V}_2}^{(k)}$) indicates the two videos are similar.

3.2.3 Grassmann Manifold and Geodesic Distance

In the previous section, subspaces and the measure of (dis)similarity of subspaces was discussed. These subspaces are often given a topological structure, that is, they are expressed as points on topological manifolds (e.g. - Grassmann Manifold). The motivation for this is that it makes it possible to look at subspaces as instances from a continuous parameter and define distance between a pair of subspaces.

This section only briefly introduces the necessary definitions and properties of Grassmann manifolds from [17], and does not discuss this subject further. Refer to [14, 17, 41] for further reading. The goal of this section is to show that geodesic distance on Grassmann manifolds provide a measure of (dis)similarity between two subspaces.

1. Manifold - An n-dimensional manifold is a topological space which locally (on a small enough scale) resembles a Euclidean space of dimension n.

2. Quotient Space - Space that is obtained by defining an equivalence relation on the original

space. Each point in the new quotient space is mapped to a group of points in the original space. Quotient space is motivated by the fact that the closed form solutions available for the original space can be easily extended to its quotient space.

3. Orthogonal Group $\mathcal{O}_n$ - It is a set of $n \times n$ orthogonal matrices that satisfy the axioms of closure, associativity, identity element, and inverse element on the operation of matrix multiplication.

4. Stiefel Manifold $\mathcal{V}_{n,p}$ - Space of p orthonormal vectors in $\mathbb{R}^n$. A point on the Stiefel manifold is stored as an $n \times p$ orthonormal matrix.

5. Grassmann Manifold $\mathcal{G}_{n,p}$ - Set of p-dimensional linear subspaces in $\mathbb{R}^n$ for $0 < p \leq n$. A point on the Grassmann manifold may be represented by an $n \times p$ matrix that stores an arbitrary orthogonal basis for a linear subspace.

Unlike the case of Stiefel manifolds, a point in Grassmann manifold cannot be represented by a unique $n \times p$ matrix, because the same p-dimensional linear subspace in $\mathbb{R}^n$ can be represented by different $n \times p$ orthogonal matrices. So, a point on the Grassmann manifold maps to a subset of points in the Stiefel manifold and Grassmann manifold itself is a collection of such subsets. Therefore, the Grassmann manifold is the quotient space of a Stiefel manifold, mathematically $\mathcal{G}_{n,p} = \mathcal{V}_{n,p}/\mathcal{O}_n$. In a similar fashion, the Stiefel manifold can be defined as the quotient space of the orthogonal group. Two $n \times n$ matrices in $\mathcal{O}_n$ map to the same point in $\mathcal{V}_{n,p}$ if their first p columns are identical. With these observations, the Stiefel and Grassmann Manifold can be re written as

$$\begin{aligned} \mathcal{V}_{n,p} &= \mathcal{O}_n/\mathcal{O}_{n-p} \\ \mathcal{G}_{n,p} &= \mathcal{O}_n/\left(\mathcal{O}_p \times \mathcal{O}_{n-p}\right) \end{aligned} \tag{3.6}$$

The closeness of two points on the Grassmann manifold is measured using the geodesic distance. It is the curve of shortest length between the two points on a Grassmann manifold. There are several distance measures for the Geodesic distance, some using principal angles and others using orthonormal basis of the subspace [17, 79]. In this particular work the

chordal distance [14] is used as a measure for the geodesic distance. The chordal distance is given by:

$$d_c(f, g) = ||\sin\theta||_2 \tag{3.7}$$

where θ is vector of principal angles between the two subspaces f and g given by Equation (3.5).

However, as will be shown here, the traditional HOSVD will not typically be useful for action classification problems. According to Equation (3.1), given a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ the mode-k unfolding is $\mathcal{A}_{(k)} \in \mathbb{R}^{n_k \times m_k}$ and the mode-k orthogonal matrix is $\mathbf{U}^{(k)} \in \mathbb{R}^{n_k \times n_k}$. Because in action classification applications $n_k < m_k$, $\mathbf{U}^{(k)}$ is a point on the orthogonal group. It is shown above that the Grassmann manifold is a quotient space of the orthogonal group, and all the points in the orthogonal group map to the same point on the Grassmann manifold. Therefore, the distance between different mode-k orthogonal matrices is always zero. On the other hand, $\mathbf{V}^{(k)} \in \mathbb{R}^{m_k \times m_k}$ in Equation (3.1) is the orthogonal matrix spanning the row space of $\mathcal{A}_{(k)}$, and it spans only n_k dimensions because $n_k < m_k$. Therefore, $\mathbf{V}^{(k)}$ can represented by an $m_k \times n_k$ orthogonal matrix and it is a point on the Grassmann manifold. So the traditional HOSVD is modified by using $\mathbf{V}^{(k)}$ instead of $\mathbf{U}^{(k)}$ from Equation (3.1). Modified HOSVD of $\mathcal{A}$ is given by :

$$\mathcal{A} = \mathcal{S} \times_1 \mathbf{V}^{(1)} \times_2 \mathbf{V}^{(2)} \cdots \times_d \mathbf{V}^{(d)} \tag{3.8}$$

where $\times_k$ denotes the mode-k product, $\mathbf{V}^{(1)}$, $\mathbf{V}^{(2)}$, $\ldots$, $\mathbf{V}^{(d)}$ are orthogonal matrices spanning the row space associated with the the non-zero singular values given by Equation (3.1), $\mathcal{S} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is called the core tensor.

Interpretation: How are points on the Grassmann manifold related to the video tensors? What is the distance between two video tensors? Consider a 3rd order tensor $\mathcal{A}$ representing a video. The modified HOSVD of $\mathcal{A}$ can then be expressed as:

$$\mathcal{A} = \mathcal{S} \times_1 \mathbf{V}^{(1)}_{hm} \times_2 \mathbf{V}^{(2)}_{vm} \times_3 \mathbf{V}^{(3)}_{app}$$

where $\mathbf{V}^{(1)}_{hm}$, $\mathbf{V}^{(2)}_{vm}$, and $\mathbf{V}^{(3)}_{app}$ are orthogonal matrices corresponding to the horizontal motion, vertical motion, and appearance, respectively. Modified HOSVD of $\mathcal{A}_1$ and $\mathcal{A}_2$ gives mode-k

singular matrices ${\mathbf{V}_1}^{(k)}$ and ${\mathbf{V}_2}^{(k)}$, respectively. The mode-k orthogonal matrices ${\mathbf{V}_1}^{(k)}$ and ${\mathbf{V}_2}^{(k)}$ are represented by points on the Grassmann manifold and the distance between these subspaces are computed using the geodesic distance given in Equation (3.7).

3.2.4 Product Manifold

Modified HOSVD on a d order tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ generates d orthogonal matrices $\mathbf{V}^{(1)}, \mathbf{V}^{(2)}, \ldots, \mathbf{V}^{(d)}$, one for each of d unfoldings of the tensor. Each of the d orthogonal matrices is represented by a point on d different Grassmann manifolds. Product manifolds allow us to generalize classification problem to higher dimensional spaces. A product manifold $\mathcal{M}$ is a Cartesian product of manifolds. Formally, let $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_d$ be a set of manifolds, then the product manifold is defined as:

$$\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2 \times \cdots \times \mathcal{M}_d \tag{3.9}$$

where $\times$ denotes the Cartesian product [48]. A product manifold composed by a set of Grassmann manifolds is called Grassmann Product Manifold (GPM). It has been shown in [5, 49] that the geodesic distance in a product manifold is defined as:

$$d_{\mathcal{M}}(\mathcal{P}, \mathcal{Q}) = ||\sin \Theta||_2 \tag{3.10}$$

where $\mathcal{P}$ and $\mathcal{Q}$ are order-d tensors and $\Theta = (\theta_1, \theta_2, \ldots, \theta_d)$, where θ_k is the set of principal angles computed on the manifold $\mathcal{M}_k$.

Interpretation: How does the geodesic distance on product manifold help in action classification? Let $\mathcal{A}$ and $\mathcal{B}^i$ be two 3rd order tensors representing query and target videos respectively. $\mathcal{A}$ and $\mathcal{B}^i$ can be represented by points on a GPM using Equation (3.8) and Equation (3.9). Action classification is then performed as follows:

$$i^* = \underset{i \in target}{\operatorname{argmin}}\, d_{\mathcal{M}}(\mathcal{A}, \mathcal{B}^i) \tag{3.11}$$

i^* identifies the action of the query video.

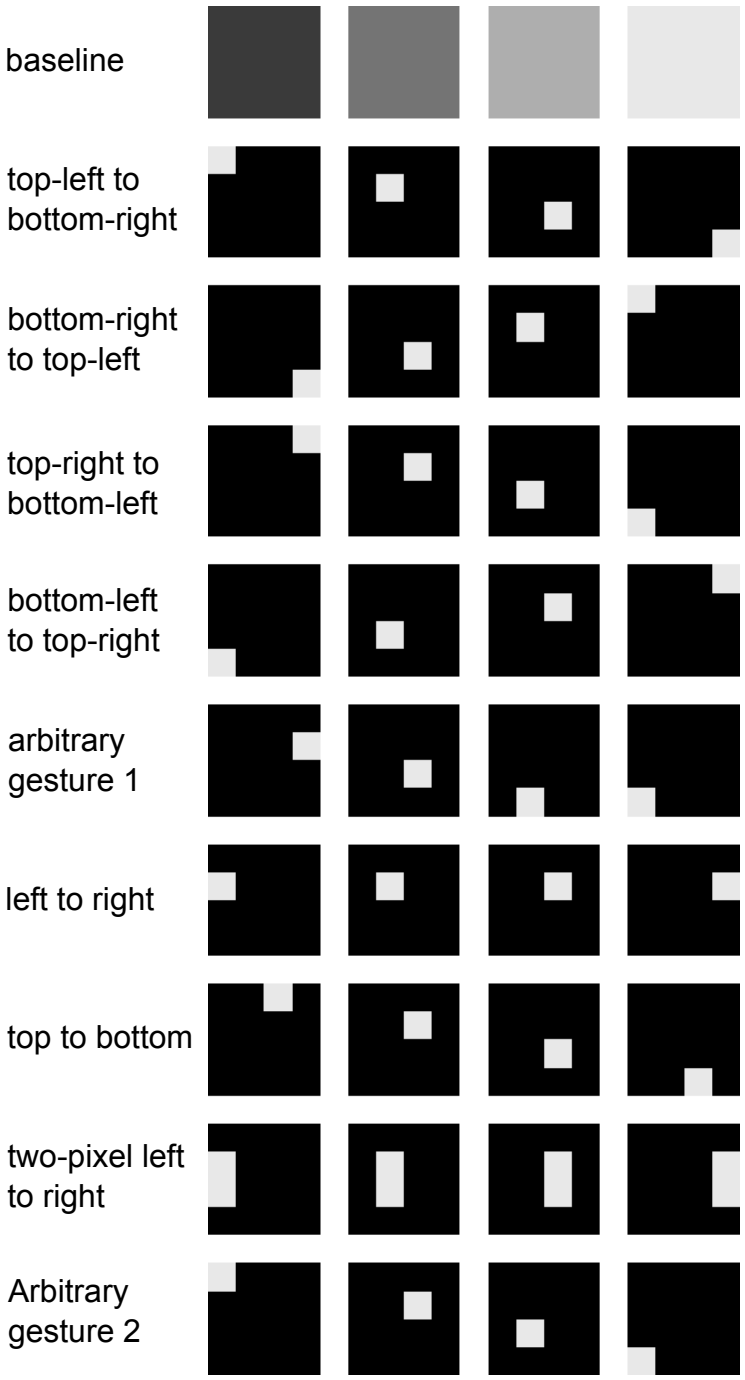


Figure 3.1: Example gestures from the synthetic gesture database. Each of the ten rows correspond to an action. From the top the gestures are: baeline (only increasing brightness), top-left to bottom-right, bottom-right to top-left, top-right to bottom-left, bottom-left to top-right, arbitrary gesture 1, left to right, top to bottom, two-pixel left to right, arbitrary gesture 2.

3.3 Action Classification on Product Manifolds: A Toy Problem

This section discusses a state-of-the-art approach for action classification from a recent paper by Lui et al. [48]. The goal of this section is to provide a tutorial level introduction to Lui's Action Classification On Product Manifolds (ACOPM) algorithm using a toy problem. Lui represents videos as 3rd order tensors and computes three orthogonal matrices using the modified HOSVD given in Equation (3.8). Each of the three matrices are represented by a Grassmann manifold. The Grassmann manifolds are combined using Equation (3.9) to map the videos to a single point on the GPM. Classification is performed using the geodesic distance on GPM coupled with a simple nearest neighbor classifier as shown in Equation (3.11). The most important contribution of Liu is to show that the action classification problem can be studied by relating tensors on a product manifold.

A synthetic data set is generated for the toy problems. Each synthetic video contains 4 frames with a fixed frame size of 4×4. Pixel intensity ranges from 0 (black) to 1 (white). There are 10 synthetic gestures resulting from 10 different movements of a white pixel in a black background. Examples of these synthetic gestures are given in Figure 3.1. In the first toy problem a simple synthetic video is used as the query video. In the second toy problem, white Gaussian noise is added to the synthetic video used in the first toy problem.

3.3.1 Example Toy Problem 1

The 5^{th} gesture in the synthetic data set is arbitrarily chosen as the query video in the first toy problem. This video can be represented using a 3rd order tensor $\mathcal{A} \in \mathbb{R}^{4\times4\times4}$ as shown below. The gray scale representation of the tensor, unfolded tensors, and the corresponding mode-k orthogonal matrices are shown in Figure 3.2.

$$\rightarrow \text{First Frame} : \mathcal{A}(:,:,1) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \rightarrow \text{Second Frame} : \mathcal{A}(:,:,2) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rightarrow \text{Third Frame} : \mathcal{A}(:,:,3) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow \text{Fourth Frame} : \mathcal{A}(:,:,4) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The unfolded (forward-cyclic) matrices are shown below:

$$\mathcal{A}_{(1)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathcal{A}_{(2)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathcal{A}_{(3)} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

The singular matrices of the above unfolded matrices computed using Equation (3.8) are show below:

$$\mathbf{V}^{(1)} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{V}^{(2)} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{V}^{(3)} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

3.3.2 Example Toy Problem 2

The query video for this toy problem is the same as from toy problem 1 except for the addition of white Gaussian noise with a $mean = 0$ and $variance = 0.1$. The flattened matrices and the corresponding orthogonal matrices are not printed here because of the page size constraint. However, the grey scale representation of the flattened matrices and the corresponding orthogonal matrices are shown in Figure 3.3. This toy video is given by the 3rd order tensor $\mathcal{A} \in \mathbb{R}^{4\times4\times4}$ shown below:

$$\rightarrow \text{First Frame} : \mathcal{A}(:,:,1) = \begin{pmatrix} 0.07 & 0 & 0.01 & 0.28 \\ 0.21 & 0 & 0 & 0 \\ 0 & 0.72 & 0.11 & 0 \\ 1 & 0 & 0.56 & 0.26 \end{pmatrix} \quad \rightarrow \text{Second Frame} : \mathcal{A}(:,:,2) = \begin{pmatrix} 0 & 0.05 & 0.31 & 0.47 \\ 0 & 0.64 & 0.32 & 0.07 \\ 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0 & 0.54 \end{pmatrix}$$

$$\rightarrow \text{Third Frame} : \mathcal{A}(:,:,3) = \begin{pmatrix} 0.31 & 0.03 & 0.21 & 0 \\ 0.19 & 0.08 & 1 & 0 \\ 0.13 & 0.05 & 0 & 0 \\ 0 & 0 & 0.30 & 0.16 \end{pmatrix} \quad \rightarrow \text{Fourth Frame} : \mathcal{A}(:,:,4) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0.19 & 0.20 & 0.67 & 0.31 \\ 0.12 & 0 & 0 & 0 \\ 0 & 0.18 & 0 & 0.19 \end{pmatrix}$$

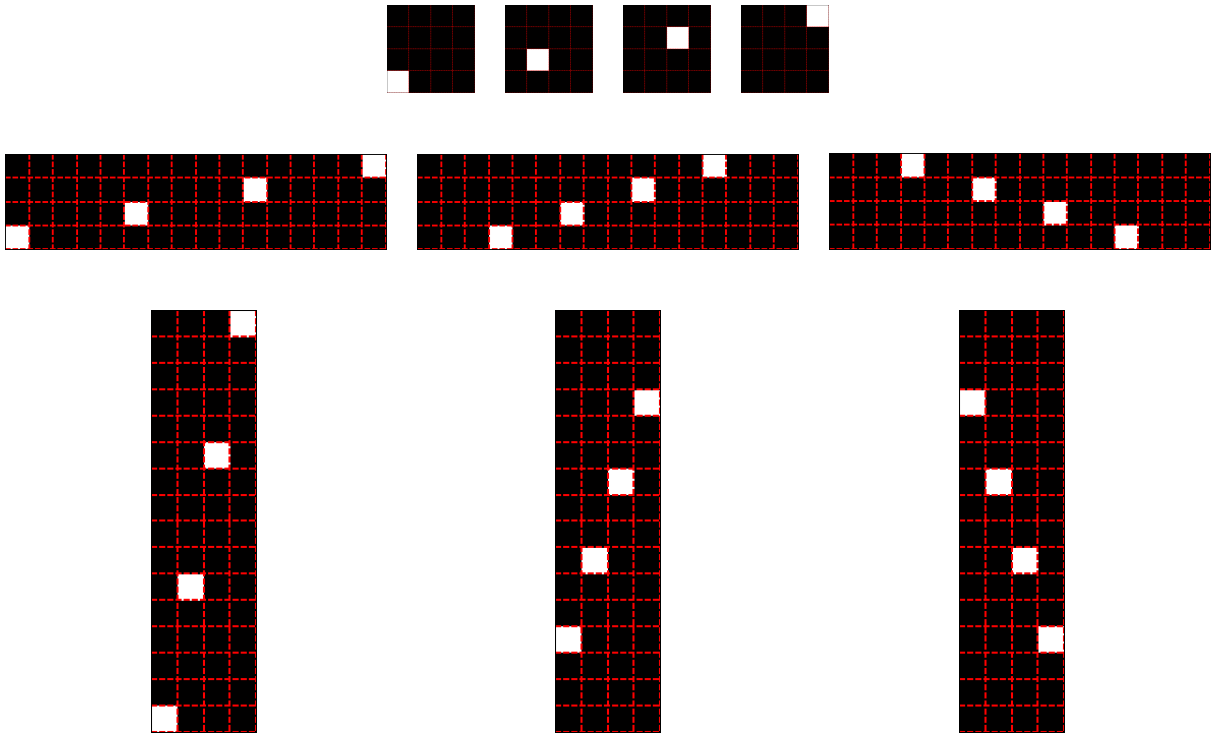


Figure 3.2: Gray scale representation of tensor decomposition of a synthetic video $\mathcal{A} \in \mathbb{R}^{4\times4\times4}$. First row shows the four frames of the synthetic video - $\mathcal{A}(:,:,1)$, $\mathcal{A}(:,:,2)$, $\mathcal{A}(:,:,3)$, and $\mathcal{A}(:,:,4)$. Second row represents the three unfoldings - $\mathcal{A}_{(1)}$, $\mathcal{A}_{(2)}$, and $\mathcal{A}_{(3)}$. Third row represents the factors obtained from modified HOSVD - $\mathbf{V}^{(1)}$, $\mathbf{V}^{(2)}$, and $\mathbf{V}^{(3)}$.

The singular matrices $\mathbf{V}^{(1)}$, $\mathbf{V}^{(2)}$, and $\mathbf{V}^{(3)}$ shown in Figure 3.3 are points on the Grassmann manifolds $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$, respectively. Let $\mathcal{B}_i \in \mathbb{R}^{4\times4\times4}$ for $i = 1, 2, \ldots, 10$ be the target gestures. The orthogonal matrices of the i^{th} target gesture $\mathbf{V}_i{}^{(1)}$, $\mathbf{V}_i{}^{(2)}$, and $\mathbf{V}_i{}^{(3)}$ are also points on the Grassmann manifolds $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$, respectively. Let θ_k^j be the j^{th} principal angle between the subspaces spanned by $\mathbf{V}^{(k)}$ and $\mathbf{V}_i{}^{(k)}$ computed on the Grassmann manifold $\mathcal{M}_k$, as defined by the Equation (3.5). Each query video is compared with each of the 10 target videos and the principal angles are shown in Table 3.1.

From Table 3.1 it can be seen that principal angles of the 5th column are relatively the smallest indicating that the query video is very similar to the 5th target gesture. The geodesic distance on GPM given by Equation (3.10) is computed for each of the 10 target gestures. The plot of geodesic distances between the query video and the target videos is

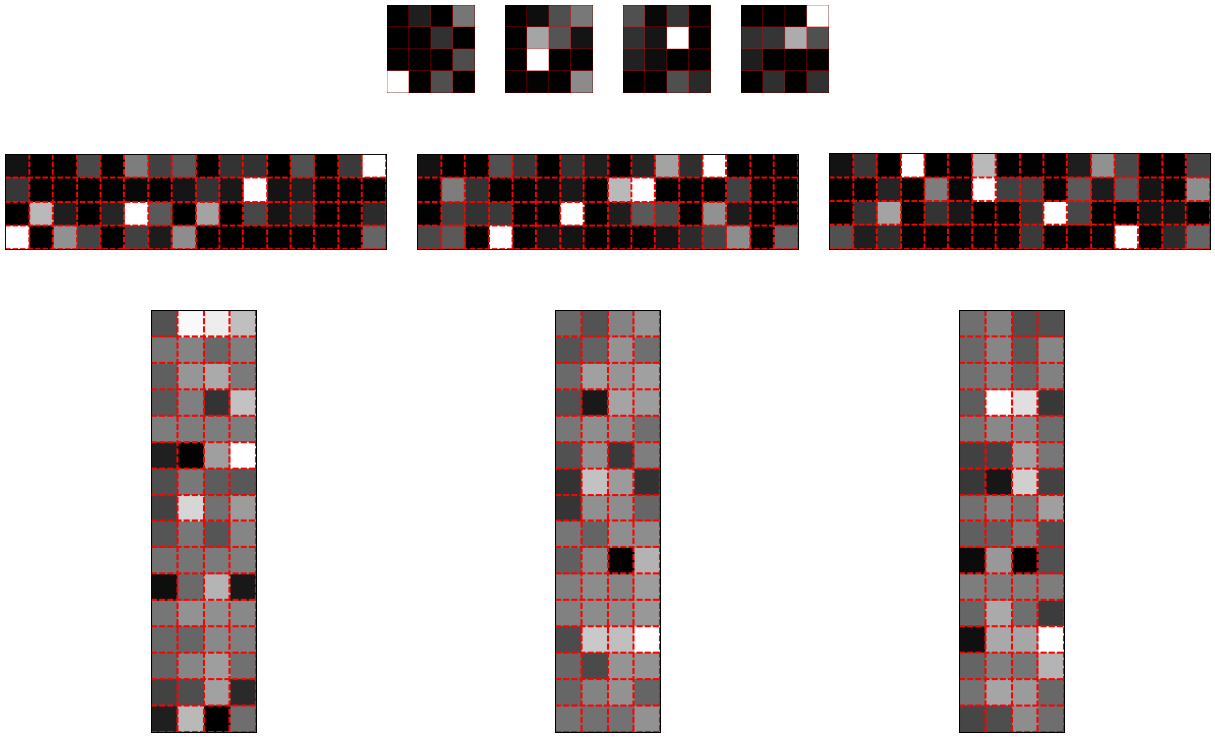


Figure 3.3: Gray scale representation of tensor decomposition of a synthetic video $\mathcal{A} \in \mathbb{R}^{4\times4\times4}$ with white Gaussian noise. First row shows the four frames of the synthetic video - $\mathcal{A}(:,:,1)$, $\mathcal{A}(:,:,2)$, $\mathcal{A}(:,:,3)$, and $\mathcal{A}(:,:,4)$. Second row represents the three unfoldings - $\mathcal{A}_{(1)}$, $\mathcal{A}_{(2)}$, and $\mathcal{A}_{(3)}$. Third row represents the factors obtained from modified HOSVD - $\mathbf{V}^{(1)}$, $\mathbf{V}^{(2)}$, and $\mathbf{V}^{(3)}$.

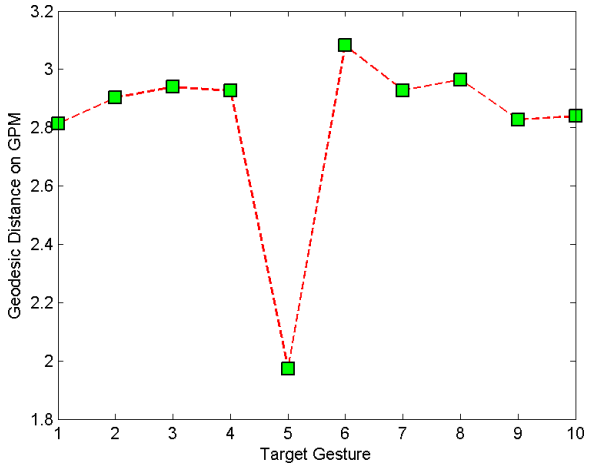


Figure 3.4: Plot of geodesic distance on Grassmann product manifold for toy problem 2.

Table 3.1: Table of principal angles between the subspaces of the query video and the target videos. Here θ_k^j is the j^{th} principal angle between the subspaces spanned by $\mathbf{V}^{(k)}$ and $\mathbf{V}_i{}^{(k)}$ computed on the Grassmann manifold $\mathcal{M}_k$ where i is the target gesture.

Target Gesture	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
θ_1^1	0.38	0.21	1.14	1.14	0.21	0.60	0.37	0.25	0.48	0.61
θ_1^2	0.69	0.52	1.34	1.34	0.52	1.15	0.49	1.07	0.62	1.20
θ_1^3	1.35	0.66	1.36	1.36	0.66	1.35	1.37	1.28	0.99	1.35
θ_1^4	1.51	0.90	1.57	1.57	0.90	1.44	1.43	1.42	1.41	1.55
θ_2^1	0.58	1.00	0.24	1.00	0.24	0.71	0.26	0.54	0.70	0.55
θ_2^2	1.06	1.17	0.58	1.17	0.58	1.23	1.34	1.17	0.82	1.01
θ_2^3	1.41	1.54	0.65	1.54	0.65	1.36	1.51	1.36	1.32	1.51
θ_2^4	1.51	1.55	0.89	1.55	0.89	1.57	1.57	1.52	1.51	1.55
θ_3^1	0.51	1.05	1.05	0.45	0.45	0.57	0.60	0.59	0.53	0.46
θ_3^2	0.61	1.19	1.19	0.49	0.49	1.19	1.36	1.12	0.96	0.57
θ_3^3	1.16	1.41	1.41	0.60	0.60	1.45	1.51	1.26	1.38	0.80
θ_3^4	1.48	1.56	1.56	0.84	0.84	1.57	1.57	1.41	1.50	1.30

shown in Figure 3.4. Using a simple nearest neighbor classifier the query video is correctly classified as the fifth gesture.

Chapter 4

Element Rearrangement Problem

4.1 Why Rearrange Elements?

Conventional tensor-based classification algorithms unfold tensors into matrices using the standard mode-k unfoldings and perform classification using established machine learning algorithms. These methods assume that the standard mode-k unfolded matrices are the best 2-dimensional representations of N-dimensional structures. A state-of-the-art example of tensor-based action classification is from a recent paper by Liu et al. [48]. Liu represents a video as a third order tensor and applies modified higher order singular value decomposition to generate three orthogonal matrices, one for each of the three standard unfoldings of the 3-dimensional structure into a 2-dimensional structure. These orthogonal matrices span subspaces that capture the variations of the row space of the three standard unfolded matrices. The success of such tensor-based methods heavily depends on the extent to which discriminating information is captured in the rows of the standard unfolded matrices. The work by Liu [48] assumes that the standard unfolded matrices are the best 2-dimensional representations of 3-dimensional structures for the purpose of classification. This observation gives rise to an important question: "Is there a better way to unfold a tensor?"

In this work, new unfoldings of a tensor are developed by rearranging elements in the original tensor and then applying the standard mode-k unfoldings. The goal of the element rearrangement problem is to reduce the intra-class distance and increase the inter-class distance, so that better classification accuracy can be achieved for action classification on product manifolds.

To understand how element rearrangement is performed, it helps to first understand the indexing scheme for the position of a tensor element. The position of a tensor element in

$\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is denoted as $(i_1, i_2, \ldots, i_d)$, and its global position g is given by:

$$g_{(i_1,\ldots,i_d)} = 1 + \sum_{s=1}^{d} \left((i_s - 1) \prod_{t=1}^{s-1} n_t \right) \tag{4.1}$$

In this work the rearrangement operator is defined as a vector $R \in \mathbb{R}^{N \times 1}$ where $N = n_1 \times n_2 \times \cdots \times n_d$ is the total number of elements in the tensor. R is also called the "rearrangement vector" and stores the global position for all the locations in a tensor. Element rearrangement is effected by rearranging the elements in R. This is essentially the N-permutation of N elements in R and there are $N!$ possible rearrangements. For the Cambridge-Gesture data set $\mathcal{A} \in \mathbb{R}^{20 \times 20 \times 32}$, global index simplifies to $g_{(i_1,i_2,i_3)} = i_1 + 20(i_2 - 1) + 400(i_3 - 1)$, rearrangement vector $R \in \mathbb{R}^{12800 \times 1}$, and there are 12800! possible rearrangements.

4.2 Element Rearrangement using Local Search

4.2.1 Combinatorial Optimization

In this section we show that the element rearrangement problem is essentially a combinatorial optimization problem. We now review some notions of combinatorial optimization problems, for more detailed descriptions refer to [52, 54].

Let S be a finite set of candidate solutions and $f : S \mapsto \mathbb{R}$ a function which assigns a value $f(s)$ to every $s \in S$. The goal of a combinatorial optimization problem is to find a solution $s_{opt} \in S$ such that

$$f(s_{opt}) = \max_{s \in S} f(s) \tag{4.2}$$

S is called a search space, $f(s)$ is called the objective function or fitness function, and the pair (S, f) is called an instance of a combinatorial optimization problem. The sought after s_{opt} is called a globally optimal solution of (S, f), and the set $S_{opt} \subseteq S$ all returning the same optimal fitness value is the set of all globally optimal solutions.

The element rearrangement problem for action classification on product manifolds is defined over a discrete search space S of possible rearrangements of pixels in a 3rd order tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$. In other words, search space S is the N-permutation of N objects, where $N = n_1 \times n_2 \times n_3$. Therefore, S contains $N!$ elements. The objective function for the

element rearrangement problem is the action classification on product manifolds algorithm given in [48]. Given a candidate solution s from the search space S this algorithm gives a classification rate (also called objective function value or fitness value). The goal of the element rearrangement problem is to find the optimal rearrangement s_{opt} given by Equation (4.2).

4.2.2 Local Search

Many algorithms have been developed for solving combinatorial optimization problems. These algorithms can be grouped into exact and approximate algorithms. Although, exact algorithms guarantee an optimal solution in bounded time for every finite size instance of the problem, no polynomial time algorithm exists for $\mathcal{NP}$-hard problems [52, 54]. For finite size optimization problems, a simple exact algorithm approach is to enumerate the full search space. For the Cambridge-Gesture data set search space S contains 12800! elements. Therefore, a completely exhaustive method for finding the optimal solution is out of the question. In approximate methods we trade-off the guarantee of finding an optimal solution for finding a good solution in a significantly reduced amount of time. Approximate algorithms can be further classified as either constructive methods or local search methods. Constructive methods are typically fast but often return solutions inferior to local search methods [10].

In this thesis local search is adapted to solve the element rearrangement problem. Local search methods are often prematurely dismissed because of the presence of multiple local optima in the search space and lack of theoretical understanding. However, important observations have been made by Tovey [64] about multiple local optima and by Johnson et al. [31] about the complexity of local search. In the last three decades local search has been widely accepted as an effective method of solving hard combinatorial problems [1, 62].

Conceptually, local search is simple. A neighborhood structure $\mathcal{N}(s)$ is defined in the search space. Formally, a neighborhood structure is a function $\mathcal{N}(s) : S \mapsto 2^S$ that assigns to every $s \in S$ a set of neighbors $\mathcal{N}(s) \subset S$. $\mathcal{N}(s)$ is called the neighborhood of s. Local search starts from some initial solution and iteratively replaces the current solution by a better solution in the the neighborhood of the current solution until a solution is found

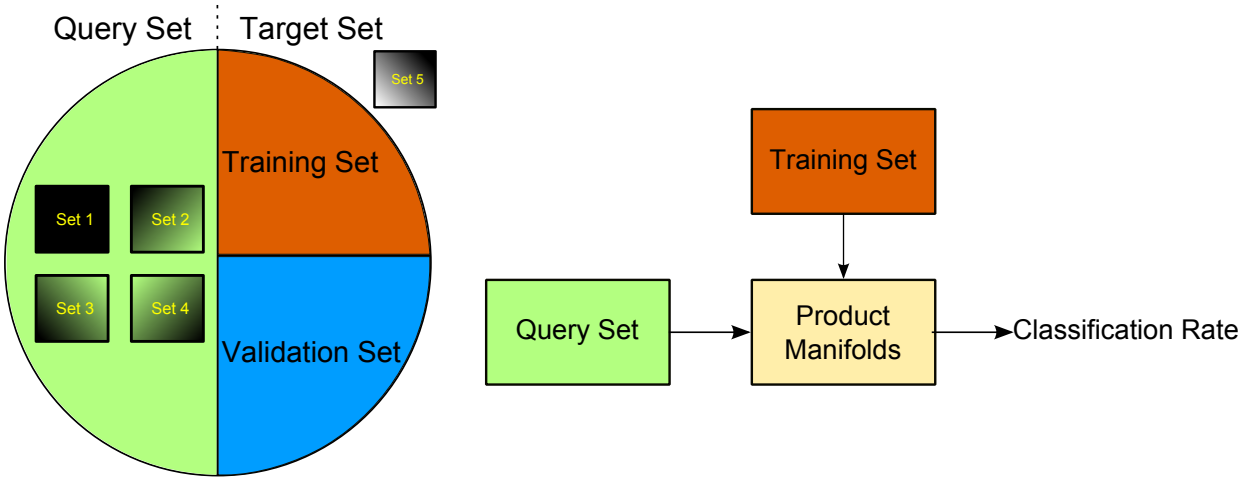


Figure 4.1: Action classification on product manifolds. Cambridge-Gesture data set is partitioned into query set (Set1, Set2, Set3, and Set4) and target set (Set5). Target set is the labeled data set and is further randomly divided into training set and validation set. Query set is identified by the ACOPM.

which is better than all its neighbors. In random starts local search, multiple local searches are initiated with initial solutions randomly drawn from the search space. This approach increases the probability of finding a near optimal solution.

4.3 A General Algorithm

In this section we present a general algorithm of the method proposed in this work. The original ACOPM method is used as the objective function and follows the experimental protocol given in [33]. The Cambridge-Gesture data set is divided into query set and target set. The Query set consists of Set1, Set2, Set3, and Set4. The Target set consists of Set5 and is further randomly divided into training set (10 samples per action) and validation set (10 samples per action) as shown in Figure 4.1. Figure 4.1 uses the same training data as in Figure 2.2, and divides them in the manner explained above. Because ACOPM does not require prior training the validation set is discarded. A single evaluation of ACOPM employs five random repetitions in selecting training and validation sets in the target set. And the output of the ACOPM is a classification rate which is the average accuracy obtained across the five repetitions. The original ACOPM algorithm has classification rate of 88% [48]. A complete description of the ACOPM algorithm is given in Section 3.3.

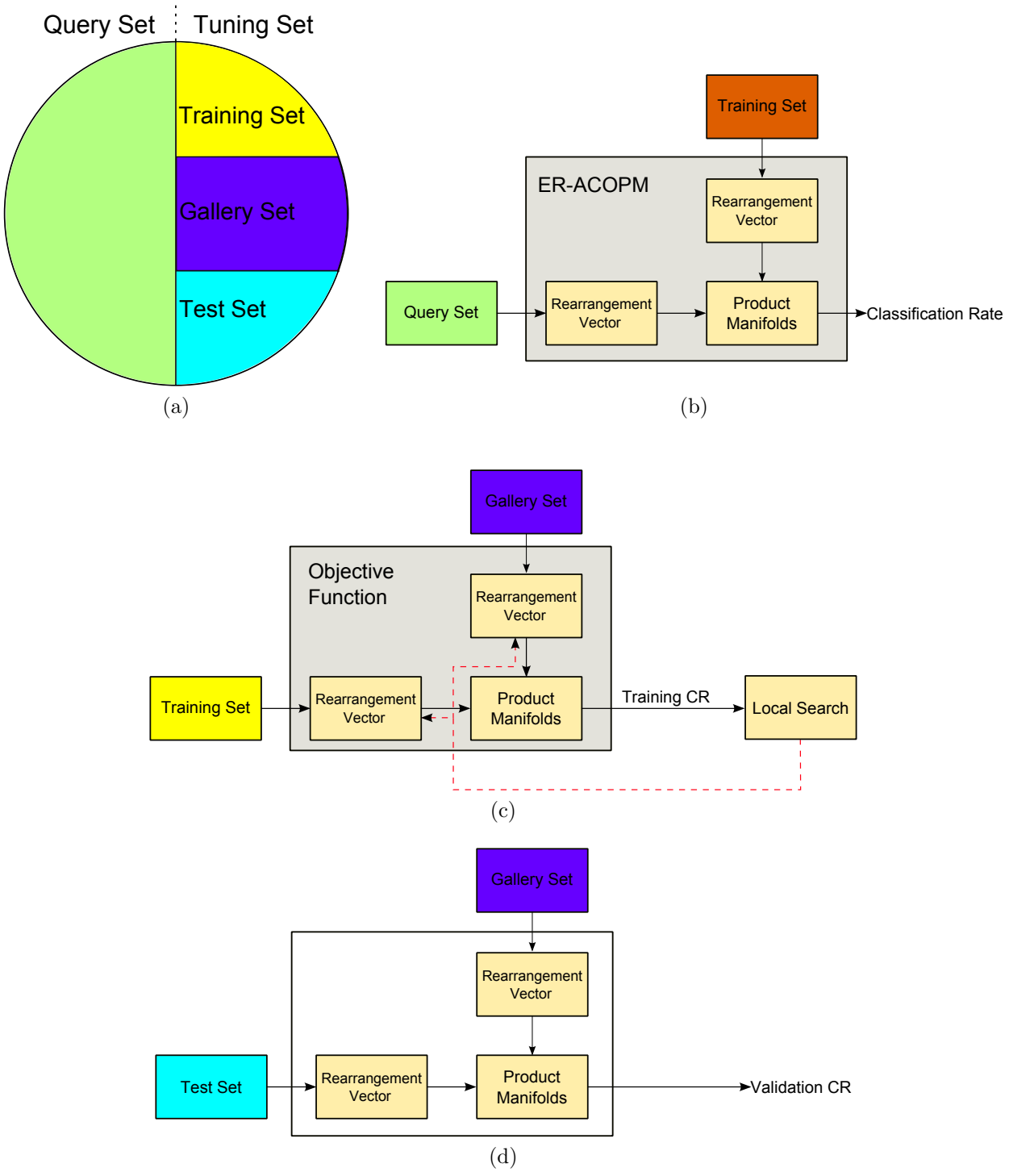


Figure 4.2: A general algorithm for element rearrangement for action classification on product manifolds (ER-ACPOM): (a) Cambridge-Gesture data set is partitioned into query set and tuning set. Tuning set is further randomly divided into training set, test set, and gallery set; (b) Query set is identified by the ER-ACOPM; (c) Local search is used on the training set to find the optimal rearrangement vector; and, (d) Performance of the local search method is evaluated the on the test set.

Although some of the details may vary depending on the variation of the local search being used, a basic algorithm for rearranging elements using local search methods is given below:

1. Initialization

   The Cambridge-Gesture data set is divided into query set and target set. Query set consists of four out of the five sets and target set consists of the remaining set. Target set is used to learn the rearrangement vector, therefore, in this context it is called the tuning set. Tuning set is further randomly divided into training set (7 samples per action), test set (7 samples per action), and gallery set (6 samples per action) as shown in Figure 4.2 (a). Gallery set is the labeled set. The local search method works on the training set and the performance of the local search method is evaluated on the test set as shown in Figure 4.2 (c) and 4.2 (d), respectively. Neighborhood structure $\mathcal{N}(\mathtt{s})$ is defined, and initial rearrangement vector R_{init} is randomly chosen from the search space.

2. Local Search

   Compute the objective function value for the current solution s and all the candidate solutions in the neighborhood $\mathcal{N}(\mathtt{s})$. The objective function value $f(s)$ is computed in two steps as shown in Figure 4.2 (c). First, the videos in the training and gallery set are rearranged using the rearrangement vector. Second, the classification rate is computed using ACOPM algorithm explained in the beginning of this section. Depending on the variant of local search method (e.g., hill climb, tabu search) being used choose the best rearrangement vector.

3. Termination

   Repeat step 2 till the stopping condition is reached. For a simple hill climb algorithm the repetition stops when the current solution is better than all its neighbors. The best rearrangement vector at the end of the local search is denoted by R_{final}.

The above 3 steps are repeated n_{trials} (number of trials) times, each time with a randomly chosen initial solution. CR_{itrain} and CR_{ftrain} denotes the classification rates on the training set with the initial R_{init} and final R_{final} rearrangement vectors, respectively. Similarly,

CR_{ival} and CR_{fval} denotes the classification rate on the test set, and CR_{init} and CR_{final} denotes the classification rate on the query set. This is shown in Figure 4.2.

4.4 Hill Climb

A simple local search method, hill climb, is implemented to understand the terrain of the search space. In the hill climb method, the initial solution is given by the default configuration of the pixels. The neighbors of the current solution is obtained in three steps. First, randomly choose two blocks of pixels within a frame. Second, swap these two blocks. Third, swap the same blocks in all the 32 frames in the video. A greedy heuristic is used in choosing the best neighbor, that is the first neighbor which improves the current solution replaces the current solution. A greedy heuristic is used because the size of the neighborhood structure is enormously large. The number of swaps is arbitrarily set to 1000.

The hill climb experiment is repeated several times, each time with a different seed for the random number generator to generate different random numbers for the pixel swap operation. The size of the block of pixels being swapped is also varied. Each hill climb took about eight hours to complete 1000 evaluations. On an average only 10 swaps out of the 1000 swaps is useful. In other words, although a greedy heuristic is used, only 10 neighborhood structures are evaluated. For many trials there is no useful swap after 200 evaluations indicating that the hill climb may have hit a local minimum and hence reached a dead end.

From this experiment it is clear that hill climb is massively time consuming despite using a greedy heuristic and is not terribly effective. Therefore, there is a need to reduce the size of the search space and adapt a local search framework which can explore the search space beyond the local minima.

4.5 Tabu Search

A major drawback of local search is the possibility that local search might return a very poor quality local minima. Although this problem can be offset to a certain extent using random starts local search for small instances, the number of local minima may increase exponentially with the increase in problem size (and therefore increase in the search space).

Furthermore, restarts with random initial solutions does not take advantage of the search space structure.

In the last two decades, a new kind of local search method has emerged which aims at tackling the problems encountered using the random starts local search method. These methods are commonly referred to as metaheuristics. There is no commonly accepted definition for the term metaheuristic. Below we quote the definition by Stützle:

> "Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristics, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly." [62]

This class of methods include tabu search, simulator annealing, genetic algorithms, iterated local search, and ant colony optimization.

Tabu Search (TS) is adopted in this thesis. TS is conceptually more simple than simulator annealing and genetic algorithms as well as easier to implement. TS is the most common metaheuristic [10], and its most distinctive feature is the explicit, systematic use of a memory to guide the search process [62]. TS is a metaheuristic algorithm that guides a local search to avoid pitfalls and explore the search space beyond the local optimality using the search history. Emphasis on the explicit use of the memory is based on the assumption that a bad strategic choice can provide more information than a good random choice. A bad strategic choice can provide useful information for modifying the strategy to be more profitable. [24]

4.5.1 Basic Elements of Tabu Search

This section deals with basic theory needed for this thesis. Only those TS strategies used in this work are reviewed here. For a more detailed discussion on different TS strategies read [20, 22, 23, 24]. The exposition here is based on [21, 62].

4.5.1.1 Fitness Landscape

As mentioned above, TS is simply a combination of local search with explicit memory structures. It then follows that the first three basic elements of TS are the definition of search space S, neighborhood structure $\mathcal{N}(\mathtt{s})$, and objective function $f(s)$. These three elements $(S,\mathcal{N}(\mathtt{s}),f(s))$ together is referred to as the fitness landscape and their choice is the most critical step in the design of any TS. A good understanding of the problem at hand is necessary to choose the members of the fitness landscape. In recent years, several researchers have addressed the problem of theoretical analysis of a search space [11, 18].

4.5.1.2 Memory and Search Strategy

The two most commonly used memories in TS are short-term memory and long-term memory, and each memory is accompanied by its own search strategy. At each evaluation TS makes a move from the current solution to the best solution in its neighborhood even if it worsens the objective function value. When this happens it is important to avoid immediately returning to the previous solution. This cycling can be prevented by declaring tabu (forbidding) on the recent moves. This restricts the neighborhood of the current solution s, and $\tilde{\mathcal{N}}(\mathtt{s})$ is the admissible subset of $\mathcal{N}(\mathtt{s})$. These tabus are stored in the short-term memory (also called tabu list). In many problems, solution components called attributes are

are stored in the the short-term memory instead of the entire solution. For example, if a move is defined by swapping two elements then the labels of these two elements (attributes) are used to enforce the tabu status.

The tabu tenure tl determines the number of evaluations for which the tabu status is active. Alternatively, the length of the short-term memory is given by tl. It is possible that TS might forbid attractive moves even when there is no danger of cycling. Aspiration criterion provides an algorithmic way to revoke the tabu status. A commonly used aspiration criterion is to drop the tabu status when the move leads to a solution better than the best-known solution.

More often than not long-term memory is needed to make TS fully effective. The two strategies associated with long-term memory are intensification and diversification. The goal of intensification strategies is to thoroughly explore the promising regions of the search the space by revisiting the elite solutions. On the other hand, the goal of diversification is to explore new regions of the search space. A common long-term memory strategy is based on the number of times each attribute have been present in the selected moves. Such long-term memory is called the frequency memory. In most situations the search performed by TS is thorough enough, and sometimes they tend to be too local. Therefore diversification is possibly more crucial then intensification in the design of TS. Some intermediate and advanced TS strategies include candidate lists, surrogate objective functions, and allowing infeasible solutions.

4.5.2 Algorithm

TS procedures are extremely sensitive to the parameter settings. TS procedure for the element rearrangement problem along with a brief description of the different parameters is

discussed in the rest of this chapter. Chapter 5 provides a more detailed discussion on the performance benefits of different parameter settings. Following are the four steps to solve the element rearrangement problem using TS:

1. Initialization

   In the first step several important parameters including number of random starts n_{trials}, number of evaluations n_{evals}, tabu tenure tl, and tuning set are defined (Section 4.5.3). The search space is reduced by pixel sampling all the videos in the Cambridge-Gesture data set (Section 4.5.4). The neighborhood structure $\mathcal{N}(\mathtt{s})$ and initial solutions for the all the n_{trials} are generated (Section 4.5.4). Tuning set is partitioned as show in Figure 4.2(a). TS method works on the training set and the performance of the TS method is evaluated on the test set as shown in Figure 4.2 (c) and 4.2 (d), respectively.

2. Best Move Decision

   The objective function value of the current solution and all its neighbors are computed (as explained in Section 4.3). Based on the short-term and long-term memory strategies (Section 4.5.5), aspiration criterion, and the objective function value the best move decision is made. Aspiration criterion adopted in this work is to drop the tabu status when the move leads to a solution better than the best-known solution.

3. Best Move Execution

   Current solution is replaced with best solution obtained in the previous step by executing the appropriate move. The tabu tenure of all the tabu moves stored in the short-term memory is reduced by one, and the current move is added to the tabu list. The long-term memory is updated by incrementing the count of the attributes involved in the current

move.

4. Termination

   Steps two and three are repeated till the termination criterion is reached. Two commonly used termination criteria in TS are: first, to stop after some number of iterations without any improvement in the objective function value and second, after a fixed number of evaluations. Both criteria are used in experiments performed in this work. But the second criterion is used for the majority of the experiments. This is because parallel programming is used to reduce the run time of TS from 30 days to under 36 hours by executing each of the n_{trials} simultaneously on a multi-core machine and this is partly made possible by having a fixed number of evaluations per trial: which ensures a good load balance.

4.5.3 Parameter Initialization

The choice of number of trials in a single TS experiment is a trade-off between probability of a near optimal solution and run time of the experiment. All the experiments in this work are performed on an Intel Xeon dual six-core server processor. Matlab's parallel computing tool box allows a maximum of twelve threads. Depending on the size of the problem the run time of a single trial varies from seven hours to seven days. Therefore, the number trials n_{trials} for all TS experiments is set to twelve. Number of evaluations n_{evals} is varied from 25 to 1000 depending on the specific search strategy employed.

The best choice of tabu tenure is determined experimentally by studying a range of tabu tenures (from 5 to 100). It might seem surprising that none of the values of tabu tenure exhibited a distinctly better performance. A possible reason is that due to the large size of

the search space and neighborhood the probability of the TS procedure cycling is very low. So, for all the experiments the tabu tenure tl is arbitrarily set to five.

As explained in Section 4.3 TS is performed on the tuning set. Cambridge-Gesture data set can be divided in multiple ways to obtain the tuning set. All action classification algorithms follow the the experimental protcol of using Set5 for the purpose of training [33, 48]. In this work also Set5 is used as the first choice of the tuning set and is represented by D_5. However, two more choices of tuning set are used in this work to better evaluate the performance of the TS procedure. Set1 is arbitrarily used as the second choice of the tuning set (D_1). For the third choice of the tuning set, four samples per action is randomly drawn from each of the 5 sets. This tuning set is represented by D_{cross}.

4.5.4 Search Space and Neighborhood Structure

The size of the search space S of the element rearrangement problem is directly related to the size of the video. As explained in Section 4.1, for a video tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ there are $(n_1 \times n_2 \times n_3)!$ elements in the search space, and the rearrangement vector is given by $R \in \mathbb{R}^{(n_1 \times n_2 \times n_3) \times 1}$. For the Cambridge-Gesture data set there are 12800! possible solutions, and the rearrangement vector $R \in \mathbb{R}^{12800 \times 1}$. To meaningfully perform TS the size of the search space needs to be pruned in an intelligent manner. This is done in two steps as described below:

1. Replication

   The size of the search space can be reduced by replicating the rearrangement of pixels within a single frame across all the frames in a video. In other words, the original rearrangement problem of considering all the possible rearrangement of the pixels in a

tensor is modified to consider only the rearrangement of the pixels within a frame and this rearrangement is replicated in all the 32 frames of the video. As a result, the number of elements in the search space reduces from 12800! to 400!, and the modified rearrangement vector $\tilde{R} \in \mathbb{R}^{400\times 1}$.

2. Pixel Sampling

   Some pixels within a frame contribute more discriminating information than other pixels. Therefore, we can reduce the size of the search space by dropping some pixels within a frame and across all the frames in a video. However, it is important to retain the tensor structure of the pixel sampled videos to take advantage of the performance benefits of processing the videos in its original tensor form. Therefore, the number of pixels per frame after sampling n_{pels} should be a perfect square. As a result of pixel sampling the number of elements in the search space is further reduced from 400! to n_{pels}! and the modified-sampled rearrangement vector is given by $\tilde{R}_{sampled} \in \mathbb{R}^{n_{pels}\times 1}$. Pixel sampling is simply choosing a subset of n_{pel} pixels from within 400 pixels in the modified rearrangement vector $\tilde{R}$. The value of n_{pels} in this work varies from 49 to 324.

   Two different approaches of pixel sampling is used in this work . In the first approach pixels are randomly sampled (image space). In the second approach pixels are chosen based on their edge strength (feature space). A simple edge detection algorithm is used on the videos from training set to compute the edge strength of each pixel in a frame. Higher the edge strength of a pixel greater the probability of being chosen. An example of the two sampling methods is show in Figure 4.3. This gives rise to two different ways of generating an initial solution. First is a random sampling of pixels and second is sampling

pixels based on their edge strength.

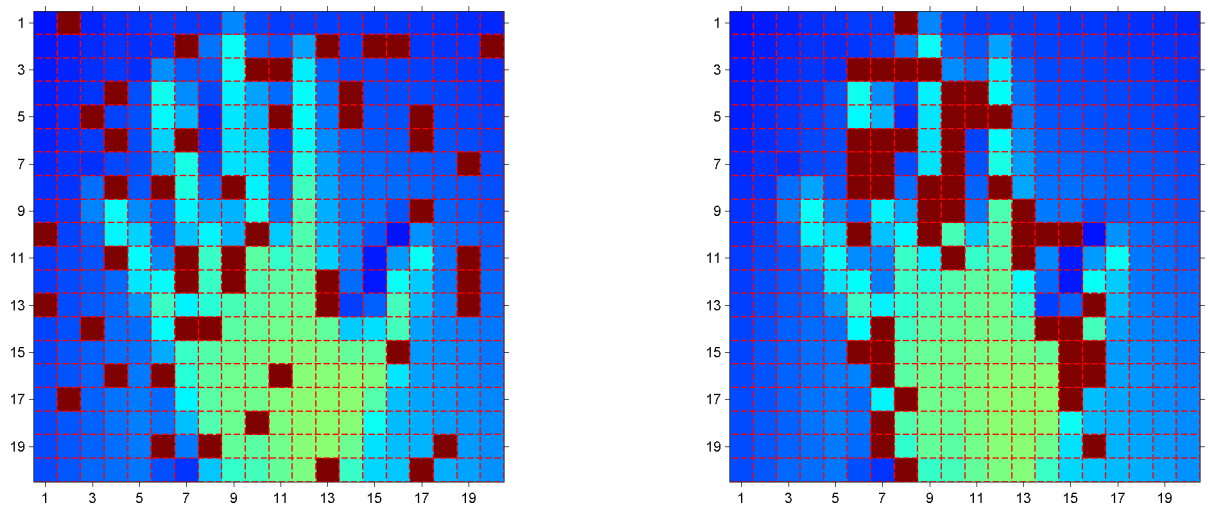


Figure 4.3: Pixel Sampling: Random pixel sampling (left) and edge pixel sampling (right). The 49 pixels in red are the chosen pixels.

In this work the neighborhood structure is defined implicitly by the possible local changes that may be applied to the rearrangement vector. This is more efficient when compared to explicitly enumerating the set of possible neighbors. There are several possible neighborhood structures for a given definition of the search space. Three neighborhood structures are considered in this work - Pixel Add-Drop Neighborhood (PADN), Pixel Add-Drop-Swap neighborhood (PADSN), and Frame Shift neighborhood (FSN). The motivation for these neighborhood structures is discussed in Chapter 5.

Pixel add-drop move is used to define PADN. Neighbors are constructed by replacing each pixel in $\tilde{R}_{sampled}$ with a pixel above it and below it. Therefore, there are $2n_{pels}$ neighbors in PADN. Consider a solution s having 49 pixels per frame as shown in Figure 4.4(a). Some neighbors of s are show in Figures 4.4(b)-(g). For example, pixel at location (2,2) in Figure 4.4(a) is replaced by the pixel above it (1,2) in Figure 4.4(b) and by the pixel below

it (3,2) in Figure 4.4(c). In some cases the pixel being added to $\tilde{R}_{sampled}$ might already be present in $\tilde{R}_{sampled}$. Such neighbors are discarded. As a result the number neighbors in PADN is usually less than $2n_{pels}$.

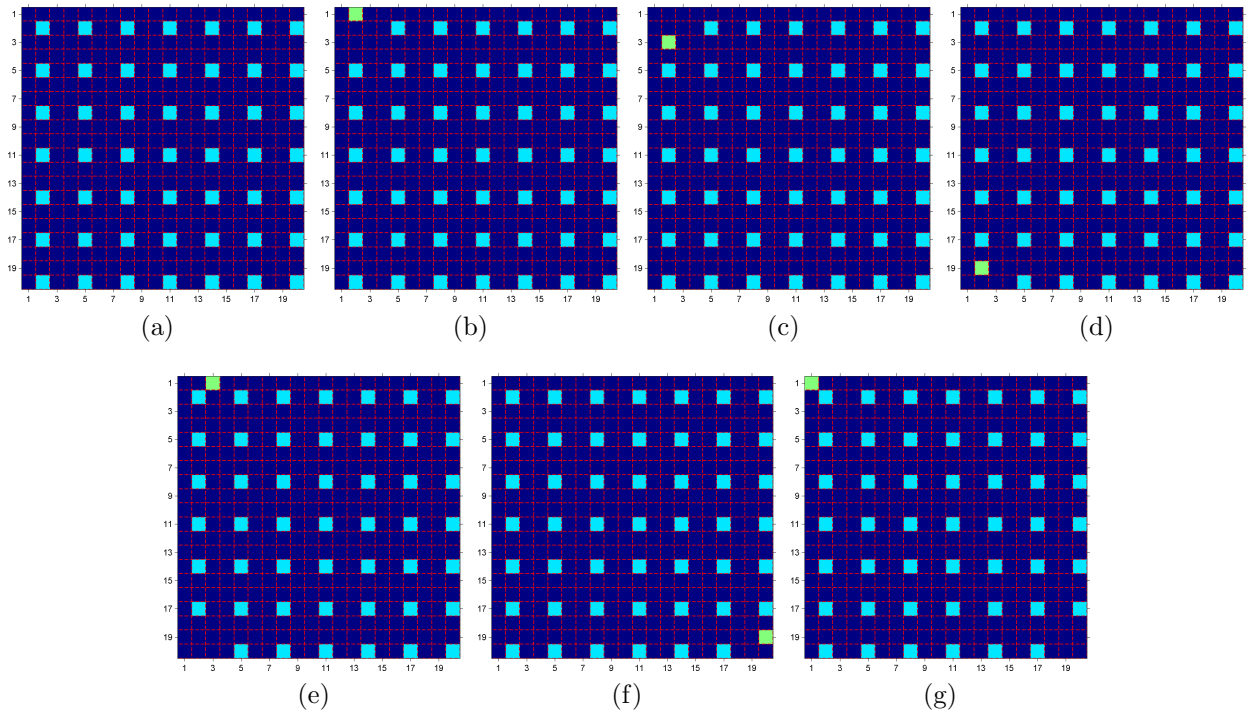


Figure 4.4: Illustration of the Pixel Add-Drop Neighborhood (PADN): (a) A solution s from the search space S. Location of 49 pixels (light blue pixels) sampled from 400 pixels are stored in $\tilde{R}_{sampled}$; and, (b)-(g) Some examples of neighbors of s obtained by replacing a pixel in $\tilde{R}_{sampled}$ with a new pixel (green pixel) either above or below the dropped pixel.

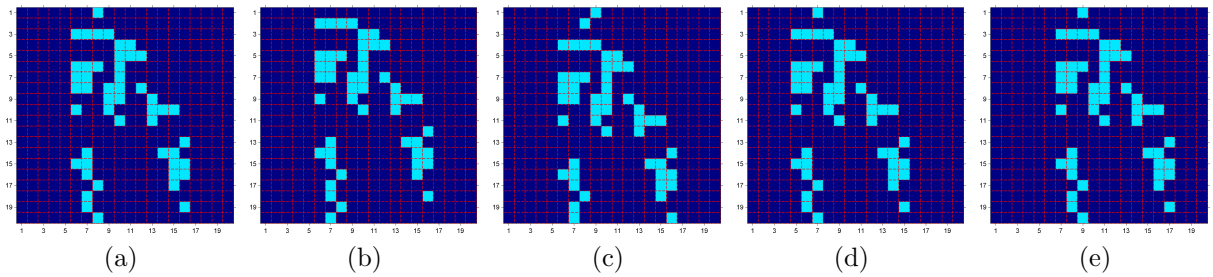


Figure 4.5: Illustration of the Frame Shift Neighborhood (FSN): (a) A solution s from the search space S. Location of the 49 pixels (light blue pixels) sampled from 400 pixels are stored in $\tilde{R}_{sampled}$; (b) Frame shift up; (c) Frame shift down; (d) Frame shift left; and, (e) Frame shift right;

PADSN is a minor variant of PADN. Along with add-drop moves this neighborhood also allows swap moves. Consider the case where the pixel (say $\bar{p}$) to replace the current pixel (say p) is already present in $\tilde{R}_{sampled}$. In such cases instead of discarding the neighbor as done in PADN the two pixels p and $\bar{p}$ are swapped. Therefore, the number of neighbors in PADSN is always $2n_{pels}$.

Unlike the previous two neighborhood definitions where the transformations are replicated in all the frames in a video, in FSN the transformation is performed separately for each frame. Neighbors are constructed by moving all the sampled pixels in a frame up, down, left, and right by one position, for each frame in the video. Therefore, there are 128 (4 moves per frame and a video has 32 frames) neighbors in FSN. Consider a solution s having 49 pixels per frame as shown in Figure 4.5(a). Some neighbors of s are show in Figures 4.5(b)-(e). A frame is shifted up by subtracting one from $\tilde{R}_{sampled}$, shifted down by adding one to $\tilde{R}_{sampled}$, shifted left by subtracting 20 (number of rows in a frame) from $\tilde{R}_{sampled}$, and shifted right by adding 20 (number of rows in a frame) to $\tilde{R}_{sampled}$. Because FSN operates on each frame separately 32 rearrangement vectors need to be maintained one for each frame.

To summarize, three neighborhood structures are proposed in this work.

1. Pixel add-drop neighborhood

2. Pixel add-drop-swap neighborhood

3. Frame shift neighborhood

The performance benefits of these 3 neighborhood structures are discussed in Chapter 5.

4.5.5 Tabu Search Strategy

As discussed before, TS is based on the assumption that a bad strategic choice can provide useful information for modifying the strategy to be more profitable. Systematic use of memory can be used to achieve this. In this work three short-term memories (tabu lists) are defined to store the tabu status (tabu tenure) corresponding to pixel add-drop moves, pixel swap moves, and frame shift moves. Three long-term memories (frequency memories) are defined to store the frequency count of each of the three moves. At the end of every evaluation the following updates are made to the memories. First, tabu tenures of all the tabu active moves are reduced by one. Second, current move is added to the appropriate tabu list. Third, the frequency count corresponding to the current move is increased by one.

Once the objective function values of all the neighbors of the current solution are computed there are several strategies to determine the best neighbor. The strategy adopted in this work is show in Figure 4.6. Each neighbor is added to the primary, secondary or discard list. The best neighbor is chosen from the secondary list only of the primary list is empty.

If a move is infeasible (e.g. duplication of pixels) then such a solution is discarded. The tabu status of the move is checked by looking into the appropriate short-term memory. If the move is tabu active (forbidden) but satisfies the aspiration criterion (the move leads to a solution better than the best-known solution) then it is added to the primary list. On the other hand, if the move is tabu active and does not satisfy the aspiration criterion it is discarded. If the move is not tabu active and its objective function value is different from the objective function value of the current solution then it is added to the primary list. On the other hand, if the move is not tabu active but its objective function value is same as

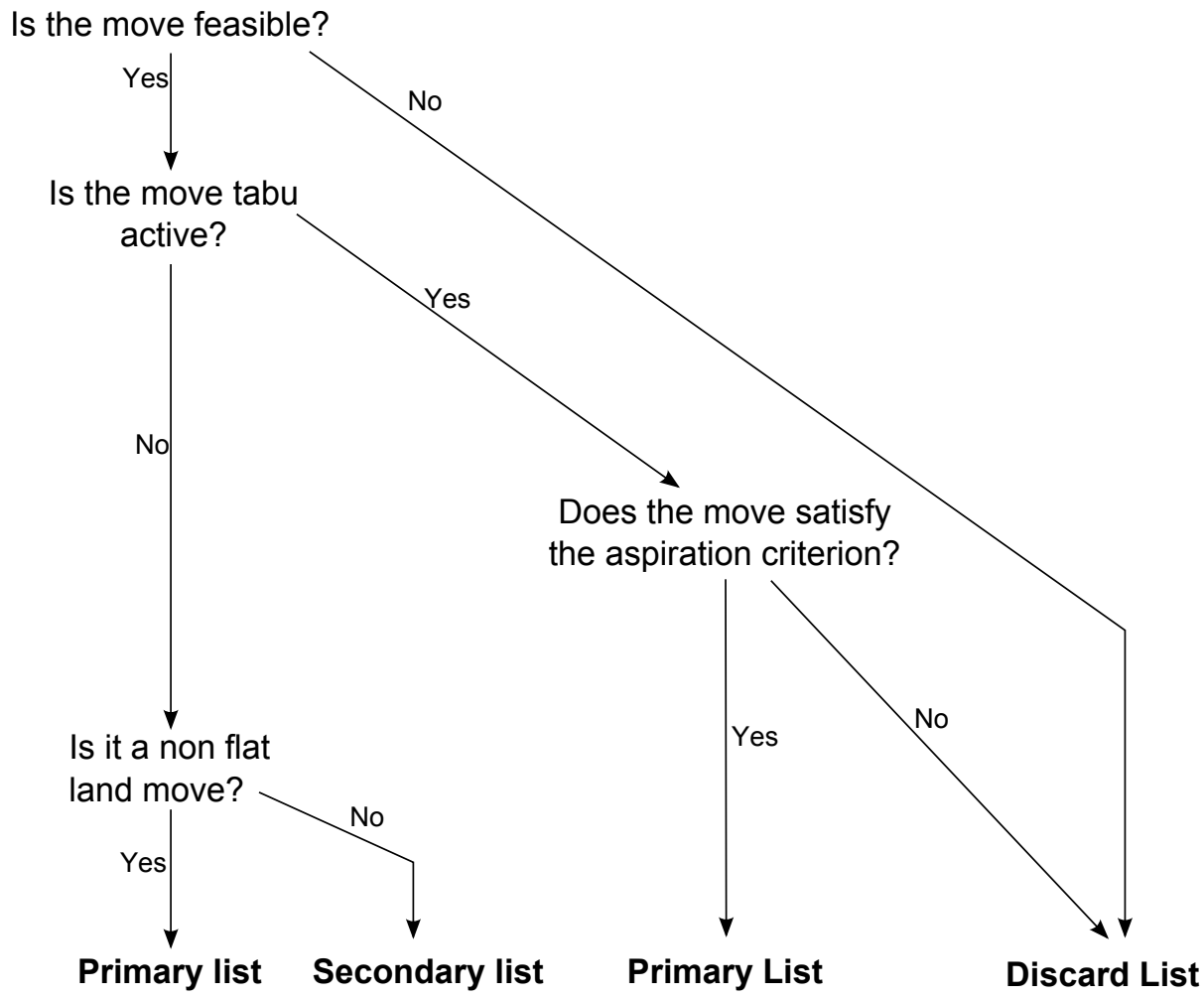


Figure 4.6: Best Move Strategy

the objective function value of the current solution it is added to the secondary list. This is done to discourage walking along a plateau.

Diversification is achieved by using the counts from the frequency memories, driving the search into new regions. The objective function values of the neighbors in the primary and secondary lists are subtracted by an amount proportional to their respective frequency counts. The proportionality constant is given by $n_{penalty} \in [0, 1]$. The diversifying influence is only restricted to cases where there are only non improving moves. Now, the move with the largest objective function value in the primary list is selected as the best move. If the

primary list is empty then the move with the largest objective function value in the secondary list is selected as the best move.

Chapter 5

Experiments

5.1 Tabu Search Tuning Procedure

As discussed in the previous chapter, TS has several parameters that need to be defined before starting the search process. TS is usually very sensitive to parameter settings. Several experiments with different parameter settings are performed. All the experiments performed in this work are grouped into four heuristics based on their search space and neighborhood structure definitions. Each of the four heuristics are incrementally developed based on the analysis of results from the previous heuristics. The general procedure followed for parameter tuning in this work is given by the following steps:

1. Define the search space S and neighborhood structure $\mathcal{N}(\mathtt{s})$ keeping in mind the problem at hand. A strong understanding of the problem is crucial in defining S and $\mathcal{N}(\mathtt{s})$. PADN is the first neighborhood definition used in this work.

2. Initialize all the other parameters including n_{trials}, n_{evals}, $n_{penalty}$, tl, tuning set, and initial solution. Implement a simple version of the TS.

3. Collect statistics, analyze the results, and vary the parameters accordingly (Heuristic 1).

4. Reconsider the definition of the search space. Edge pixel sampling is used instead of random pixel sampling (Heuristic 2).

5. Reconsider the definition of the neighborhood structure. Two new neighborhood structures are defined - PADSN (Heuristic 3) and FSN (Heuristic 4).

6. Reconsider diversification strategies.

7. Continue to experiment with the parameters.

5.2 Heuristics and Experimental Results

All the TS experiments are grouped into four heuristics. The parameters setting for these four heuristics is given in Table 5.1, and their corresponding results are show in Figures 5.1 and 5.2. The four heuristics are described below.

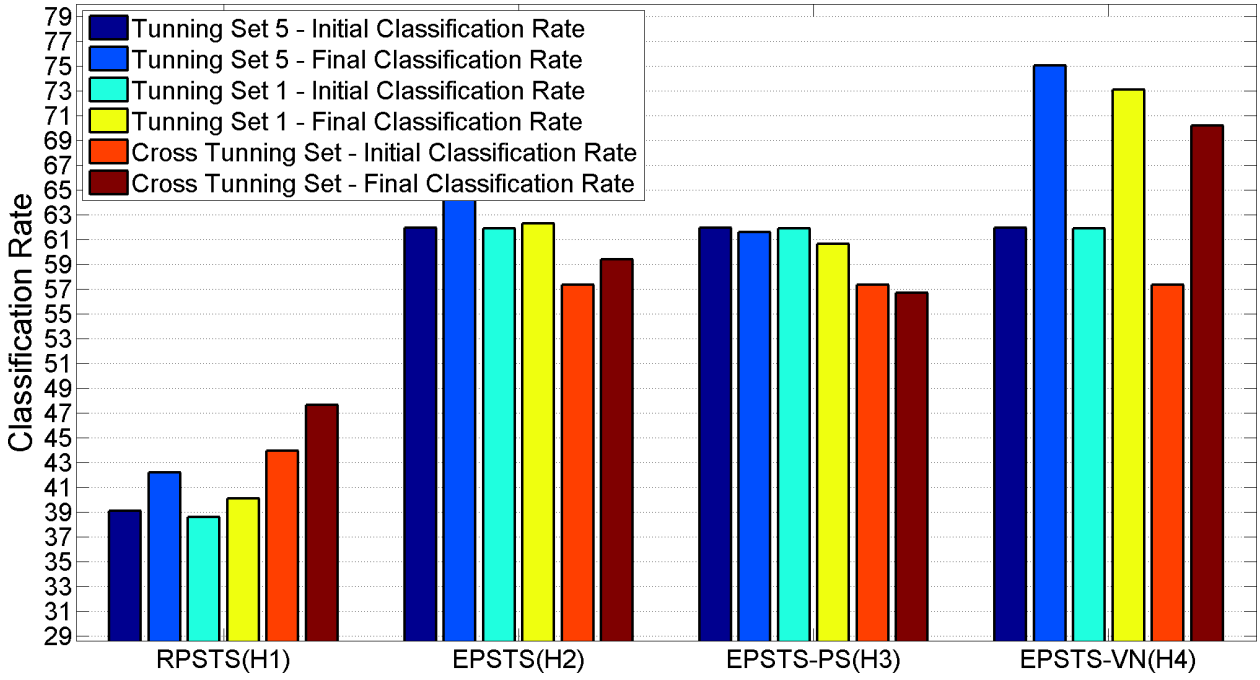
Table 5.1: Parameter settings for the four heuristics. The parameters listed in this table are: number of trials (n_{trials}), number of evaluations (n_{evals}), number of sampled pixels (n_{pels}), tabu tenure (tl), search space (S), neighborhood structure ($\mathcal{N}(\mathtt{s})$), Short-term Memory (STM), Long-term Memory (LTM), and Tuning set.

	RPSTS	**EPSTS**	**EPSTS-PS**	**EPSTS-VN**
n_{trials}	12	12	12	12
n_{evals}	100	100	100	300
n_{pels}	49	49	49	49
tl	5	5	5	5
S	Random pixels	Edge pixels	Edge pixels	Edge pixels
$\mathcal{N}(\mathtt{s})$	PADN	PADN	PADSN	PADN, FSN
STM	Yes	Yes	Yes	Yes
LTM	No	No	No	Yes
Tuning Set	D_1, D_5, D_{cross}	D_1, D_5, D_{cross}	D_1, D_5, D_{cross}	D_1, D_5, D_{cross}

5.2.1 Heuristic 1 - Random Pixel Sampled Tabu Search (RPSTS)

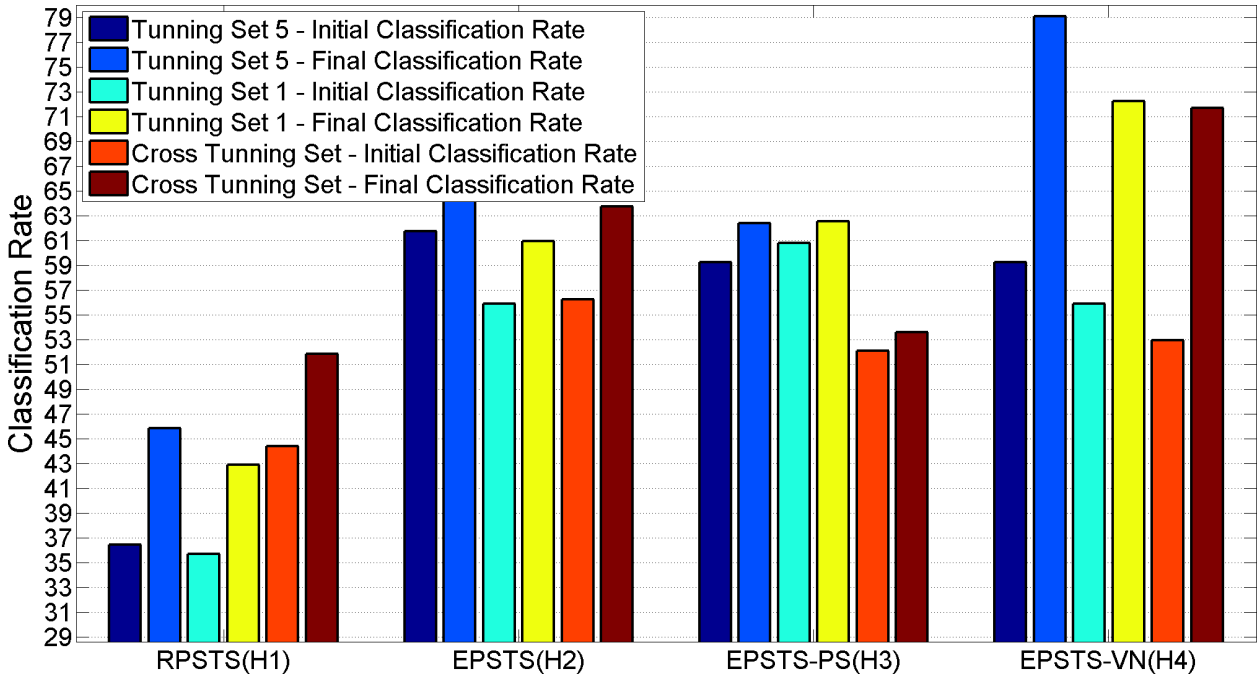
In this heuristic, the search space is pruned by random pixel sampling (as discussed in Section 4.5.4) and PADN is used as the neighborhood structure. As shown in Figure 5.1, this heuristic improves the average classification rate by about 3% when training is performed on D_5, D_{cross}, and about 1.5% when training is performed on D_1. For the best case (best of the 12 trials in terms of performance gain), RPSTS improves the classification rate by around 8% as shown in Figure 5.2. The motivation for using random pixel sampling is that with a good search strategy and sufficient evaluations, TS will eventually move towards the pixels with most discriminating information.

Additional experiments are performed to experimentally determine the number of evaluations in a TS. The number of evaluations per trial is gradually increased from 100 to 1000. But, increasing the number of evaluations did not provide any significant trend. Few experiments had a performance gain close to 4.5% and some others had performance gain of about 2%. This could possibly be due to overfitting. Another possible reason is that the search space is very rugged with several local minima, and the search procedure continuously runs into a local minimum. The number of evaluations is not increased above 1000 because for 1000 evaluations CR_{ftrain} is around 98%. This leaves very little room for learning, and any further increase in number of evaluations will probably result in more overfitting.



	D_5			D_1			D_{cross}		
	CR_{init}	CR_{final}	Gain	CR_{init}	CR_{final}	Gain	CR_{init}	CR_{final}	Gain
RPSTS	39.11	42.21	3.10	38.62	40.11	1.49	43.94	47.63	3.69
EPSTS	61.94	64.37	2.43	61.92	62.33	0.41	57.36	59.40	2.04
EPSTS-PS	61.94	61.60	**-0.34**	61.92	60.63	**-1.29**	57.36	56.71	0.65
EPSTS-VN	61.94	75.03	**13.09**	61.92	73.10	**11.18**	57.36	70.21	**12.85**

Figure 5.1: Average initial and final classification rates (averaged over 12 trials) for the four heuristics on the three different tuning sets.



	D_5			D_1			D_{cross}		
	CR_{init}	CR_{final}	Gain	CR_{init}	CR_{final}	Gain	CR_{init}	CR_{final}	Gain
RPSTS	36.44	45.83	9.39	35.69	42.88	7.19	44.38	51.86	7.48
EPSTS	61.77	69.50	7.73	55.88	60.97	5.09	56.25	63.77	7.52
EPSTS-PS	59.27	62.41	3.14	60.80	62.55	1.75	52.11	53.61	1.50
EPSTS-VN	59.27	79.11	**19.84**	55.88	72.25	**16.37**	52.97	71.72	**18.75**

Figure 5.2: Best initial and final classification rates (best of the 12 trials in terms of gain) for the four heuristics on the three different tuning sets.

5.2.2 Heuristic 2 - Edge Pixel Sampled Tabu Search (EPSTS)

The assumption made in this heuristic is that for the task of action classification edge pixels in a video provide more useful information than a random set of pixels. Therefore, the search space in this heuristic is pruned by edge pixels, and this can result in a less rugged search space compared to the previous heuristic. As expected the average initial classification rates are boosted by about 20% across all tuning sets as shown in Figure 5.1. But the average gain in performance is still only about 2% for D_5, D_{cross}, and almost negligible for D_1. For the best case the improvement is around 7% as shown in Figure 5.2.

Increasing the number of evaluations did not provide any significant improvement. A possible bottleneck for further improvement is the definition of the neighborhood structure. It is evident from Figure 4.3 that, with edge sampling, many spatially contiguous pixels will be present in the initial rearrangement vector. As a result, many of the neighbors constructed using the pixel add-drop move will be discarded.

5.2.3 Heuristic 3 - Edge Pixel Sampled Tabu Search with Pixel Swap (EPSTS-PS)

This heuristic avoids the problem of discarding neighbors by allowing pixel swap moves. Surprisingly, this heuristic resulted in a very poor performance. Although the performance gain for the best case is still in the positive, the average classification rate dropped by 0.34% for D_5 and 1.54% for D_1. However, the performance gain on the training set is about 20%. This clearly indicates a case of overfitting.

5.2.4 Heuristic 4 - Edge Pixel Sampled Tabu Search with Variable Neighborhood (EPSTS-VN)

In all the previous heuristics, the pixel is used as the basic element to implement local transformations within a frame, and all the local transformations are replicated across the frames. This could be a possible reason to stunt further improvement in these heuristics. In this heuristic, a new neighborhood structure FSN is used in conjunction with PADN. PADN is used as the neighborhood structure for the first 100 evaluations. For the next 200 evaluations, FSN is used as the neighborhood structure. Within these 200 evaluations only frame shift left and frame shift right moves are permitted for the first 100 evaluations, and only frame shift up and frame shift down moves are permitted for the next 100 evaluations.

As shown in Figures 5.1 and 5.2 EPSTS-VN outperforms all the previous heuristics by a large margin. The average gain in classification rate is around 12.5%, and the best case gain in classification rate is 19.84%. Examples of gestures rearranged using the rearrangement vector from the best trial are given in Figure 5.3. When the number of evaluations is doubled from 300 to 600, the average gain in classification rate jumps from 13.09% to 16%, and the best case gain jumps from 19.84% to 22.75%. Although, further increase in number

of evaluations did not provide any significant improvement. This could possibly be due to overfitting.

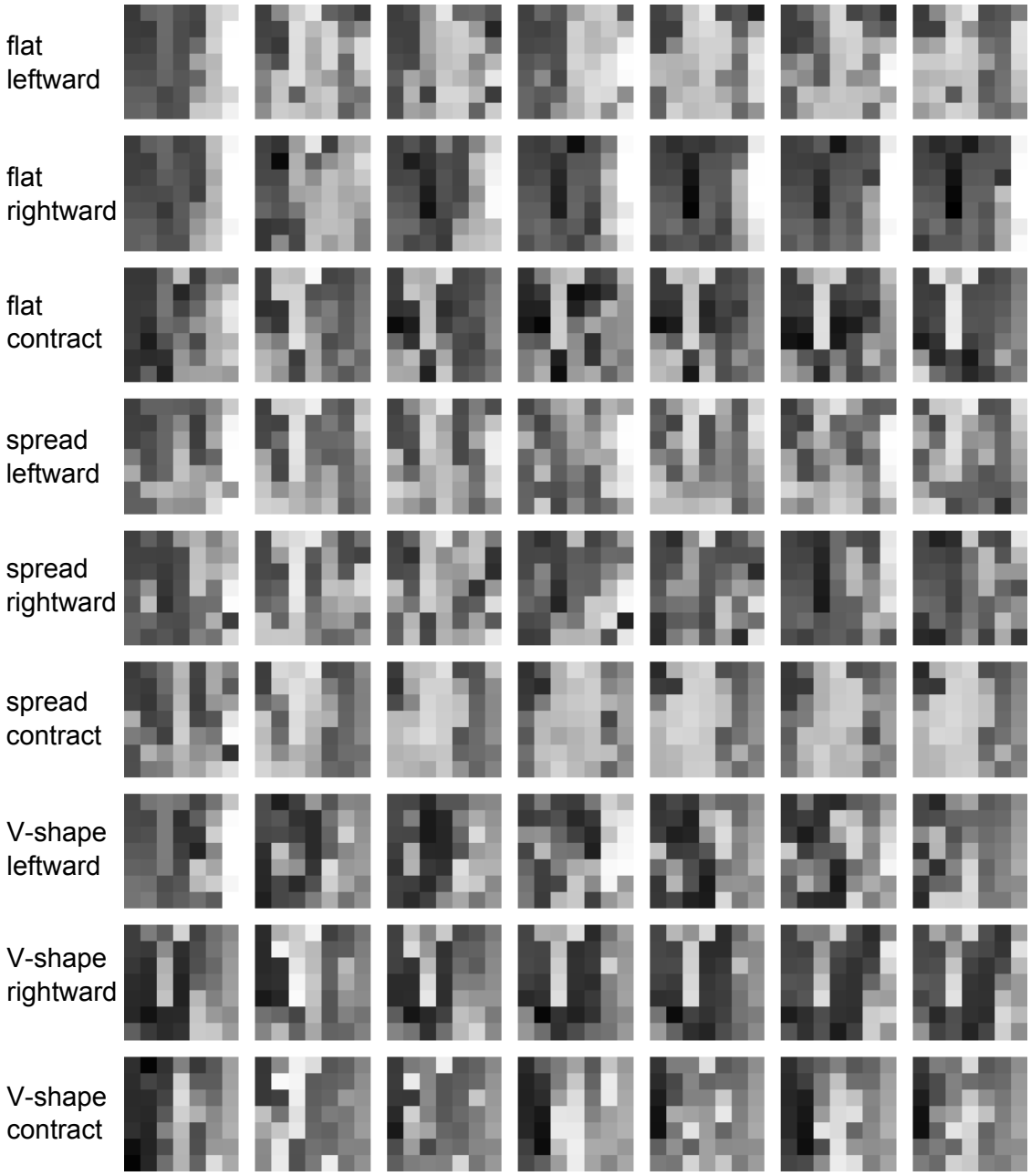


Figure 5.3: Example gestures from Set5 of the Cambridge-Gesture database after pixel rearrangement. Each of the nine rows correspond to a gesture. In each row seven frames out of the 32 frames are presented for each gesture. From top the gestures are: flat leftward, flat rightward, flat contract, spread leftward, spread rightward, spread contract, V-shape leftward, V-shape rightward, and V-shape contract.

In the first three heuristics, frequency count based diversification strategy is not implemented because the frequency count of the add, drop, or swap moves never exceeded five. In other words, the TS procedure did not repeat any particular move sufficient number of times to penalize the move. This is possibly because at any point there are many add, drop, and swap moves, so the probability of repeating a move is much less. But for FSN, at any point of the search process there are only 128 possible moves to choose from, so the chances of a particular move repeating is higher. Initial experiments show that some of the moves have a frequency count as large as 27.

Diversification is applied to the TS experiments where D_5 and D_{cross} are used as the tuning set. Values of 0.5 and 1 are used for $n_{penalty}$. With this penalty, the frequency count of all the moves is kept under 20. In some experiments the average performance gain increases by about 2%, but in others there is no significant improvement in classification rates.

5.2.5 Comparing the performance of the heuristics

In reviewing the experimental results shown in Figures 5.1 and 5.2, the following three conclusions can be made. Heuristics using edge pixels have higher initial classification rates. Using variable neighborhoods, in particular FSN, results in a large gain in classification rates, and using the pixel swap move to define neighborhood structure results in poor performance. Finally, the most interesting observation is, all the heuristics except for the EPSTS-PS heuristic, improve the classification accuracy irrespective of the choice of the tuning set. This suggests that the element rearrangement algorithm presented here benefits all the illumination cases in the Cambridge-Gesture data set.

Another important parameter to consider in the analysis of the heuristics is the number of trials in a TS experiment where there is a decrease in the classification rate (n_{fail}). This is shown in Figure 5.4. The number of failures is shown both for the test set and the query set. Only EPSTS-VN heuristic always results in a positive performance gain. This is important because regardless of where the initial solution is in the search space the EPSTS-VN heuristic always finds a solution better than the initial solution. On the other hand, almost half the

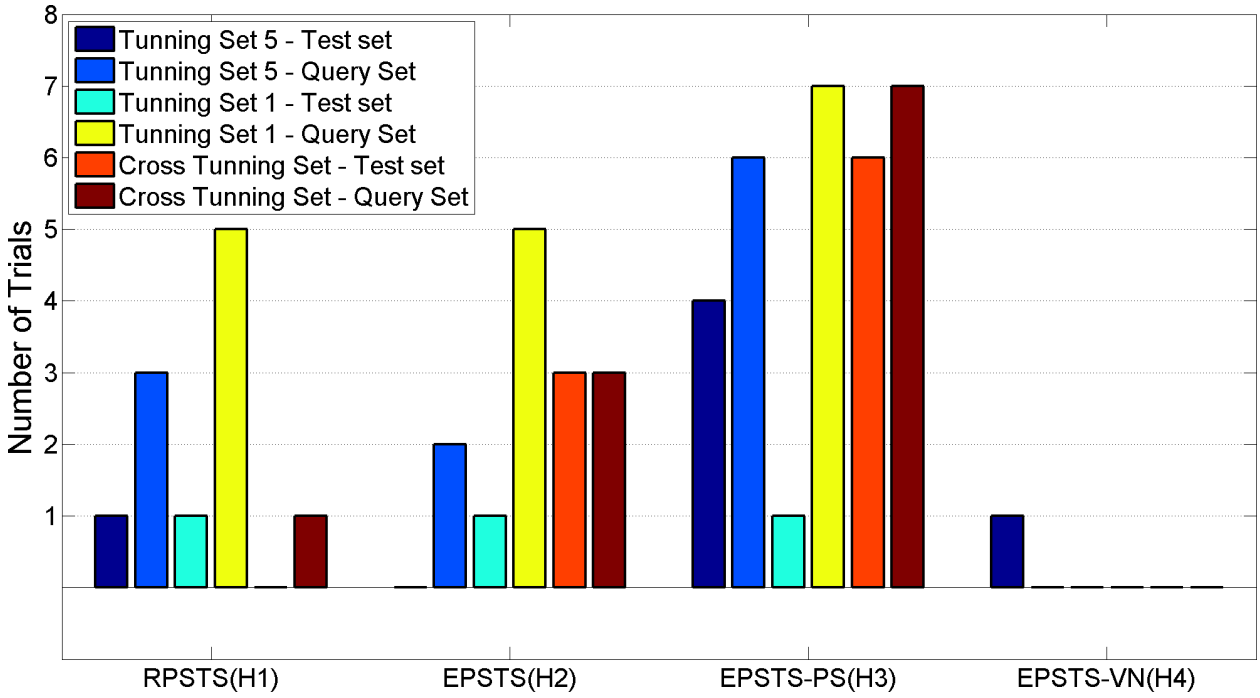


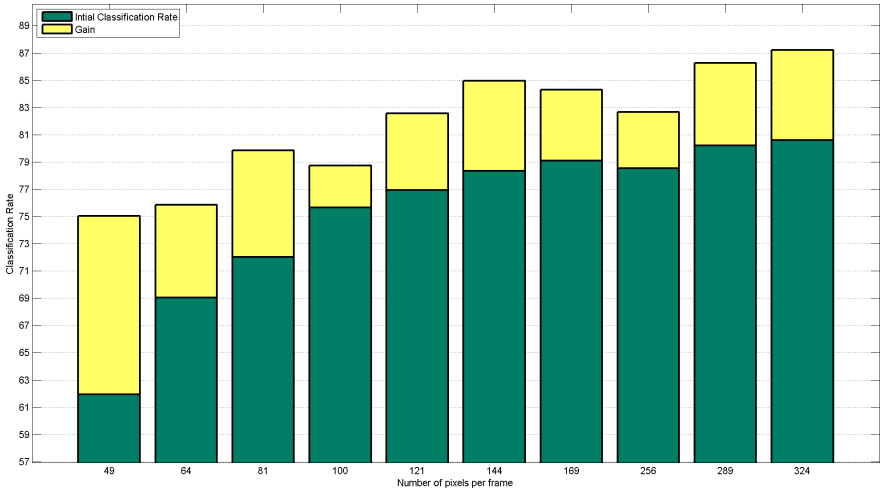
Figure 5.4: Number of trials (out of the 12 trials) in which the classification rate decreased as a result of TS. The results are shown for four the heuristics on the three different tuning sets.

trials using the EPSTS-PS heuristic result in a performance drop.

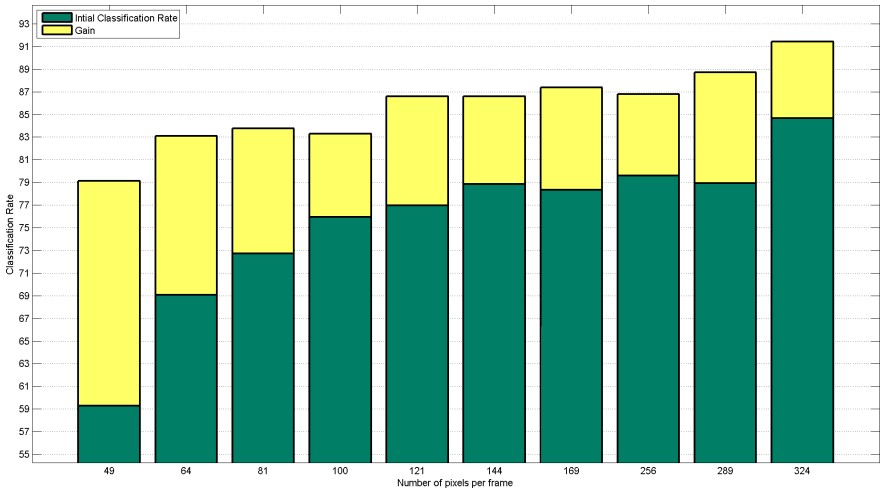
5.3 Parameter Tuning

From the discussion in the previous section it is clear that EPSTS-VN heuristic performs the best among the four heuristics. This heuristic will be used for all the remaining experiments in this work. As previously described in the TS tuning procedure in Section 5.1, the parameters of the EPSTS-VN heuristic are further tuned. The experimental determination of the number of sampled pixels, n_{pels}, and number of evaluations, n_{evals}, that maximize the classification rate are show in Figures 5.5 and 5.6, respectively.

Figure 5.5 shows the results of the EPSTS-VN heuristic for different values of n_{pels} (ranging from 49 to 324 pixels). The higher the number of pixels used greater is the classification rate. The best performance is achieved using the greatest number of pixels, in this case for $n_{pels} = 324$. We use this as the number of sampled pixels for all the remaining experiments in this work. The best case classification rate for $n_{pels} = 324$ is better than the classification



(a)



(b)

Figure 5.5: Classification rates for the EPSTS-VN heuristic with different values for n_{pels}. D_5 is used as the tuning set, $n_{trials} = 12$, $n_{evals} = 300$, $n_{penalty} = 1$, and $tl = 5$: (a) Average classification rate (averaged over 12 trials) and (b) Best classification rate (best of the 12 trials in terms of gain).

rate achieved by the original ACOPM method using all pixels by 3.4%. From the Figure 5.5 it can be seen that the initial classification rate increases with the increase in the number of sampled pixels. This is because with more pixels there is more information and this results in better initial classification rates. However, the gain in performance decreases with the increase in n_{pels}. Figure 5.5 shows that gain in performance for the case 324 pixels per frame is about 8% which is half of the gain in performance for the case with 49 pixels per frame.

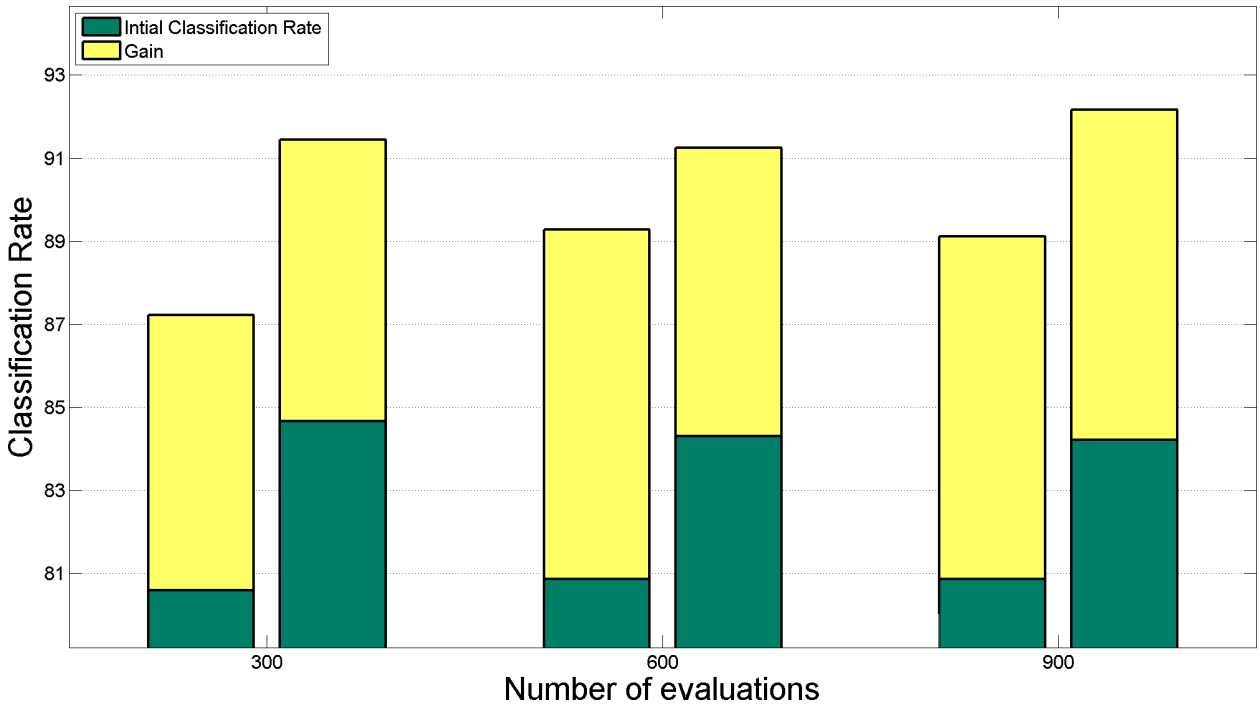


Figure 5.6: Classification rates for the EPSTS-VN heuristic with different values for n_{evals}. The first bar in each group corresponds to the average classification rate (averaged over 12 trials), and the second bar in each group corresponds to the best classification rate (best of the 12 trials in terms of gain). D_5 is used as the tuning set, $n_{trials} = 12$, $n_{pels} = 324$, $n_{penalty} = 1$, and $tl = 5$.

Figure 5.6 shows an experimental determination of the number of evaluations. The average classification rate for 600 and 900 evaluations is about 1.1% more than the original ACOPM method. The best case classification rate for 300, 600, and 900 evaluations improves the ACOPM method by 3.4%, 3.25%, and 4.15%, respectively. It took a little over ten days to complete an experiment with 900 evaluations on an Intel Xeon dual six-core server processor with 12 parallel workers. To perform experiments with larger number of evaluations, it would be important to further optimize the code. Examples of gestures rearranged using the rearrangement vector from the best trial for the EPSTS-VN heuristic with 900 evaluations are given in Figure 5.7.

5.4 Tensor Subspace Analysis

As discussed in Chapter 3, the ACOPM method represents videos as third order tensors and computes three orthogonal matrices $\mathbf{V}_{hm}^{(1)}$, $\mathbf{V}_{vm}^{(2)}$, and $\mathbf{V}_{app}^{(3)}$ using the modified HOSVD

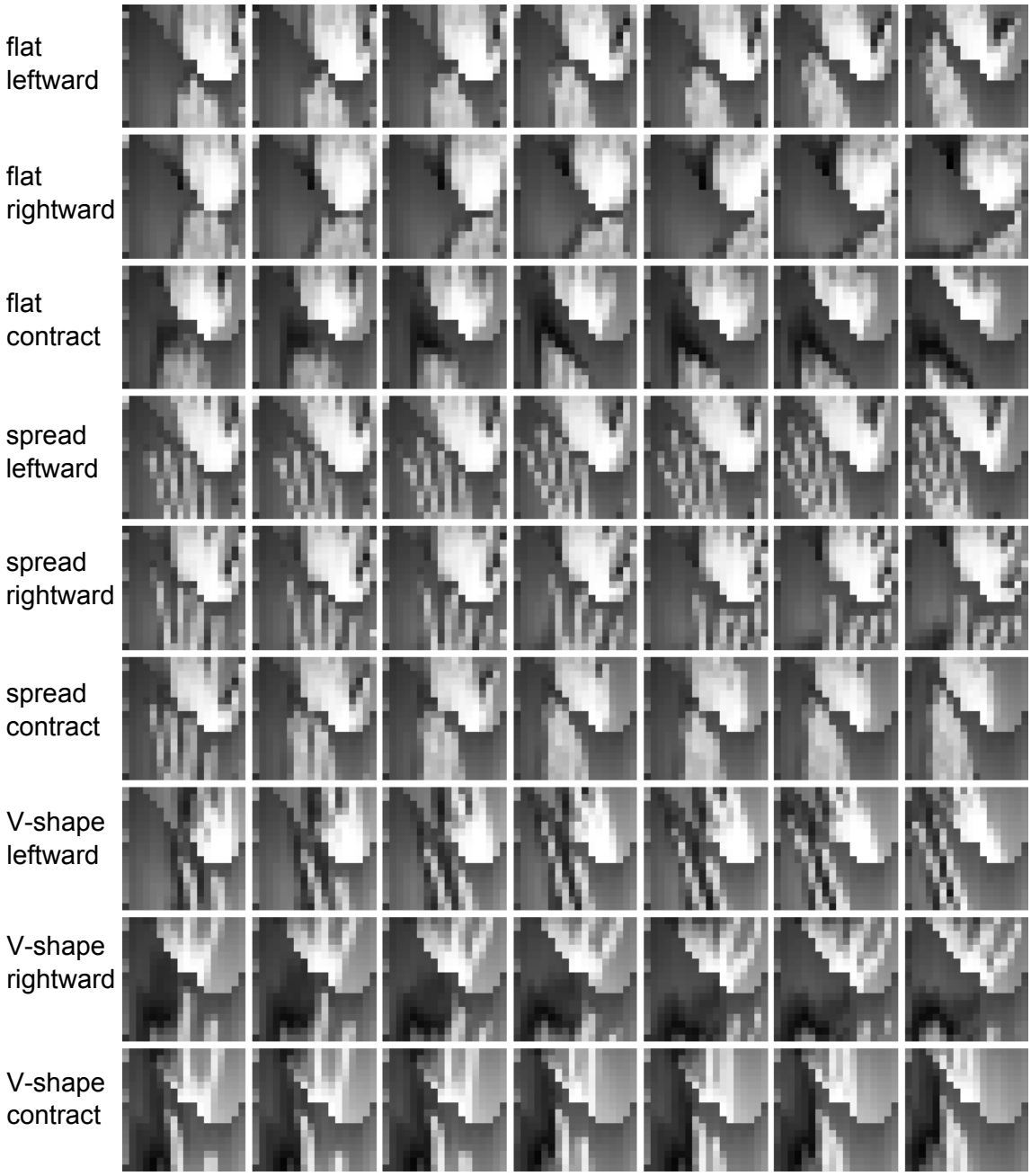


Figure 5.7: Example gestures from Set5 of the Cambridge-Gesture database after rearrangement. Each of the nine rows correspond to a gesture. In each row seven frames out of the 32 frames are presented for each gesture. From top the gestures are: flat leftward, flat rightward, flat contract, spread leftward, spread rightward, spread contract, V-shape leftward, V-shape rightward, and V-shape contract.

given in Equation (3.8). Each of the three matrices are represented as points on three different Grassmann manifolds $\mathcal{M}_{hm}^{(1)}$, $\mathcal{M}_{vm}^{(2)}$, and $\mathcal{M}_{app}^{(3)}$. The Grassmann manifolds are combined using Equation (3.9) to map the videos to a single point on the Grassmann Product Manifold (GPM). Classification is performed using the geodesic distance on the GPM coupled with a simple nearest neighbor classifier as shown in Equation (3.11). This section provides some insight into the relationship between the element rearrangement operation and the subspace spanned by the three orthogonal matrices.

In all the previous experiments, classification is performed on the GPM, that is the subspaces spanned by all the three mode-k flattened matrices is used for the task of action classification. Classification performed independently on each of the three Grassmann manifolds is considered here to understand how element rearrangement affects each of the three subspaces spanned by the three mode-k flattened matrices. The results for classification performed on the GPM and each of the three Grassmann manifolds are shown in Figures 5.8 and 5.9.

For all the experimental results shown in Figures 5.8 and 5.9, the parameters are initialized as shown in Table 5.1 except that only D_5 is used as the tuning set. Results in Figure 5.8 show the tensor subspace analysis for all the four heuristics, and results in Figure 5.9 show the tensor subspace analysis for the EPSTS-VN heuristic with different values for number of sampled pixels. The results and observations are summarized below:

1. For the ACOPM method and the RPSTS heuristic,the appearance subspace (the subspace spanned by the columns of $\mathbf{V}_{app}^{(3)}$) provides the most discriminating information.

2. In TS experiments using the edge pixel based heuristics, classification on $\mathcal{M}_{hm}^{(1)}$ and $\mathcal{M}_{vm}^{(2)}$ performs better than classification on $\mathcal{M}_{app}^{(3)}$. This is because the edge pixels are better suited to capture the variations in the horizontal and vertical motions rather than capture variations in appearance.

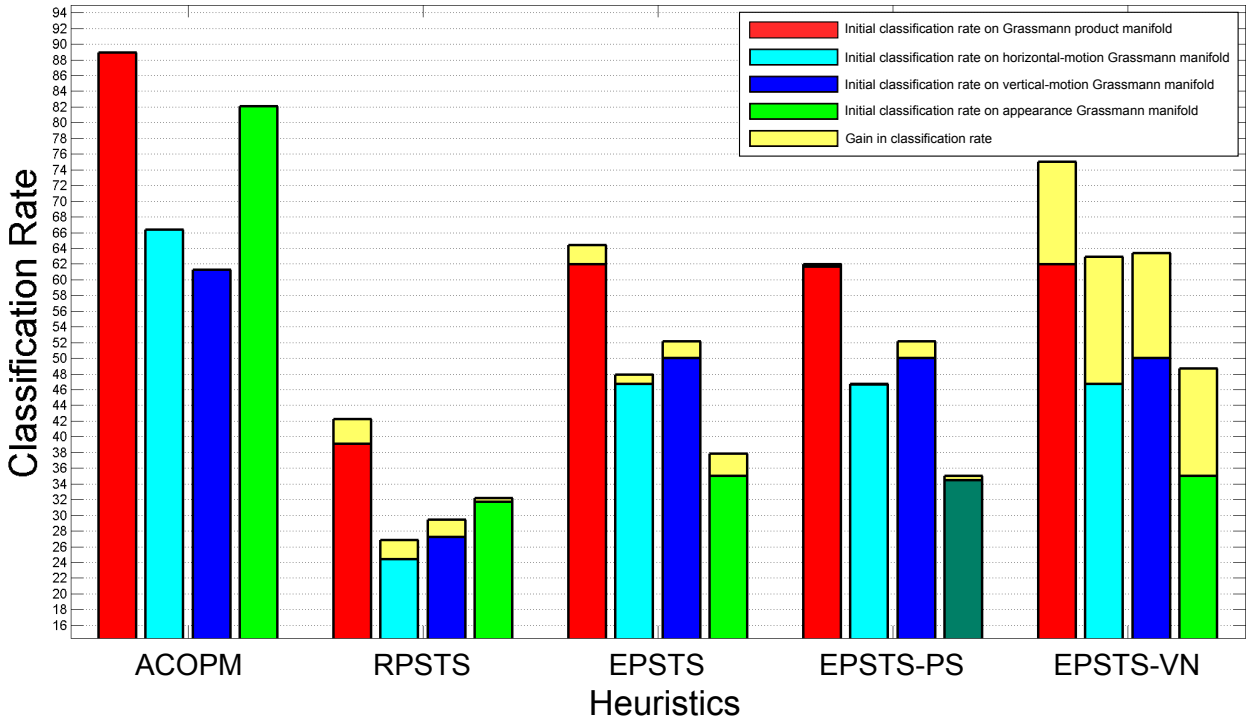
3. In the case of the EPSTS-PS heuristic, there is a negative gain in the average classification rate primarily due to the poor performance in the appearance Grassmann manifold

$\mathcal{M}_{app}^{(3)}$. This is because swapping pixels is synonymous to swapping columns in the mode-3 flattened matrix, and swapping columns does not change the subspace spanned by it. Therefore, the TS with the swap move will possibly result in overfitting and perform poorly on the query set.

4. The EPSTS-VN heuristic is the only heuristic among the four heuristics to perform well on all the three Grassmann manifolds.

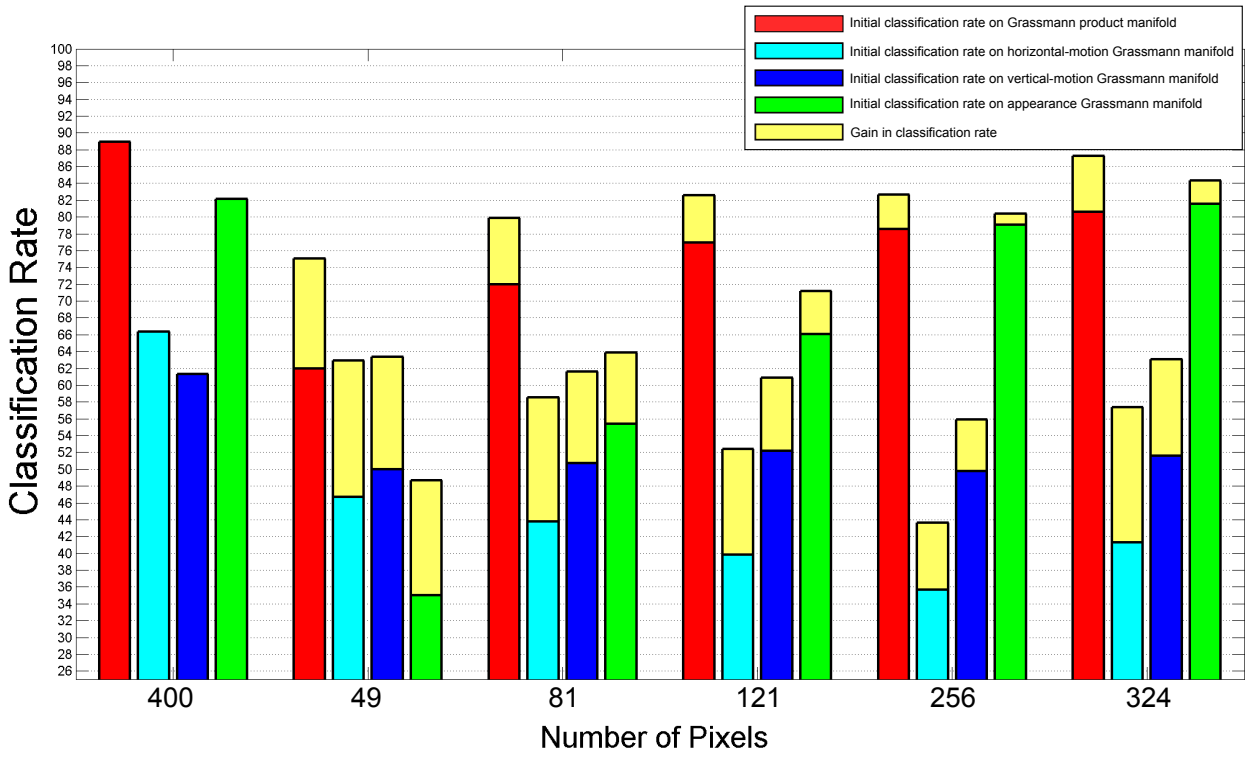
5. For the EPSTS-VN heuristic, as number sampled pixels per frame increases (with the exception of $n_{pels} = 324$), the initial classification rates and gains associated with $\mathcal{M}_{hm}^{(1)}$ and $\mathcal{M}_{vm}^{(2)}$ decreases. This is possibly because with the increase in the number of sampled pixels per frame, the weak (less useful) edge pixels are also included in the frame. This can dampen the ability of edge pixels to capture the variations in the horizontal and vertical motions.

6. However, the classification rate of the EPSTS-VN heuristic on GPM increases with the increase in number of sampled pixels per frame. This is because the initial classification rate in the appearance Grassmann manifold increases with the increase in the number of sampled pixels per frame.



	GPM		$\mathcal{M}_{hm}^{(1)}$		$\mathcal{M}_{vm}^{(2)}$		$\mathcal{M}_{app}^{(3)}$	
	CR_{init}	Gain	CR_{init}	Gain	CR_{init}	Gain	CR_{init}	Gain
ACOPM	88.91	0	66.36	0	61.27	0	82.11	0
RPSTS	39.11	3.09	24.37	2.45	27.22	2.16	31.70	0.50
EPSTS	61.94	2.42	46.70	1.19	49.97	2.14	35.03	2.80
EPSTS-PS	61.94	**-0.33**	46.70	**-0.03**	49.97	2.14	35.03	**-0.58**
EPSTS-VN	61.94	**13.09**	46.70	**16.18**	49.97	**13.36**	35.03	**13.60**

Figure 5.8: Tensor subspace analysis of the ACOPM method and the four heuristics. First bar in each group gives the classification rate when the classification is performed on the GPM. The second, third, and fourth bars of each group give the classification rates achieved when classification is performed on $\mathcal{M}_{hm}^{(1)}$, $\mathcal{M}_{vm}^{(2)}$, and $\mathcal{M}_{app}^{(3)}$, respectively. All the classification rates are averaged over 12 trials.



	GPM		$\mathcal{M}_{hm}^{(1)}$		$\mathcal{M}_{vm}^{(2)}$		$\mathcal{M}_{app}^{(3)}$	
	CR_{init}	Gain	CR_{init}	Gain	CR_{init}	Gain	CR_{init}	Gain
400	88.91	0	66.36	0	61.27	0	82.11	0
49	61.94	13.09	46.70	16.18	49.97	13.36	35.03	13.60
81	72.01	7.84	43.75	14.80	50.73	10.88	55.41	8.47
121	76.93	5.62	39.84	12.55	52.18	8.67	66.09	5.11
256	78.54	4.10	35.69	7.92	49.79	6.13	79.03	1.33
324	80.60	6.62	41.31	16.03	51.61	11.43	81.52	2.83

Figure 5.9: Tensor subspace analysis of the EPSTS-VN heuristic for different values of n_{pels}. First bar in each group gives the classification rate when the classification is performed on the GPM. The second, third, and fourth bars of each group give the classification rates achieved when classification is performed on $\mathcal{M}_{hm}^{(1)}$, $\mathcal{M}_{vm}^{(2)}$, and $\mathcal{M}_{app}^{(3)}$, respectively. All the classification rates are averaged over 12 trials.

Chapter 6

Conclusion

In this thesis, new unfoldings of a tensor are developed by rearranging elements in the original tensor and then applying the standard mode-k unfoldings. We study the problem of rearranging elements in a video to achieve better classification accuracy. Tabu search is used in this work to search for the best rearrangement of pixels in a video tensor. The usefulness of the element rearrangement operation is demonstrated experimentally by employing it as a preprocessing step to the action classification on product manifolds algorithm. Experimental results are reported on different tabu search heuristics. By properly choosing a search space and a neighborhood structure for tabu search, the classification accuracy of the action classification on product manifolds algorithm is improved from 88% to 92.15%.

A summary of the experimental results for the best heuristic, EPSTS-VN, is given below:

1. Regardless of where the initial solution is in the search space, the tabu search metaheuristic always finds a solution better than the initial solution.

2. The element rearrangement algorithm is invariant to the illumination variations in the Cambridge-Gesture data set.

3. The gain in performance due to element rearrangement decreases as the number of pixels per frame increases.

4. The algorithm presented in this work answers an interesting question: "Which 10% of the pixels in a video are useful for the task of action classification?" This question can be of interest in computational-resource-constrained environments. The algorithm can identify 10% of pixels in a video that achieves a classification accuracy of 79.11%, and 80% of pixels in a video that achieves a classification accuracy of 92.15%, as opposed to the

original action classification on product manifolds algorithm that achieves a classification accuracy of 88% by using all the pixels. The most interesting observation is that by simply dropping 20% of pixels from a video the classification accuracy increases by 4.15%.

Even though 90% of the pixels are dropped the performance drops only by 10%. One possible reason for this behavior could be that the dropped pixels are unrelated to the class labels sought by the classifier, then removal of such pixels is of benefit to any approach trying to learn features needed to perform classification. A second possible reason could be due to the curse of dimensionality. By dropping pixels, the dimensionality of the space in which the learning must take place is reduced. This is beneficial simply because of the fundamental relationship between the number of training samples available for the machine learning algorithm and the degrees of freedom in the space in which the learning algorithm must construct a decision criteria. Decrease the number of dimensions, and in the most general sense, one needs fewer training samples. In all cases, the tabu search metaheuristic always finds a solution better than the initial solution. This is because when different pixels are chosen different variations are captured in the three unfoldings. Therefore, some set of pixels tend to hold more discriminating information than others.

In summary, the element rearrangement algorithm presented in this work suggests that the gain in performance is at best modest. In most cases, choosing a subset of pixels drops rather than improves the performance relative to the original ACOPM algorithm. Although, in compute-resource-constrained environments it may be beneficial to use the proposed method to choose a subset of pixels that provides more discriminating information.

6.1 Future Work

The results reported in this thesis encourage further study of the role of element rearrangement as preprocessing step in tensor-based classification algorithms. Given that a metaheuristic technique, tabu search, is used in this work there exists several untested heuristics, looking into some of those heuristics is a potential direction of future work. For example, rearrangement of pixels across different frames within a video is not considered in this work.

Another example is instead of replacing a pixel by only the pixel above it or below it, all the 8 neighboring pixels can be considered for replacing the current pixel.

Another possible extension of this will work be looking at other classes of data objects that can benefit for element rearrangement. One possibility is face recognition by image-set matching. Another direction of future work is to explore other frameworks, such as genetic algorithms, simulator annealing, and numerical methods for solving the element rearrangement problem.

References

[1] Emile Aarts and Jan K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[2] E. Acar and B. Yener. Unsupervised multiway data analysis: A literature survey. *Knowledge and Data Engineering, IEEE Transactions on*, 21(1):6 –20, jan. 2009.

[3] Brett W. Bader and Tamara G. Kolda. Matlab tensor toolbox version 2.4. `http://csmr.ca.sandia.gov/~tgkolda/TensorToolbox/`, March 2010.

[4] Ronen Basri and David W. Jacobs. Lambertian reflectance and linear subspaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(2):218–233, February 2003.

[5] Evgeni Begelfor and Michael Werman. Affine invariance revisited. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2087–2094, Washington, DC, USA, 2006. IEEE Computer Society.

[6] Peter N. Belhumeur and David J. Kriegman. What is the set of images of an object under all possible illumination conditions? *Int. J. Comput. Vision*, 28(3):245–260, July 1998.

[7] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, pages 217–235, London, UK, UK, 1999. Springer-Verlag.

[8] M.A.-A. Bhuiyan, M.E. Islam, N. Begum, M. Hasanuzzaman, Chang Hong Liu, and H. Ueno. Vision based gesture recognition for human-robot symbiosis. In *Computer and information technology, 2007. iccit 2007. 10th international conference on*, pages 1 –6, dec. 2007.

[9] Aske Bjorck and Gene H. Golub. Numerical methods for computing angles between linear subspaces. *Mathematics of Computation*, 27(123):pp. 579–594, 1973.

[10] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003.

[11] Kenneth D. Boese, Andrew B. Kahng, and Sudhakar Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Oper. Res. Lett.*, 16(2):101–113, September 1994.

[12] L. Cammoun, C.A. Castao-Moraga, E. Muoz-Moreno, D. Sosa-Cabrera, B. Acar, M.A. Rodriguez-Florido, A. Brun, H. Knutsson, and J. P. Thiran. A review of tensors and tensor signal processing. In Santiago Aja-Fernndez, Rodrigo Luis Garca, Dacheng Tao,

and Xuelong Li, editors, *Tensors in Image Processing and Computer Vision*, Advances in Pattern Recognition, pages 1–32. Springer London, 2009.

[13] H.E. Cetingul and R. Vidal. Intrinsic mean shift for clustering on stiefel and grassmann manifolds. In *Computer Vision and Pattern Recognition. IEEE Conference on*, pages 1896–1902, June 2009.

[14] John H. Conway, Ronald H. Hardin, and Neil J. A. Sloane. Packing lines, planes, etc.: Packings in grassmannian spaces. *Experimantal Mathematics*, 5(2):139–159, 1996.

[15] L. De Lathauwer. A survey of tensor methods. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 2773 –2776, may 2009.

[16] Selva di Fasano. From matrix to tensor: The transition to computational multilinear algebra @ONLINE. `http://issnla2010.ba.cnr.it/Course_Van_Loan.htm`, June 2010.

[17] Alan Edelman, Tomás A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM J. Matrix Anal. Appl.*, 20(2):303–353, April 1999.

[18] Bernd Freisleben and Peter Merz. New genetic local search operators for the traveling salesman problem. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, PPSN IV, pages 890–899, London, UK, UK, 1996. Springer-Verlag.

[19] Kazuhiro Fukui and Osamu Yamaguchi. Face recognition using multi-viewpoint patterns for robot vision. In Paolo Dario and Raja Chatila, editors, *Robotics Research, The Eleventh International Symposium, ISRR, October 19-22, 2003, Siena, Italy*, volume 15 of *Springer Tracts in Advanced Robotics*, pages 192–201. Springer, 2003.

[20] Michel Gendreau. Recent advances in tabu search. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, number 369-377. Kluwer Academic Publishers, 2002.

[21] Michel Gendreau. An introduction to tabu search. In Fred Glover and GaryA. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 37–54. Springer US, 2003.

[22] F. Glover. Tabu search – Part II. *ORSA Journal on Computing*, 2:4–32, 1990.

[23] Fred Glover. Tabu search – Part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

[24] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[25] Gene H. Golub and Hongyuan Z ha. The canonical correlations of matrix pairs and their numerical computation. Technical report, Stanford, CA, USA, 1992.

[26] Peter Gruber and Fabian J. Theis. Grassmann clustering. In *Proceedings of the European Signal Processing Conference*, 2006.

[27] Jihun Hamm. *Subspace-based learning with Grassmann kernels*. PhD thesis, University of Pennsylvania, 2008.

[28] Jihun Hamm and Daniel D. Lee. Grassmann discriminant analysis: a unifying view on subspace-based learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 376–383, New York, NY, USA, 2008. ACM.

[29] Md. Hasanuzzaman, T. Zhang, V. Ampornaramveth, H. Gotoda, Y. Shirai, and H. Ueno. Adaptive visual gesture recognition for human-robot interaction using a knowledge-based software platform. *Robot. Auton. Syst.*, 55(8):643–657, August 2007.

[30] Harold Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):pp. 321–377, 1936.

[31] David S. Johnson, Christos H. Papadimtriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, August 1988.

[32] Daniel Kersten. Predictability and redundancy of natural images. *J. Opt. Soc. Am. A*, 4(12):2395–2400, Dec 1987.

[33] Tae-Kyun Kim and Roberto Cipolla. Canonical correlation analysis of video volume tensors for action categorization and detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(8):1415–1428, August 2009.

[34] Tae-Kyun Kim, J. Kittler, and R. Cipolla. Discriminative learning and recognition of image set classes using canonical correlations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1005 –1018, june 2007.

[35] Tae-Kyun Kim, Shu-Fai Wong, and R. Cipolla. Tensor canonical correlation analysis for action classification. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1 –8, june 2007.

[36] M. Kirby and L. Sirovich. Application of the karhunen-loeve procedure for the characterization of human faces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(1):103 –108, January 1990.

[37] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51:455–500, August 2009.

[38] Volker Kruger, Danica Kragic, Ales Ude, and Christopher Geib. The meaning of action: a review on action recognition and mapping. *Advanced Robotics*, 21(13):1473–1501, 2007.

[39] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1 –8, june 2008.

[40] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21:1253–1278, March 2000.

[41] John M. Lee. *Introduction to Smooth Manifolds*, volume 218 of *Graduate Texts in Mathematics*. Springer, first edition, 2005.

[42] Kuang-Chih Lee, Jeffrey Ho, and David J. Kriegman. Acquiring linear subspaces for face recognition under variable lighting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(5):684–698, May 2005.

[43] J Li, R Janardan, and Q Li. Two-dimensional linear discriminant analysis. *Advances in Neural Information Processing Systems*, 17:1569–1576, 2004.

[44] Xi Li, Weiming Hu, Zhongfei Zhang, Xiaoqin Zhang, and Guan Luo. Robust visual tracking based on incremental tensor subspace learning. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1 –8, oct. 2007.

[45] X. Liu, A. Srivastava, and K. Gallivan. Optimal linear representations of images for object recognition. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–229–I–234 vol.1, June 2003.

[46] Yui Man Lui. *Geometric methods on special manifolds for visual recognition*. PhD thesis, Colorado State University. Libraries, 2010.

[47] Yui Man Lui. Advances in matrix manifolds for computer vision. *Image and Vision Computing*, 30(67):380 – 388, 2012.

[48] Yui Man Lui, J.R. Beveridge, and M. Kirby. Action classification on product manifolds. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 833 –839, june 2010.

[49] Yi Ma, Jana Koseck, and Shankar Sastry. Optimal motion from image sequences: A riemannian viewpoint. In *In Proceeding of the Conference on Mathematical Theory of Networks and Systems*, 1998.

[50] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Comput. Vis. Image Underst.*, 104(2):90–126, November 2006.

[51] Hiroshi Murase and Shree K. Nayar. Visual learning and recognition of 3-d objects from appearance. *Int. J. Comput. Vision*, 14:5–24, January 1995.

[52] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[53] JuanCarlos Niebles, Hongcheng Wang, and Li Fei-Fei. Unsupervised learning of human action categories using spatial-temporal words. *International Journal of Computer Vision*, 79:299–318, 2008.

[54] Christos H. Papadimitriou and Kenneth 1939 Steiglitz. *Combinatorial optimization algorithms and complexity*. Dover Publications, 1998,1982.

[55] Ronald Poppe. A survey on vision-based human action recognition. *Image and Vision Computing*, 28(6):976 – 990, 2010.

[56] Ravi Ramamoorthi. Analytic pca construction for theoretical analysis of lighting variability in images of a lambertian object. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(10):1322–1333, October 2002.

[57] Q. Rentmeesters, P-A Absil, P. Van Dooren, K. Gallivan, and A. Srivastava. An efficient particle filtering technique on the grassmann manifold. In *Acoustics Speech and Signal Processing (ICASSP), IEEE International Conference on*, pages 3838–3841, March 2010.

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

[59] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: a local svm approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32 – 36 Vol.3, aug. 2004.

[60] Terence Sim, Simon Baker, and Maan Bsat. The cmu pose, illumination, and expression database. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(12):1615–1618, December 2003.

[61] E. P. Simoncelli and B. A. Olshausen. Natural image statistics and neural representation. *Annual Review of Neuroscience*, 24:1193–1216, 2001.

[62] T Stützle. *Local Search Algorithms for Combinatorial Problems*. PhD thesis, Darmstadt University of Technology, 1998.

[63] J. B. Tenenbaum, V. Silva, and J. C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[64] Craig A Tovey. Hill climbing with multiple local optima. *SIAM Journal on Algebraic Discrete Methods*, 6(3):384–393, 1985.

[65] LedyardR Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

[66] Pavan Turaga, Ashok Veeraraghavan, Anuj Srivastava, and Rama Chellappa. Statistical computations on grassmann and stiefel manifolds for image and video-based recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(11):2273–2286, November 2011.

[67] P.K. Turaga, A. Veeraraghavan, and R. Chellappa. From videos to verbs: Mining videos for activities using a cascade of dynamical systems. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.

[68] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3:71–86, January 1991.

[69] M. A. O. Vasilescu. Human motion signatures: analysis, synthesis, recognition. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 456 – 460 vol.3, 2002.

[70] M. A. O. Vasilescu and Demetri Terzopoulos. Multilinear Analysis of Image Ensembles: TensorFaces. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*, pages 447–460, London, UK, 2002. Springer-Verlag.

[71] Tiesheng Wang, A.G. Backhouse, and I.Y.H. Gu. Online subspace learning on grassmann manifold for moving object tracking in video. In *Acoustics, Speech and Signal Processing. IEEE International Conference on*, pages 969–972, 31 2008.

[72] Shu-Fai Wong and R. Cipolla. Extracting spatiotemporal interest points using global information. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1 –8, oct. 2007.

[73] Dong Xu, Shuicheng Yan, Lei Zhang, Hong-Jiang Zhang, Zhengkai Liu, and Heung-Yeung Shum. Concurrent subspaces analysis. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 203–208, Washington, DC, USA, 2005. IEEE Computer Society.

[74] O. Yamaguchi, K.. Fukui, and K.-i. Maeda. Face recognition using temporal image sequence. In *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, pages 318 –323, 1998.

[75] Shuicheng Yan, Dong Xu, S. Lin, T.S. Huang, and Shih-Fu Chang. Element rearrangement for tensor-based subspace learning. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 2007.

[76] Shuicheng Yan, Dong Xu, Qiang Yang, Lei Zhang, Xiaoou Tang, and Hong-Jiang Zhang. Discriminant analysis with tensor representation. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 526–532, Washington, DC, USA, 2005. IEEE Computer Society.

[77] Jian Yang, David Zhang, Alejandro F. Frangi, and Jing-yu Yang. Two-dimensional pca: A new approach to appearance-based face representation and recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(1):131–137, January 2004.

[78] Jieping Ye. Generalized low rank approximations of matrices. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 112–, New York, NY, USA, 2004. ACM.

[79] Guido Zuccon, Leif A. Azzopardi, and C. J. Rijsbergen. Semantic spaces: Measuring the distance between different subspaces. In *Proceedings of the 3rd International Symposium on Quantum Interaction*, QI '09, pages 225–236, Berlin, Heidelberg, 2009. Springer-Verlag.

Appendix A

Symbol Glossary

Table A.1: Symbol Glossary

Notations	Descriptions
$\mathcal{A}$	Tensor (upper-case letters in calligraphic font)
$\mathbf{A}$	Matrix (upper-case letters in bold font)
a	Vector (lower-case letters)
$\mathcal{A}_{(k)}$	Mode-k unfolded matrix
$\mathbf{U}^{(k)}$	Mode-k singular (orthonormal) matrix
$\times_k$	Mode-k product
$\|\|x\|\|$	Euclidean norm of x
$\mathcal{V}_{n,p}$	Stiefel Manifold (a set of p orthonormal vectors in $\mathbb{R}^n$)
$\mathcal{G}_{n,p}$	Grassmann manifold (a set of p-dimensional linear subspaces in $\mathbb{R}^n$)
n_{trials}	Number of trials in a tabu search experiment
n_{evals}	Number of evaluations per trial in a tabu search experiment
n_{pels}	Number of pixels sampled per frame
$n_{penalty}$	Proportionality constant used in the diversification strategy of the tabu search
tl	Tabu tenure of the short-term memory used in tabu search