ABSTRACT

Title of dissertation: GEOMETRIC REPRESENTATIONS AND DEEP GAUSSIAN CONDITIONAL RANDOM FIELD NETWORKS FOR COMPUTER VISION

Raviteja Vemulapalli
Doctor of Philosophy, 2016

Dissertation directed by: Professor Rama Chellappa
Department of Electrical and Computer Engineering

Representation and context modeling are two important factors that are critical in the design of computer vision algorithms. For example, in applications such as skeleton-based human action recognition, representations that capture the 3D skeletal geometry are crucial for achieving good action recognition accuracy. However, most of the existing approaches focus mainly on the temporal modeling and classification steps of the action recognition pipeline instead of representations. Similarly, in applications such as image enhancement and semantic image segmentation, modeling the spatial context is important for achieving good performance. However, the standard deep network architectures used for these applications do not explicitly model the spatial context. In this dissertation, we focus on the representation and context modeling issues for some computer vision problems and make novel contributions by proposing new 3D geometry-based representations for recognizing human actions from skeletal sequences, and introducing Gaussian conditional random field model-based deep network architectures that explicitly model the spatial context by considering the interactions among the output variables. In addition, we also propose a kernel learning-based framework for the classification of manifold features such as linear subspaces and covariance matrices which are widely used for image set-based recognition tasks.

This dissertation has been divided into five parts. In the first part, we introduce various 3D geometry-based representations for the problem of skeleton-based human action recognition. The proposed representations, referred to as R3DG features, capture the relative 3D geometry between various body parts using 3D rigid body transformations. We model human actions as curves in these R3DG feature spaces, and perform action recognition using a combination of dynamic time warping, Fourier temporal pyramid representation and support vector machines. Experiments on several action recognition datasets show that the proposed representations perform better than many existing skeletal representations.

In the second part, we represent 3D skeletons using only the relative 3D rotations between various body parts instead of full 3D rigid body transformations. This skeletal representation is scale-invariant and belongs to a Lie group based on the special orthogonal group. We model human actions as curves in this Lie group and map these curves to the corresponding Lie algebra by combining the logarithm map with rolling maps. Using rolling maps reduces the distortions introduced in the action curves while mapping to the Lie algebra. Finally, we perform action recognition by classifying the Lie algebra curves using Fourier temporal pyramid representation and a support vector machines classifier. Experimental results show that by combining the logarithm map with rolling maps, we can get improved performance when compared to using the logarithm map alone.

In the third part, we focus on classification of manifold features such as linear subspaces and covariance matrices. We present a kernel-based extrinsic framework for the classification of manifold features and address the issue of kernel selection using multiple kernel learning. We introduce two criteria for jointly learning the kernel and the classifier by solving a single optimization problem. In the case of support vector machine classifier, we formulate the problem of learning a good kernel-classifier combination as a convex optimization problem. The proposed approach performs better than many existing methods for the classification of manifold features when applied to image set-based classification task.

In the fourth part, we propose a novel end-to-end trainable deep network architecture for image denoising based on a Gaussian Conditional Random Field (CRF) model. Contrary to existing discriminative denoising approaches, the proposed network explicitly models the input noise variance and hence is capable of handling a range of noise levels. This network consists of two sub-networks: (i) a parameter generation network that generates the Gaussian CRF pairwise potential parameters based on the input image, and (ii) an inference network whose layers perform the computations involved in an iterative Gaussian CRF inference procedure. Experiments on several images show that the proposed approach produces results on par with the state-of-the-art without training a separate network for each noise level.

In the final part of this dissertation, we propose a Gaussian CRF model-based deep network architecture for the task of semantic image segmentation. This network explicitly models the interactions between output variables which is important for structured prediction tasks such as semantic segmentation. The proposed network is composed of three sub-networks: (i) a Convolutional Neural Network (CNN) based unary network for generating the unary potentials, (ii) a CNN-based pairwise network for generating the pairwise potentials, and (iii) a Gaussian mean field inference network for performing Gaussian CRF inference. When trained end-to-end in a discriminative fashion the proposed network outperforms various CNN-based semantic segmentation approaches.

GEOMETRIC REPRESENTATIONS AND DEEP GAUSSIAN CONDITIONAL RANDOM FIELD NETWORKS FOR COMPUTER VISION

by

Raviteja Vemulapalli

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2016

Advisory Committee:
Professor Rama Chellappa, Chair/Advisor
Professor Larry S. Davis
Professor Min Wu
Professor Amitabh Varshney
Professor Ramani Duraiswami

© Copyright by
Raviteja Vemulapalli
2016

Dedication

To all those researchers who played a role in the advancement of the field of computer vision.

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost, I would like to thank my advisor, Professor Rama Chellappa, for his invaluable guidance over the past five years. I am immensely grateful to him for giving me the freedom to pursue new ideas and work on problems that I was interested in. He has always encouraged me to think out of the box and differentiate myself from the rest, which I believe is very important to be a successful researcher. The discussions I had with him broadened my perspective about the field of computer vision and constantly motivated me throughout my graduate life. I have also learned a great deal about the qualities required to be a successful individual by observing him over the past five years. I feel extremely lucky to have worked with such a great researcher and a wonderful human being.

Next, I would like to thank Dr. Oncel Tuzel from Mitsubishi Electric Research Laboratories (MERL) for his wonderful guidance while I was an intern at MERL. Dr. Oncel Tuzel is one of the sharpest minds in the field of computer vision and I cherish the discussions I had with him. Personally, he is also one of the most wonderful people I have met in my life, and I feel lucky to have spent a year interning with him at MERL. Next, I would like to thank Dr. Jaishanker Pillai, Dr. Felipe Arrate, and Professor Pierre-Antoine Absil for helping me with many technical ideas during the first half of my PhD. They played a crucial role in jump-starting my research career and I am grateful to them.

I would like to thank Professor David Jacobs for his insightful thoughts during his graduate course on manifolds, which were helpful for my research. I would also like to thank Dr. Kevin Zhou, Dr. Hien Van Nguyen who mentored me during my internship at Siemens, and Dr. Ming-Yu Liu for his research inputs during my internship at MERL.

It is an honor to have Professor Larry Davis, Professor Min Wu, Professor Amitabh Varshney and Professor Ramani Duraiswami in my dissertation committee. I am thankful to them for serving in my committee and providing insightful suggestions to improve this dissertation.

I would like to thank the ECE department for selecting me for the ECE Distinguished Dissertation Fellowship Award, and the A. James Clark School of Engineering for selecting me for the Dean's Doctoral Research Award. Both these awards have encouraged me to continue my research in the field of computer vision.

I would like to thank all my colleagues, roommates and friends for making my graduate life memorable. I would also like to thank the staff in UMIACS, ECE and Cfar for helping me in various different ways during my graduate life.

I owe my deepest thanks to my parents and sister who have always stood by me and motivated me throughout my career. Words cannot express the gratitude I owe them.

Finally, I would like to thank the Office of Naval Research (ONR) and Mitsubishi Electric Research Laboratories for funding this research.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AIGD	Affine-Invariant Geodesic Distance
BM3D	Block Matching and 3D filtering
CNN	Convolutional Neural Network
CRF	Conditional Random Field
DTW	Dynamic Time Warping
EPLL	Expected Patch Log Likelihood
FTP	Fourier Temporal Pyramid
GMF	Gaussian Mean Field
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
HOG	Histogram of Oriented Gradients
HQS	Half Quadratic Splitting
InfNet	Inference Network
LDA	Linear Discriminant Analysis
LDS	Linear Dynamical Systems
LED	Log-Euclidean Distance
MAP	Maximum a Posteriori
MKL	Multiple Kernel Learning
MLP	Multi-Layer Percepton
MRF	Markov Random Field
PCA	Principal Component Analysis
PGNet	Parameter Generation Network
PLS	Partial Least Squares
PSNR	Peak Signal-to-Noise Ratio
RBF	Radial Basis Function
RTF	Regression Tree Field
SDP	Semi-Definite Programming
SPD	Symmetric and Positive Definite
SVM	Support Vector Machines

List of Notations

	Vectors are represented using boldface lowercase letters.
	Matrices are represented using boldface uppercase letters.
$\mathcal{R}^n$	n-dimensional real vector space
$\boldsymbol{I}_n$	$n \times n$ identity matrix
$\mathbf{1}$	Matrix of appropriate size with all ones
$\mathbf{0}$	Matrix of appropriate size with all zeros
$\ln(a)$	Natural logarithm of scalar a
$\\|\boldsymbol{v}\\|_2$	ℓ_2 norm of vector $\boldsymbol{v}$
$\boldsymbol{u} \leq \boldsymbol{v}$	Element wise inequalities
$\text{vector}(\boldsymbol{A})$	Column vector representation of matrix $\boldsymbol{A}$
$\boldsymbol{A}(:, i)$	i^{th} column of matrix $\boldsymbol{A}$
$\boldsymbol{A}(i, :)$	i^{th} row of matrix $\boldsymbol{A}$
$\boldsymbol{A}^T$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^{-1}$	Inverse of matrix $\boldsymbol{A}$
$\det(\boldsymbol{A})$	Determinant of matrix $\boldsymbol{A}$
$\text{trace}(\boldsymbol{A})$	Trace of matrix $\boldsymbol{A}$
$\\|\boldsymbol{A}\\|_{Fr}$	Frobenius norm of matrix $\boldsymbol{A}$
$\mathbf{e}^{\boldsymbol{A}}$	Matrix exponential of $\boldsymbol{A}$
$\mathbf{log}(\boldsymbol{A})$	Matrix logarithm of $\boldsymbol{A}$
$\boldsymbol{A} \succeq 0$	$\boldsymbol{A}$ is symmetric and positive semidefinite
$\boldsymbol{A} \succ 0$	$\boldsymbol{A}$ is symmetric and positive definite
$\boldsymbol{A} * \boldsymbol{B}$	Hadamard product between matrices $\boldsymbol{A}$ and $\boldsymbol{B}$
$\otimes$	Direct product between groups
$\oplus$	Direct sum between vector spaces
$T_P\mathcal{M}$	Tangent space to manifold $\mathcal{M}$ at point P

Chapter 1: Introduction

1.1 Motivation

Representation and context modeling are two important factors that have improved the performance of computer vision algorithms over the past two decades. Representations such as Scale-Invariant Feature Transform [3], Histogram of Oriented Gradients (HOG) [4], and more recently, deep network-based features [5–7] have played a crucial role in various applications such as depth estimation, image retrieval, 3D reconstruction, object detection, object recognition, etc. Similarly, context modeling tools such as graphical models [8, 9] have played a crucial role in applications like image enhancement, image segmentation, semantic scene understanding, etc.

While it is widely agreed that representation is the most important component of any computer vision algorithm, most of the existing skeleton-based human action recognition approaches still use simple skeletal representations such as joint positions [10, 11], relative joint positions [12, 13] or joint angles [14, 15]. However, capturing the 3D skeletal geometry in the representation is crucial for achieving good action recognition accuracy. Motivated by this, we introduce various 3D geometry-based skeletal representations for human action recognition.

Manifold features such as linear subspaces [16] and covariance matrices [17,18] are used in various computer vision applications such as image set-based object and face recognition [16,18–23], pedestrian detection [17,24], texture classification [25,26] and activity recognition [16]. Due to the non-Euclidean nature of the underlying feature spaces, these representations are often classified using kernel-based approaches [18,20,22–24]. However, for kernel-based methods, choosing an appropriate kernel is important for achieving good classification performance. Motivated by this, we propose a kernel learning-based extrinsic classification framework to address the issue of kernel-selection for the classification of manifold features.

While modeling the spatial context is important for applications such as image enhancement and semantic image segmentation, the standard deep network architectures [27–31] used for these applications do not explicitly model the spatial context. Motivated by this, we propose novel deep network architectures based on Gaussian Conditional Random Field (CRF) models [9] that explicitly model the spatial context by considering the interactions among the output variables.

1.2 Proposed Algorithms and their Contributions

In this section, we briefly describe the algorithms introduced in this dissertation and their key contributions.

1. **R3DG features for skeleton-based human action recognition:**

   In this part of the dissertation, we try to answer the following basic question: Which is a good skeletal representation for human action recognition? Inspired by the observation that for human actions, the relative geometry between various body parts provides a more meaningful description than their absolute locations, we propose new skeletal representations that explicitly model the relative 3D geometry between all pairs of body parts. Given two rigid body parts, their relative geometry can be described using the rigid body transformation required to take one body part to the position and orientation of the other. Rigid body transformations in 3D space can be mathematically represented in different ways using the special orthogonal group, quaternions, the special Euclidean group, and dual quaternions. Using these mathematical representations, we introduce a family of relative 3D geometry-based skeletal representations for human action recognition, which we refer to as R3DG features. Using the proposed skeletal representations, we model actions as curves in R3DG feature spaces, and perform action recognition using a combination of dynamic time warping [32], Fourier Temporal Pyramid (FTP) representation [13], and a Support Vector Machines (SVM) classifier [33]. The proposed representations outperform many existing skeletal representations when evaluated on several benchmark action recognition datasets. Since the size of the skeleton varies from subject to subject, we need to scale-normalize the skeletal data before using the rigid body transformation-based R3DG features. Instead of doing explicit scale-normalization, we can obtain scale-invariant skeletal representations by using only the rotational part of the rigid body transformation to describe the relative 3D geometry between body parts. In this part of the dissertation, we also show that just by using the relative 3D rotations, we can get a classification accuracy that is close to the accuracy obtained by using the full rigid body transformation-based representations computed from scale-normalized skeletons.

2. **Rolling rotations for skeleton-based human action recognition:** In this part of the dissertation, instead of doing explicit scale-normalization, we use only the rotational part of rigid body transformation to describe the relative 3D geometry between body parts. Since 3D rotations are members of the special orthogonal group $SO(3)$, we represent each skeleton as a point in the product Lie group $SO(3) \otimes \ldots \otimes SO(3)$, and actions as curves in this group. Since classification of temporal curves in the Lie group is difficult due to the non-Euclidean nature of the underlying space, we first map the curves to the corresponding Lie algebra, which is a vector space, and then classify the Lie algebra curves using the FTP representation and an SVM classifier. For mapping the action curves to the Lie algebra, instead of directly using the standard logarithm map, we combine it with rolling maps. We show that rolling maps reduce the distortions in the action curves when mapping them to the Lie algebra and improves the action recognition performance. We also derive new closed form expressions for the rolling maps in the case of piecewise-geodesic rolling curves.

3. **Extrinsic classification of manifold features using kernel learning:** In this part of the dissertation, we try to answer the following important question: How to find good kernels for the classification of manifold features?

Manifold features such as linear subspaces and covariance matrices are used in various computer vision applications. Popular learning algorithms such as Fisher discriminant analysis, partial least squares, support vector machines, etc., are not directly applicable to such features due to the non-Euclidean nature of the underlying spaces. Hence, classification is often performed in an extrinsic manner by mapping the manifolds to Euclidean spaces using kernels. However, for kernel-based approaches, a poor choice of kernel often results in reduced performance. We address this issue of kernel-selection for the classification of manifold features using the kernel learning approach. We propose two criteria for jointly learning the kernel and the classifier using a single optimization problem. Specifically, for the SVM classifier, we formulate the problem of learning a good kernel-classifier combination as a convex optimization problem and solve it efficiently following the multiple kernel learning approach. The proposed approach outperforms various existing methods for the classification of manifold features when evaluated using image set-based object and face recognition tasks.

4. **Deep Gaussian CRF network for image denoising:**

   State-of-the-art deep network-based denoising methods train a separate model for each noise level, which is not desirable. In this part of the dissertation, we address this issue by proposing a novel deep network architecture for image denoising based on a Gaussian CRF model. The proposed deep network explicitly models the input noise variance and hence is capable of handling a range of noise levels. Our deep network architecture consists of two sub-networks: (i) a parameter generation network that generates the pairwise potential parameters based on the noisy input image, and (ii) an inference network whose layers perform the computations involved in an iterative Gaussian CRF inference procedure. All the components of our network are differentiable and hence it can be trained end-to-end using standard gradient-based techniques. Experimental results show that the proposed network can achieve state-of-the-art results without training specific networks for each noise level.

5. **Gaussian CRF network for semantic image segmentation:**

   In the past few years, deep networks have revolutionized the field of computer vision by improving the state-of-the-art results in various applications by a huge margin. However, standard feed-forward networks do not explicitly model the interactions between output variables, which is important for structured prediction tasks such as semantic image segmentation. Traditionally, graphical models, especially the CRF models, have been widely used to model the interactions between output variables. In this part of the dissertation, we combine both these ideas and propose a feed-forward deep network based on a Gaussian CRF model. The proposed Gaussian CRF network is composed of three sub-networks: (i) a Convolutional Neural Network (CNN) based unary network for generating the unary potentials, (ii) a CNN-based pairwise network for generating the pairwise potentials, and (iii) an inference network whose layers perform Gaussian mean field inference. The proposed inference network has the desired property that each of its layers produces an output that is closer to the maximum a posteriori solution of the Gaussian CRF compared to its input. The proposed network significantly improves the semantic segmentation results when compared to standard CNN architectures.

1.3 Organization

This dissertation is organized as follows. Chapter 2 introduces the special orthogonal group, the special Euclidean group, quaternions and dual quaternions, which will be used in subsequent chapters of this dissertation. Chapter 3 presents various relative 3D geometry-based skeletal representations for human action recognition. Chapter 4 discusses the rolling of special orthogonal group and its application to skeleton-based human action recognition. Chapter 5 presents an extrinsic framework for the classification of manifold features using multiple kernel learning. Chapters 6 and 7 present Gaussian CRF-based deep network architectures for image denoising and semantic image segmentation, respectively. Chapter 8 concludes the dissertation and discusses future research directions.

Chapter 2: Lie groups, Quaternions and Dual Quaternions

In this chapter, we introduce the special orthogonal group $SO(3)$, the special Euclidean group $SE(3)$, quaternions and dual quaternions, which will be used in subsequent chapters of this dissertation. Please refer to [34, 35] for additional details on Lie groups, and [35–37] for additional details on quaternions and dual quaternions.

2.1 Lie Groups

A Lie group G is a group that is also a smooth manifold [34]. The tangent space $\mathfrak{g}$ at the identity element e of G is referred to as the Lie algebra of G. A matrix Lie group is a Lie group of $n \times n$ invertible matrices with the usual matrix multiplication and inversion as the group multiplication and inversion operations, and the $n \times n$ identity matrix as the group identity element.

The mapping from a Lie algebra to the corresponding Lie group, referred to as the Lie exponential map, is given by $\mathrm{Lexp}_G(\boldsymbol{u}) = \gamma_{\boldsymbol{u}}(1)$, where $\gamma_{\boldsymbol{u}} : \mathcal{R} \to G$ is the unique one-parameter subgroup of G whose tangent vector at the identity element e is equal to $\boldsymbol{u} \in \mathfrak{g}$. The inverse of Lie exponential map is known as the Lie logarithm map, and is denoted by Llog_G. Figure 2.1 gives an illustration of the Lie exponential

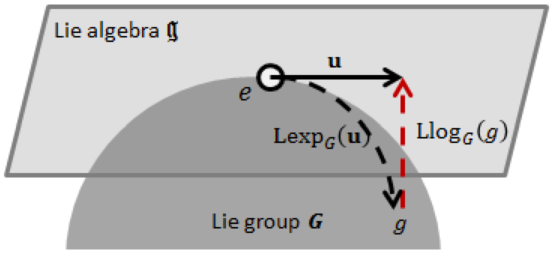

Figure 2.1: Illustration of the Lie exponential and Lie logarithm maps between a Lie group G and its Lie algebra $\mathfrak{g}$.

and Lie logarithms maps. In the case of matrix Lie groups, the Lie exponential and Lie logarithm maps are given by

$$\mathrm{Lexp}_G(\boldsymbol{u}) = \mathbf{e}^{\boldsymbol{u}}, \quad \mathrm{Llog}_G(g) = \mathbf{log}(g), \tag{2.1}$$

where $\mathbf{e}$ and $\mathbf{log}$ represent the usual matrix exponential and logarithm, respectively.

2.1.1 Special Orthogonal Group $SO(3)$

The special orthogonal group $SO(3)$ is a three dimensional matrix Lie group formed by the set of all 3×3 matrices $\boldsymbol{R}$ that satisfy the following constraints [34,35]:

$$\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I}_3, \ \det(\boldsymbol{R}) = 1. \tag{2.2}$$

The Lie algebra of $SO(3)$, denoted by $\mathfrak{so}(3)$, is the three dimensional vector space spanned by the set of all 3×3 skew symmetric matrices. For any element

$$\boldsymbol{A} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \in \mathfrak{so}(3), \tag{2.3}$$

its vector representation is given by $\mathrm{vec}(\boldsymbol{A}) = [a_1, a_2, a_3]$. Since $SO(3)$ is a matrix Lie group, the Lie exponential and Lie logarithm maps between $SO(3)$ and $\mathfrak{so}(3)$ are given by

$$\mathrm{Lexp}_{SO(3)}(\boldsymbol{A}) = \mathbf{e}^{\boldsymbol{A}}, \quad \mathrm{Llog}_{SO(3)}(\boldsymbol{R}) = \mathbf{log}(\boldsymbol{R}). \tag{2.4}$$

The Lie logarithm map is not unique in the case of $SO(3)$. In this dissertation, we use the $\mathbf{log}(\boldsymbol{R})$ with the smallest norm.

Elements of $SO(3)$ are commonly used to represent 3D rotations. Let $\boldsymbol{z}'$ be a 3D point obtained by rotating $\boldsymbol{z} \in \mathcal{R}^3$ by an angle θ about an axis $\boldsymbol{n}$. Then, we have

$$\boldsymbol{z}' = \mathbf{e}^{skew(\theta \boldsymbol{n})} \boldsymbol{z}, \tag{2.5}$$

where $skew(\theta \boldsymbol{n})$ is a skew-symmetric matrix that satisfies $\mathrm{vec}(skew(\theta \boldsymbol{n})) = \theta \boldsymbol{n}$. Hence, the matrix $\mathbf{e}^{skew(\theta \boldsymbol{n})} \in SO(3)$ represents the 3D rotation by an angle θ about an axis $\boldsymbol{n}$.

Riemannian geometry of $\boldsymbol{SO(3)}$ [38]: Along with being a Lie group, $SO(3)$ is also a Riemannian manifold. The tangent space $T_{\boldsymbol{R}_0}SO(3)$ at $\boldsymbol{R}_0 \in SO(3)$ is the vector space spanned by the set of all 3×3 matrices $\boldsymbol{A}$ such that $\boldsymbol{A} = \boldsymbol{\Omega} \boldsymbol{R}_0$ for some skew-symmetric matrix $\boldsymbol{\Omega}$. The inner product in the tangent space $T_{\boldsymbol{R}_0}SO(3)$ is given by the Frobenius inner product:

$$\langle \boldsymbol{A}_1, \boldsymbol{A}_2 \rangle_{\boldsymbol{R}_0} = \mathrm{trace}(\boldsymbol{A}_1^\top \boldsymbol{A}_2), \;\; \boldsymbol{A}_1, \boldsymbol{A}_2 \in T_{\boldsymbol{R}_0}SO(3). \tag{2.6}$$

Under this Riemannian metric, the exponential and logarithm maps between $SO(3)$

and its tangent space at $\boldsymbol{R}_0 \in SO(3)$ are given by

$$\begin{aligned} \exp_{SO(3)}(\boldsymbol{R}_0, \boldsymbol{A}) &= \mathbf{e}^{\boldsymbol{A}\boldsymbol{R}_0^\top}\boldsymbol{R}_0,\ \boldsymbol{A} \in T_{\boldsymbol{R}_0}SO(3), \\ \log_{SO(3)}(\boldsymbol{R}_0, \boldsymbol{R}_1) &= \mathbf{log}(\boldsymbol{R}_1\boldsymbol{R}_0^\top)\boldsymbol{R}_0,\ \boldsymbol{R}_1 \in SO(3). \end{aligned} \tag{2.7}$$

The geodesic curve from $\boldsymbol{R}_0$ to $\boldsymbol{R}_1$ is given by $\mathbf{e}^{t\ \mathbf{log}(\boldsymbol{R}_1\boldsymbol{R}_0^\top)}\boldsymbol{R}_0,\ t \in [0,1]$, and the geodesic distance between $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ is given by $\|\log_{SO(3)}(\boldsymbol{R}_0, \boldsymbol{R}_1)\|_{Fr}$.

Interpolation: Various approaches have been proposed in the past for interpolation on $SO(3)$ [39]. In this dissertation, we use a simple piecewise geodesic interpolation scheme. Given $\boldsymbol{R}_1, \ldots, \boldsymbol{R}_m \in SO(3)$ at time instances $t_1, \ldots, t_m$ respectively, we use the following curve for interpolation:

$$\gamma(t) = \boldsymbol{R}_i \ \mathrm{Lexp}_{SO(3)}\left(\frac{t - t_i}{t_{i+1} - t_i}\boldsymbol{A}_i\right) \text{ for } t \in [t_i, t_{i+1}], \tag{2.8}$$

where $\boldsymbol{A}_i = \mathrm{Llog}_{SO(3)}\left(\boldsymbol{R}_i^{-1}\boldsymbol{R}_{i+1}\right)$ for $i = 1, 2, \ldots, m-1$.

$\boldsymbol{SO(3)} \otimes \ldots \otimes \boldsymbol{SO(3)}$: We can combine multiple $SO(3)$ groups using the direct product to form a new Lie group

$$SO(3)^n := SO(3) \otimes \ldots \otimes SO(3) \tag{2.9}$$

with the corresponding Lie algebra

$$\mathfrak{so}(3)^n := \mathfrak{so}(3) \oplus \ldots \oplus\ \mathfrak{so}(3). \tag{2.10}$$

The Lie exponential and Lie logarithm maps for $(\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_n) \in \mathfrak{so}(3)^n$ and $(\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_n) \in SO(3)^n$ are given by

$$\begin{aligned} \mathrm{Lexp}_{SO(3)^n}(\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_n) &= (\mathbf{e}^{\boldsymbol{A}_1}, \mathbf{e}^{\boldsymbol{A}_2}, \ldots, \mathbf{e}^{\boldsymbol{A}_n}), \\ \mathrm{Llog}_{SO(3)^n}(\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_n) &= (\mathbf{log}(\boldsymbol{R}_1), \mathbf{log}(\boldsymbol{R}_2), \ldots, \mathbf{log}(\boldsymbol{R}_n)). \end{aligned} \tag{2.11}$$

Interpolation on $SO(3)^n$ can be performed by simultaneously interpolating on individual $SO(3)$.

2.1.2 Special Euclidean Group $SE(3)$

The special Euclidean group $SE(3)$ [34, 35] is a six dimensional matrix Lie group formed by the set of all 4×4 matrices of the form

$$\boldsymbol{P}(\boldsymbol{R}, \boldsymbol{d}) = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{d} \\ 0 & 1 \end{bmatrix}, \ \boldsymbol{d} \in \mathcal{R}^3, \ \boldsymbol{R} \in SO(3). \tag{2.12}$$

The Lie algebra of $SE(3)$, denoted by $\mathfrak{se}(3)$, is the six dimensional vector space spanned by the set of all 4×4 matrices of the form

$$\boldsymbol{B} = \begin{bmatrix} 0 & -a_3 & a_2 & w_1 \\ a_3 & 0 & -a_1 & w_2 \\ -a_2 & a_1 & 0 & w_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.13}$$

The vector representation of $\boldsymbol{B} \in \mathfrak{se}(3)$ is given by

$$\text{vec}(\boldsymbol{B}) = [a_1, a_2, a_3, w_1, w_2, w_3]. \tag{2.14}$$

Since $SE(3)$ is a matrix Lie group, the Lie exponential and Lie logarithm maps between $SE(3)$ ans $\mathfrak{se}(3)$ are given by

$$\text{Lexp}_{SE(3)}(\boldsymbol{B}) = \mathbf{e}^{\boldsymbol{B}}, \ \ \text{Llog}_{SE(3)}(\boldsymbol{P}) = \mathbf{log}(\boldsymbol{P}). \tag{2.15}$$

The Lie logarithm map is not unique in the case of $SE(3)$. In this dissertation, we use the $\mathbf{log}(\boldsymbol{P})$ with the smallest norm.

Elements of $SE(3)$ are commonly used to represent 3D rigid body transformations. Let $\boldsymbol{z}'$ be a 3D point obtained by transforming $\boldsymbol{z} \in \mathcal{R}^3$ using a rotation by an angle θ about an axis $\boldsymbol{n}$ followed by a translation $\boldsymbol{d}$. Then, we have

$$\begin{bmatrix} \boldsymbol{z}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{e}^{skew(\theta \boldsymbol{n})} & \boldsymbol{d} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{z} \\ 1 \end{bmatrix}. \tag{2.16}$$

Hence, the matrix $\begin{bmatrix} \boldsymbol{R} & \boldsymbol{d} \\ 0 & 1 \end{bmatrix} \in SE(3)$, where $\boldsymbol{R} = \mathbf{e}^{skew(\theta \boldsymbol{n})}$, represents the 3D rigid body transformation composed of a rotation by an angle θ about an axis $\boldsymbol{n}$ and a translation $\boldsymbol{d}$.

Interpolation: Various approaches have been proposed in the past for interpolation on $SE(3)$ [40,41]. In this dissertation, we use a simple piecewise interpolation scheme based on screw motions [42]. Given $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_m \in SE(3)$ at time instances $t_1, \ldots, t_m$ respectively, we use the following curve for interpolation:

$$\gamma(t) = \boldsymbol{P}_i \ \mathrm{Lexp}_{SE(3)} \left(\frac{t - t_i}{t_{i+1} - t_i} \boldsymbol{B}_i \right) \ \text{ for } t \in [t_i, t_{i+1}], \tag{2.17}$$

where $\boldsymbol{B}_i = \mathrm{Llog}_{SE(3)} \left(\boldsymbol{P}_i^{-1} \boldsymbol{P}_{i+1} \right)$ for $i = 1, 2, \ldots, m-1$.

$\boldsymbol{SE(3) \otimes \ldots \otimes SE(3)}$: We can combine multiple $SE(3)$ groups using the direct product to form a new Lie group

$$SE(3)^n := SE(3) \otimes \ldots \otimes SE(3) \tag{2.18}$$

with the corresponding Lie algebra

$$\mathfrak{se}(3)^n := \mathfrak{se}(3) \oplus \ldots \oplus \ \mathfrak{se}(3). \tag{2.19}$$

The Lie exponential and Lie logarithm maps for $(\boldsymbol{B}_1, \boldsymbol{B}_2, \ldots, \boldsymbol{B}_n) \in \mathfrak{se}(3)^n$ and $(\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n) \in SE(3)^n$ are given by

$$\begin{aligned} \mathrm{Lexp}_{SE(3)^n}(\boldsymbol{B}_1, \boldsymbol{B}_2, \ldots, \boldsymbol{B}_n) &= (\mathbf{e}^{\boldsymbol{B}_1}, \mathbf{e}^{\boldsymbol{B}_2}, \ldots, \mathbf{e}^{\boldsymbol{B}_n}), \\ \mathrm{Llog}_{SE(3)^n}(\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n) &= (\mathbf{log}(\boldsymbol{P}_1), \mathbf{log}(\boldsymbol{P}_2), \ldots, \mathbf{log}(\boldsymbol{P}_n)). \end{aligned} \tag{2.20}$$

Interpolation on $SE(3)^n$ can be performed by simultaneously interpolating on individual $SE(3)$.

2.2 Quaternions

The set of quaternions $\mathcal{Q}$ [35, 43–45] is equivalent to the 4-dimensional vector space $\mathcal{R}^4$ equipped with the quaternion multiplication operation. Let $\{\mathbf{e_0}, \mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}\}$ be the canonical basis for the vector space $\mathcal{R}^4$. The quaternion multiplication is defined by giving the following multiplication table for the basis:

$$\begin{aligned} &\mathbf{e_0 e_1} = \mathbf{e_1 e_0} = \mathbf{e_1}, \ \ \mathbf{e_1 e_2} = -\mathbf{e_2 e_1} = \mathbf{e_3}, \\ &\mathbf{e_0 e_2} = \mathbf{e_2 e_0} = \mathbf{e_2}, \ \ \mathbf{e_2 e_3} = -\mathbf{e_3 e_2} = \mathbf{e_1}, \\ &\mathbf{e_0 e_3} = \mathbf{e_3 e_0} = \mathbf{e_3}, \ \ \mathbf{e_3 e_1} = -\mathbf{e_1 e_3} = \mathbf{e_2}, \\ &\mathbf{e_0 e_0} = \mathbf{e_0}, \qquad\qquad \mathbf{e_1 e_1} = \mathbf{e_2 e_2} = \mathbf{e_3 e_3} = -\mathbf{1}. \end{aligned} \tag{2.21}$$

A quaternion $\boldsymbol{q}$ is commonly represented as $(s_q, \boldsymbol{v}_q)$, where $s_q \in \mathcal{R}$ is referred to as the scalar or real part and $\boldsymbol{v}_q \in \mathcal{R}^3$ is referred to as the vector or imaginary part. Addition of two quaternions $\boldsymbol{p} = (s_p, \boldsymbol{v}_p)$ and $\boldsymbol{q} = (s_q, \boldsymbol{v}_q)$ is given by

$$\boldsymbol{p} + \boldsymbol{q} = (s_p + s_q, \boldsymbol{v}_p + \boldsymbol{v}_q). \tag{2.22}$$

Using (2.21), multiplication of $\boldsymbol{p}$ and $\boldsymbol{q}$ can be computed as

$$\boldsymbol{pq} = (s_p s_q - \boldsymbol{v}_p \odot \boldsymbol{v}_q, s_p \boldsymbol{v}_q + s_q \boldsymbol{v}_p + \boldsymbol{v}_p \times \boldsymbol{v}_q), \tag{2.23}$$

where $\boldsymbol{v}_p \odot \boldsymbol{v}_q$ and $\boldsymbol{v}_p \times \boldsymbol{v}_q$ represent the dot product and cross product between $\boldsymbol{v}_p$ and $\boldsymbol{v}_q$, respectively. Note that the quaternion multiplication is not commutative.

The conjugate $\bar{\boldsymbol{q}}$, the norm $\|\boldsymbol{q}\|$, and the exponential $e^{\boldsymbol{q}}$ of a quaternion $\boldsymbol{q} = (s_q, \boldsymbol{v}_q)$ are given by

$$\begin{aligned} \bar{\boldsymbol{q}} &= (s_q, -\boldsymbol{v}_q),\ \ \|\boldsymbol{q}\| = \sqrt{s_q^2 + \|\boldsymbol{v}_q\|_2^2}, \\ e^{\boldsymbol{q}} &= \left(e^{s_q} \cos(\|\boldsymbol{v}_q\|_2),\ \ e^{s_q} \sin(\|\boldsymbol{v}_q\|_2) \frac{\boldsymbol{v}_q}{\|\boldsymbol{v}_q\|_2} \right). \end{aligned} \tag{2.24}$$

The quaternions with unit norm are known as *unit quaternions*. The set of unit quaternions, denoted by $\mathcal{UQ}$, forms a Lie group with quaternion multiplication as the group multiplication operation, and $\boldsymbol{q}_e = (1, \mathbf{0})$ as the group identity element. The Lie algebra of $\mathcal{UQ}$, denoted by $\mathfrak{uq}$, is the three dimensional vector space spanned by the set of purely imaginary quaternions. The Lie exponential and Lie logarithm maps for $\boldsymbol{w} \in \mathfrak{uq}$ and $\boldsymbol{q} = (s_q, \boldsymbol{v}_q) \in \mathcal{UQ}$ are given by

$$\begin{aligned} \text{Lexp}_{\mathcal{UQ}}(\boldsymbol{w}) &= e^{\boldsymbol{w}} = \left(\cos(\|\boldsymbol{w}\|_2),\ \sin(\|\boldsymbol{w}\|_2) \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} \right), \\ \text{Llog}_{\mathcal{UQ}}(\boldsymbol{q}) &= \cos^{-1}(s_q) \frac{\boldsymbol{v}_q}{\sqrt{1 - s_q^2}}. \end{aligned} \tag{2.25}$$

Unit quaternions are commonly used to represent rotations in 3D space. Let $\boldsymbol{z}$ be a 3D point, and $\boldsymbol{q}_z = (0, \boldsymbol{z})$ be its quaternion representation. Let $\boldsymbol{z}'$ be a 3D point obtained by rotating $\boldsymbol{z}$ by an angle θ about an axis $\boldsymbol{n}$, and $\boldsymbol{q}_{z'} = (0, \boldsymbol{z}')$ be its quaternion representation. Then, we have $\boldsymbol{q}_{z'} = \boldsymbol{r}\boldsymbol{q}_z\bar{\boldsymbol{r}}$, where $\boldsymbol{r} = e^{\boldsymbol{n}\frac{\theta}{2}}$. Hence, the unit quaternion

$$e^{\boldsymbol{n}\frac{\theta}{2}} = \left(\cos\left(\frac{\theta}{2}\right),\ \boldsymbol{n} \sin\left(\frac{\theta}{2}\right) \right) \tag{2.26}$$

represents the 3D rotation by an angle θ about the axis $\boldsymbol{n}$. We can easily convert between unit quaternion and $SO(3)$ representations using

$$\begin{aligned} &\boldsymbol{r} = \mathrm{Lexp}_{\mathcal{UQ}}\left(\frac{\boldsymbol{w}}{2}\right),\ \boldsymbol{w} = \mathrm{vec}\left(\mathrm{Llog}_{SO(3)}(\boldsymbol{R})\right), \\ &\boldsymbol{R} = \mathrm{Lexp}_{SO(3)}(skew(\boldsymbol{w})),\ \boldsymbol{w} = 2\left(\mathrm{Llog}_{\mathcal{UQ}}(\boldsymbol{r})\right). \end{aligned} \tag{2.27}$$

$\boldsymbol{\mathcal{UQ} \otimes \ldots \otimes \mathcal{UQ}}$: We can combine multiple $\mathcal{UQ}$ groups using the direct product to form a new Lie group

$$\mathcal{UQ}^n := \mathcal{UQ} \otimes \ldots \otimes \mathcal{UQ} \tag{2.28}$$

with the corresponding Lie algebra

$$\mathfrak{uq}^n := \mathfrak{uq} \oplus \ldots \oplus\ \mathfrak{uq}. \tag{2.29}$$

2.3 Dual Quaternions

The set of dual quaternions $\mathcal{D}$ is the extension of quaternions using dual number theory [37]. Each dual quaternion consists of eight elements or two quaternions:

$$\boldsymbol{\zeta} = \boldsymbol{q}_r + \epsilon \boldsymbol{q}_d, \tag{2.30}$$

where $\boldsymbol{q}_r = (s_r, \boldsymbol{v}_r), \boldsymbol{q}_d = (s_d, \boldsymbol{v}_d)$ are quaternions, and ϵ is the dual operator, i.e., $\epsilon^2 = 0, \epsilon \neq 0$. The dual quaternion addition, multiplication, conjugate and magnitude are given by

$$\begin{aligned} &\boldsymbol{\zeta}_1 + \boldsymbol{\zeta}_2 = (\boldsymbol{q}_{1r} + \boldsymbol{q}_{2r}) + \epsilon(\boldsymbol{q}_{1d} + \boldsymbol{q}_{2d}), \\ &\boldsymbol{\zeta}_1\boldsymbol{\zeta}_2 = \boldsymbol{q}_{1r}\boldsymbol{q}_{2r} + \epsilon(\boldsymbol{q}_{1r}\boldsymbol{q}_{2d} + \boldsymbol{q}_{1d}\boldsymbol{q}_{2r}), \\ &\bar{\boldsymbol{\zeta}} = \bar{\boldsymbol{q}}_r + \epsilon\bar{\boldsymbol{q}}_d, \\ &\|\boldsymbol{\zeta}\| = \|\boldsymbol{q}_r\| + \epsilon\left(\frac{s_r s_d + \boldsymbol{v}_r \odot \boldsymbol{v}_d}{\|\boldsymbol{q}_r\|}\right). \end{aligned} \tag{2.31}$$

Note that the magnitude of dual quaternion is a dual number. Dual quaternions that satisfy

$$\|\boldsymbol{\zeta}\| = 1, \text{ i.e., } \|\boldsymbol{q}_r\| = 1, \; s_r s_d + \boldsymbol{v}_r \odot \boldsymbol{v}_d = 0, \tag{2.32}$$

are called *unit dual quaternions.* We denote the set of all unit dual quaternions using $\mathcal{UD}$.

While a unit quaternion can represent a 3D rotation, a unit dual quaternion can represent a full 3D rigid body transformation, i.e, both rotation and translation. Let $\boldsymbol{z}$ be a 3D point, and $\boldsymbol{\zeta}_z = (1, \boldsymbol{0}) + \epsilon(0, \boldsymbol{z})$ be its dual quaternion representation. Let $\boldsymbol{z}'$ be a 3D point obtained by transforming $\boldsymbol{z}$ using a rotation by an angle θ about an axis $\boldsymbol{n}$ followed by a translation $\boldsymbol{d}$, and $\boldsymbol{\zeta}_{z'} = (1, \boldsymbol{0}) + \epsilon(0, \boldsymbol{z}')$ be its dual quaternion representation. Then, we have $\boldsymbol{\zeta}_{z'} = \boldsymbol{\zeta}_{rd}\boldsymbol{\zeta}_z\bar{\boldsymbol{\zeta}}_{rd}$, where

$$\begin{aligned} \boldsymbol{\zeta}_{rd} &= \boldsymbol{r} + \epsilon\left(\frac{1}{2}\boldsymbol{t}\boldsymbol{r}\right) \in \mathcal{UD}, \\ \boldsymbol{r} &= e^{\boldsymbol{n}\frac{\theta}{2}} \in \mathcal{UQ}, \; \boldsymbol{t} = (0, \boldsymbol{d}) \in \mathcal{Q}. \end{aligned} \tag{2.33}$$

Hence, the unit dual quaternion $\boldsymbol{\zeta}_{rd}$ represents the 3D rigid body transformation composed of a rotation by an angle θ about an axis $\boldsymbol{n}$ and a translation $\boldsymbol{d}$.

Chapter 3: Relative 3D Geometry-based Skeletal Representations for Human Action Recognition

3.1 Introduction

Human action recognition has been an active area of research for the past several decades due to its applications in surveillance, video games, human computer interaction, robotics, health care, etc. In the past few decades, several approaches have been proposed for recognizing human actions from monocular RGB video sequences [46, 47]. Unfortunately, the monocular RGB data is highly sensitive to various factors like illumination changes, variations in view-point, occlusions and background clutter. Moreover, monocular video sensors do not fully capture the human motion in a 3D space. Hence, despite significant research efforts over the past few decades, human action recognition still remains a challenging problem.

Human body can be represented as an articulated system of rigid segments connected by joints, and human motion can be considered as a continuous evolution of the spatial configuration of these rigid segments [48–50]. So, if we can reliably extract the human skeleton, action recognition can be performed by classifying its temporal evolution. Using skeletal data for action recognition has several advantages such as ease of interpretability, low processing time, fast/cheap transmission and storage, etc. Skeletal data makes it easier to analyze which part of the body is playing a major role in discriminating one action against the other, and allows us to correlate this with human interpretation of motion. Interpretability is an important factor in various applications such as exercise monitoring, human computer interaction, post-surgery rehabilitation, etc. Skeletons provide a compact low-dimensional representation that can be stored easily, transmitted and processed quickly. Storage and transmission are critical in applications where the recognition module runs on a central server.

Unfortunately, extracting the human skeleton from monocular RGB videos is a very difficult task [51]. Sophisticated motion capture systems can be used to get the 3D locations of landmarks placed on the human body, but such systems are very expensive, and require the user to wear a motion capture suit with markers which can hinder natural movements. With the recent availability of cost-effective depth sensors, extracting the human skeleton has become relatively easier. These sensors provide 3D depth data of the scene, which is robust to illumination changes and offers more useful information to infer human skeletons. Recently, a quick method was proposed in [52] to accurately estimate the 3D positions of skeletal joints using a single depth image. These recent advances have generated a renewed interest in skeleton-based human action recognition.

Existing skeleton-based action recognition approaches can be broadly grouped into two main categories: joint-based approaches and body part-based approaches. Inspired by the moving lights display experiment of [53], joint-based approaches

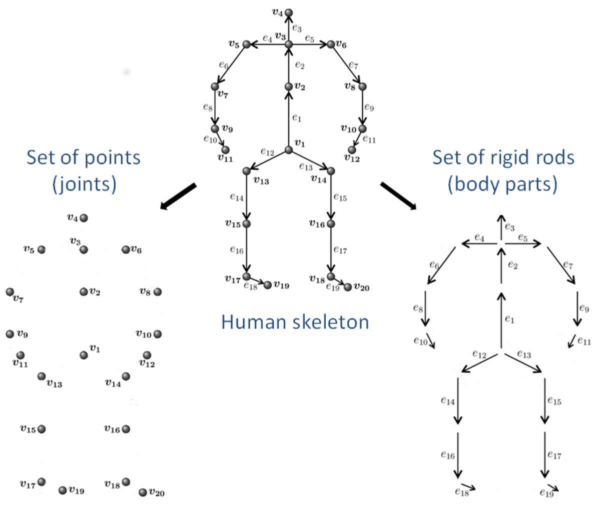

Figure 3.1: Two views of a human skeleton

consider the human skeleton as a set of points (Figure 3.1 left). These approaches try to model the motion of either the individual joints or combinations of multiple joints using various features like joint positions [11, 54–56], joint orientations with respect to a fixed root node [57, 58], pairwise relative joint positions [13, 59, 60], etc. On the other hand, motivated by the 3D-shape representations of [61], body part-based approaches consider the human skeleton as a connected set of rigid segments (Figure 3.1 right). These approaches either model the temporal evolution of individual body parts [62] or focus on directly-connected pairs of body parts and model the temporal evolution of joint angles [14, 15, 63].

In this chapter, we introduce a new family of body part-based skeletal representations for recognizing human actions. Inspired by the observation that for human actions, the relative geometry between various body parts (though not directly connected by a joint) provides a more meaningful description than their absolute locations (for example, clapping is more intuitively described using the relative geometry between the two hands), we explicitly model the relative 3D geometry between different body parts in our skeletal representations.

Given two rigid body parts, their relative geometry can be described using the rigid body transformation (rotation and translation) required to take one body part to the position and orientation of the other. Hence, we use the rigid body transformations between all pairs of body parts to represent a human skeleton. Rigid body transformations in 3D space can be mathematically represented in various ways using the special orthogonal group $SO(3)$, quaternions, the special Euclidean group $SE(3)$, and dual quaternions. Using these mathematical representations, we introduce a family of relative 3D geometry-based skeletal representations for action recognition, which we refer to as R3DG features.

One of the major issues while working with skeletal-data is scale variation. This can be handled by normalizing all the skeletons (without changing the joint angles) such that their body part lengths are equal to the corresponding lengths of a fixed reference skeleton. Interestingly, while the relative translations between various body parts vary with this scale normalization, the relative rotations do not change. Hence, we can get scale-invariant skeletal representations by using only the relative rotations between different body parts. In this chapter, we experimentally show that just by using the relative 3D rotations, we can get a recognition accuracy that is close to the accuracy obtained by using the full rigid body transformation-based representations computed from scale-normalized skeletons. This suggests that the translational information might possibly be redundant for human action recognition when the 3D rotations between all pairs of body parts are used.

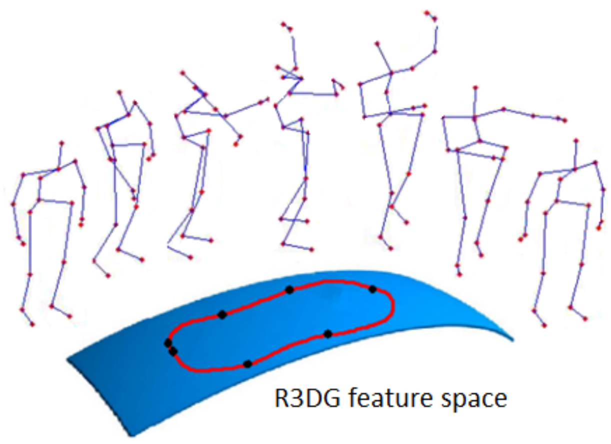

Figure 3.2: Representation of an action as a curve in an R3DG feature space.

Using any of the proposed skeletal representations, human actions can be modeled as curves (Figure 3.2) in an R3DG feature space, and action recognition can be performed by classifying these curves. Irrespective of the skeletal representation being used, classification of temporal sequences into different action categories is a difficult problem due to various issues like rate variations, temporal misalignment, noise, etc. To handle rate variations, for each action category, we compute a nominal curve using Dynamic Time Warping (DTW) [32], and warp all the curves to this nominal curve. Then, we represent the warped curves using the low frequency FTP representation, which was shown to be robust to noise and temporal misalignment in [13]. Finally, classification is performed using an SVM classifier with the FTP representation.

Contributions: We introduce a new family of body part-based 3D skeletal representations for human action recognition. The proposed representations explicitly model the relative geometry between various body parts using 3D rigid body transformations. We use the special Euclidean group and dual quaternions in our skeletal representations. To the best of our knowledge, they have not been explored before in the context of skeleton-based human action recognition. We experimentally show that the proposed representations outperform several existing skeletal representations by evaluating them on several benchmark action datasets. We also introduce scale-invariant skeletal representations that use only the 3D rotations between various body parts. We experimentally show that the performance of the scale-invariant rotation-only representations is very close to that of the full rigid body transformation-based representations.

Organization: Section 3.2 provides an overview of existing skeleton-based human action recognition approaches. Section 3.3 introduces the proposed family of R3DG features, and Section 3.4 presents the proposed temporal modeling and classification approach. Experimental results and conclusions are presented in Sections 3.5 and 3.6, respectively.

3.2 Related Work

In this section, we provide an overview of various existing skeleton-based human action recognition approaches. Various depth map-based action recognition approaches have also been proposed in the recent past, which use features extracted from the 3D depth data. Since the focus of this chapter is on skeleton-based action recognition, we refer the readers to [64, 65] for a review of depth map-based recognition approaches.

Existing skeleton-based action recognition approaches can be broadly grouped into two main categories: joint-based approaches and body part-based approaches. While the joint-based approaches consider the human skeleton as a set of independent points, the body part-based approaches consider the human skeleton as a connected set of rigid segments. Approaches that use joint angles for representing the human skeleton can be classified as part-based approaches since joint angles measure the geometry between pairs of body parts that are directly connected to each other.

3.2.1 Joint-based Approaches

A set of 13 joint trajectories in XYZT space was used to represent human actions in [54], and their affine projections were compared using a subspace angle-based similarity measure. In [55], the trajectories of individual joints and groups of joints were modeled using Hidden Markov Models (HMMs). Each HMM was considered as a weak classifier, which were then combined using AdaBoost. HMMs were also used in [66] to model the joint trajectories of whole body, upper body and lower body separately for performing action recognition.

The 3D joint locations were combined with silhouette-based features in [67], and their temporal evolutions were compared using DTW. Dynamic time warping was also used in [56] for comparing the sequences of joint positions. Instead of giving equal weight to all the joints in the DTW distance computation, a feature weighting approach was used in [56], where each joint was assigned its own weight. In [11], the temporal evolutions of joint locations were modeled using a temporal hierarchy of covariance features, and action recognition was performed using an SVM classifier. In [10], the 3D trajectory of each joint was projected onto three Cartesian planes to get three 2D trajectories. Each 2D trajectory was represented using the histogram of displacements between consecutive points. The histograms from all the joints were concatenated to get the final representation, which was classified using an SVM classifier. Recently, hierarchical recurrent neural networks were used in [68] for modeling the temporal dynamics of skeletal joints.

A view invariant representation of human skeleton was obtained in [57] by quantizing the 3D joint locations into histograms based on their orientations with respect to a coordinate system attached to the hip center. The temporal evolutions of this representation were modeled using HMMs. In [69], human skeletons were represented using 3D joint positions, their first and second order derivatives, i.e., joint velocities and accelerations, and a nearest neighbor-based approach was used to perform low-latency action recognition. In [58], one of the joints was selected as a root joint, and all the remaining joints were represented using their orientations with respect to a coordinate system attached to the root joint. The temporal evolutions of this representation were compared using dynamic time warping.

In [13, 59], pairwise relative positions of the joints were used to represent the human skeleton, and the temporal evolutions of this representation were modeled using low-frequency Fourier coefficients [13] and wavelets [59]. A similar skeletal representation was also used in [12], where a discriminative learning-based temporal alignment method was used for comparing temporal sequences.

In [70], the human skeleton was represented using distances between all pairs of joints in the current frame, distances between all pairs of joints in the current frame and the previous frame, distances between all pairs of joints in the current frame and the first frame of the sequence. Action recognition was then performed using a logistic regression-based approach. In [60], the human skeleton was represented using relative joint positions, temporal displacements of the joints, and offsets of the joints with respect to the initial frame. Action classification was then performed using the Naive-Bayes nearest neighbor rule in a low-dimensional space obtained using Principal Component Analysis (PCA). A similar representation was also used in [71] along with random forests.

A local skeleton descriptor, referred to as *skeletal quad*, was introduced in [72], which encodes the relative position of joint quadruples. This descriptor represents a set of four joints using the coordinates of third and fourth joints in a coordinate system with the first joint as the origin and the second joint as $(1, 1, 1)$. These skeletal quads were combined with Fisher vectors [73] and a linear SVM classifier to perform action recognition. An interesting aspect of this descriptor is that it can be used to represent the relative 3D geometry between two body parts (since two body parts can be considered as four joints). However, the main difference between the skeletal quad descriptor and the proposed R3DG features is that while the proposed features directly use the translation and rotation between body parts, the skeletal quad descriptor encodes this information indirectly using the joint coordinates.

In [74], human skeleton was divided into five parts and each part was represented using the coordinates of the joints that belonged to the part. Then, a dictionary of pose templates was learned for each body part, and these templates were used to obtain a quantized representation of part poses. The authors further defined spatial-part-sets to capture the spatial configurations of multiple body parts, and temporal-part-sets to capture the joint pose evolutions of multiple body parts. Finally, the bag-of-words model was used to get the action representation, which was classified using a one-vs-one intersection kernel SVM classifier.

Different from the above mentioned approaches, [75] introduced various types of relational pose features that describe the geometric relations between specified joints of the skeleton, and used them successfully for indexing and retrieval of motion capture data. Similar features were later used in [76–78] for skeleton-based human action recognition.

3.2.2 Part-based Approaches

In [62], the human body was divided into five different parts, and human actions were represented using the motion parameters of individual parts like horizontal and vertical translations, in-plane rotations, etc. Principal component analysis was used to represent an action as a linear combination of a set of basis actions, and classification was performed by comparing the PCA coefficients. In [79], human skeletons were divided into smaller parts and each body part was represented using certain bio-inspired shape features. The temporal evolutions of these bio-inspired features were modeled using Linear Dynamical Systems (LDS), and a discriminative metric learning approach was used for comparing the LDS models.

In [63], human skeletons were represented using 3D joint angles, and the temporal evolutions of this representation were compared using DTW. While [80] represented human actions as curves in low-dimensional phase spaces related to joint angles, [15] represented human actions using pairwise affinities between joint angle trajectories. In [81], human skeletons were represented using joint angle quaternions. These skeletal features were augmented with RGB and depth-based HOG features, and a maximum entropy Markov model was used for action detection. In [14], a set of few informative skeletal joints was selected at each time instance based on highly interpretable measures such as mean and variance of joint angles, angular velocity of the joints, etc. Human actions were represented as sequences of these informative joints, which were compared using the normalized edit distance.

3.3 Relative 3D Geometry-based Skeletal Representations

Let $S = (V, E)$ be a skeleton, where $V = \{v_1, \ldots, v_N\}$ denotes the set of joints and $E = \{e_1, \ldots, e_M\}$ denotes the set of oriented rigid body parts. Let e_{m1}, e_{m2} denote the starting and end points of e_m, respectively.

Given a pair of body parts e_m and e_n, to describe their relative 3D geometry, we use the rigid body transformations required to take one body part to the position and orientation of the other. A full rigid body transformation T is composed of a rotation by an angle θ about an axis $\boldsymbol{n}$ and a translation $\boldsymbol{d}$. To measure the rigid body transformation $T_{m,n} = (\theta_{m,n}, \boldsymbol{n}_{m,n}, \boldsymbol{d}_{m,n})$ required to take e_n to the position and orientation of e_m, we use a local coordinate system attached to e_n (Figure 3.3(a)).

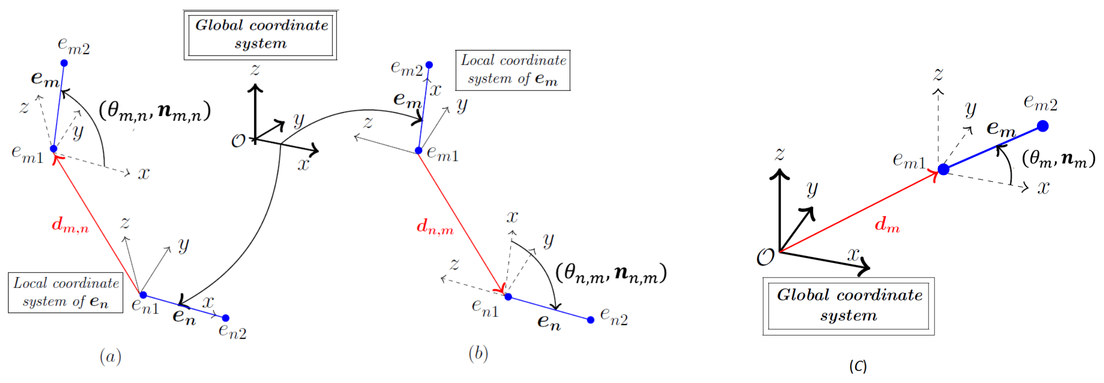
Global coordinate system
Local coordinate system of e_n
Local coordinate system of e_m
$(\theta_{m,n}, \boldsymbol{n}_{m,n})$
$\boldsymbol{d}_{m,n}$
$(\theta_{n,m}, \boldsymbol{n}_{n,m})$
$\boldsymbol{d}_{n,m}$
$(\theta_m, \boldsymbol{n}_m)$
$\boldsymbol{d}_m$
e_{m1}
e_{m2}
e_{n1}
e_{n2}
O
(a)
(b)
(C)

Figure 3.3: (a) Rigid body transformation $T_{m,n} = (\theta_{m,n}, \boldsymbol{n}_{m,n}, \boldsymbol{d}_{m,n})$ from e_n to e_m measured in the coordinate system of e_n, (b) Rigid body transformation $T_{n,m} = (\theta_{n,m}, \boldsymbol{n}_{n,m}, \boldsymbol{d}_{n,m})$ from e_m to e_n measured in the coordinate system of e_m, (c) Rigid body transformation $T_m = (\theta_m, \boldsymbol{n}_m, \boldsymbol{d}_m)$ of e_m with respect to global x-axis.

Similarly, to measure the rigid body transformation $T_{n,m} = (\theta_{n,m}, \boldsymbol{n}_{n,m}, \boldsymbol{d}_{n,m})$ required to take e_m to the position and orientation of e_n, we use a local coordinate system attached to e_m (Figure 3.3(b)). We obtain the local coordinate system of a body part e_m by rotating (with minimum rotation) and translating the global coordinate system such that e_{m1} becomes the origin and e_m coincides with the x-axis.

At first glance it might appear that using only $T_{m,n}$ or $T_{n,m}$ would be sufficient to represent the relative geometry between e_m and e_n. Consider the case in which e_n is rotating about an axis parallel to e_m. Though there is relative motion between the two, $T_{m,n}$ will not change. Similarly, if e_m is rotating about an axis parallel to e_n, then $T_{n,m}$ will not change. So, if we represent the relative geometry using only one of them, the representation will not change under certain kinds of relative motions, which is undesirable. Hence, we use both $T_{m,n}$ and $T_{n,m}$ to represent the relative geometry between e_m and e_n. Note that both $T_{m,n}$ and $T_{n,m}$ do not change only when both e_m and e_n undergo same rotation and translation.

Using the relative geometry between all pairs of body parts, we represent a skeleton S at time instance t using

$$C(t) = (T_{1,2}(t),\ T_{2,1}(t),\ \ldots,\ T_{M-1,M}(t),\ T_{M,M-1}(t)), \tag{3.1}$$

where M is the number of body parts. The total number of rigid body transformations used in the skeletal representation is $K = M(M-1)$. Using the proposed representation, a skeletal sequence describing an action can be represented as a curve $\{C(t),\ t \in [0, T']\}$, and action recognition can be performed by classifying such curves into different action categories.

Note that we are using only the relative measurements $T_{m,n}(t)$ in our skeletal representation. We also performed experiments by adding the absolute 3D locations of body parts to the skeletal representation. The 3D location of a body part e_m can be described using its rigid body transformation T_m with respect to the global x-axis (Figure 3.3(c)). But, this did not give any improvement, suggesting that the absolute measurements are redundant when the relative measurements are used.

3.3.1 R3DG Features

There are multiple ways to mathematically represent the rigid body transformations in 3D space. In this chapter, we consider the following four representations: $SE(3)$, $SO(3) \otimes \mathcal{R}^3$, $\mathcal{UQ} \otimes \mathcal{R}^3$, and $\mathcal{UD}$. Using each representation we get a full rigid body transformation-based R3DG feature. Please refer to Chapter 2 for details about the special Euclidean group $SE(3)$, the special orthogonal group $SO(3)$, the unit quaternions $\mathcal{UQ}$, and the unit dual quaternions $\mathcal{UD}$.

$\boldsymbol{SE(3)}$ **:** Each rigid body transformation $T_{i,j}(t)$ is represented as a member of $SE(3)$ using the 4×4 matrix

$$\boldsymbol{P}_{i,j}(t) = \begin{bmatrix} \boldsymbol{R}_{i,j}(t) & \boldsymbol{d}_{i,j}(t) \\ 0 & 1 \end{bmatrix}, \tag{3.2}$$

where $\boldsymbol{R}_{i,j}(t)$ is the $SO(3)$ representation of 3D rotation $(\theta_{i,j}(t), \boldsymbol{n}_{i,j}(t))$, and the entire skeleton is represented using

$$\big(\boldsymbol{P}_{1,2}(t),\ \boldsymbol{P}_{2,1}(t), \ldots, \boldsymbol{P}_{M-1,M}(t), \boldsymbol{P}_{M,M-1}(t)\big) \in SE(3)^K. \tag{3.3}$$

Since $SE(3)^K$ is a curved space, classification of action curves in this space is not an easy task. Standard classification approaches like SVM, which are defined for vector space representations, are not directly applicable to this non-vector space. Also, temporal modeling approaches like Fourier analysis are not applicable to this space. Note that the standard Fourier analysis is defined for functions whose output varies along the real line. Here, the action curve $C(t)$ evolves in the non-Euclidean space $SE(3)^K$ as a function of time, and the standard Fourier analysis is not defined for this case. To overcome these difficulties, we map the action curves from the Lie group $SE(3)^K$ to its Lie algebra $\mathfrak{se}(3)^K$, which is a $6M(M-1)$-dimensional vector space. The final representation of action curve $C(t)$ is given by

$$\begin{aligned} \mathfrak{C}_1(t) = \big[\ &\mathrm{vec}(\mathbf{log}(\boldsymbol{P}_{1,2}(t))), \mathrm{vec}(\mathbf{log}(\boldsymbol{P}_{2,1}(t))), \ldots, \\ &\qquad \mathrm{vec}(\mathbf{log}(\boldsymbol{P}_{M-1,M}(t))), \mathrm{vec}(\mathbf{log}(\boldsymbol{P}_{M,M-1}(t)))\big]. \end{aligned} \tag{3.4}$$

$\boldsymbol{SO(3)\otimes\mathcal{R}^3}$ **:** In this case, the rotations and translations are separately represented as members of $SO(3)$ and $\mathcal{R}^3$, respectively, and the entire skeleton is represented using

$$\begin{aligned}&\big(\boldsymbol{R}_{1,2}(t), \boldsymbol{R}_{2,1}(t), \dots, \boldsymbol{R}_{M-1,M}(t), \boldsymbol{R}_{M,M-1}(t),\\ &\qquad \boldsymbol{d}_{1,2}(t), \boldsymbol{d}_{2,1}(t), \dots, \boldsymbol{d}_{M-1,M}(t), \boldsymbol{d}_{M,M-1}(t)\big) \in SO(3)^K \otimes \mathcal{R}^{3K}.\end{aligned} \tag{3.5}$$

Similar to $SE(3)^K$, the Lie group $SO(3)^K$ is also a curved space. So, we map the action curves from $SO(3)^K \otimes \mathcal{R}^{3K}$ to the $6M(M-1)$-dimensional vector space $\mathfrak{so}(3)^K \oplus \mathcal{R}^{3K}$ by mapping the rotational part from the Lie group $SO(3)^K$ to its Lie algebra $\mathfrak{so}(3)^K$. Note that the translational part remains the same. The final vector space representation of action curve $C(t)$ is given by

$$\begin{aligned}\mathfrak{C}_2(t) = \big[\ &\text{vec}(\mathbf{log}(\boldsymbol{R}_{1,2}(t))), \text{vec}(\mathbf{log}(\boldsymbol{R}_{2,1}(t))), \dots, \text{vec}(\mathbf{log}(\boldsymbol{R}_{M-1,M}(t))),\\ &\text{vec}(\mathbf{log}(\boldsymbol{R}_{M,M-1}(t))), \boldsymbol{d}_{1,2}(t), \boldsymbol{d}_{2,1}(t), \dots, \boldsymbol{d}_{M-1,M}(t), \boldsymbol{d}_{M,M-1}(t)\big].\end{aligned} \tag{3.6}$$

$\boldsymbol{\mathcal{UQ}\otimes\mathcal{R}^3}$ **:** In this case, the rotations and translations are separately represented as elements of $\mathcal{UQ}$ and $\mathcal{R}^3$, respectively, and the entire skeleton is represented using

$$\begin{aligned}&\big(\boldsymbol{r}_{1,2}(t), \boldsymbol{r}_{2,1}(t), \dots, \boldsymbol{r}_{M-1,M}(t), \boldsymbol{r}_{M,M-1}(t),\\ &\qquad \boldsymbol{d}_{1,2}(t), \boldsymbol{d}_{2,1}(t), \boldsymbol{d}_{M-1,M}(t), \boldsymbol{d}_{M,M-1}(t)\big) \in \mathcal{UQ}^K \otimes \mathcal{R}^{3K},\end{aligned} \tag{3.7}$$

where $\boldsymbol{r}_{i,j}(t) = (s_{i,j}(t), \boldsymbol{v}_{i,j}(t))$ is the unit quaternion representation of 3D rotation $(\theta_{i,j}(t), \boldsymbol{n}_{i,j}(t))$.

Similar to $SO(3)$ and $SE(3)$, the Lie group $\mathcal{UQ}$ is also a curved surface. In fact, the set of unit quaternions forms a three dimensional unit sphere in $\mathcal{R}^4$. Hence, to get a vector space representation, we directly use the 4-dimensional ambient space representation of unit quaternions. With this, we get the following $7M(M-1)$-dimensional vector space representation for the action curve $C(t)$:

$$\begin{aligned}\mathfrak{C}_3(t) = \big[&s_{1,2}(t), \boldsymbol{v}_{1,2}(t), s_{2,1}(t), \boldsymbol{v}_{2,1}(t), \ldots, s_{M-1,M}(t), \boldsymbol{v}_{M-1,M}(t), \\ &s_{M,M-1}(t), \boldsymbol{v}_{M,M-1}(t), \boldsymbol{d}_{1,2}(t), \boldsymbol{d}_{2,1}(t), \ldots, \boldsymbol{d}_{M-1,M}(t), \boldsymbol{d}_{M,M-1}(t)\big].\end{aligned} \tag{3.8}$$

Here, we could have used the Lie algebra representation instead of the ambient space representation. But, the $\mathfrak{uq}$ representation is nothing but a scaled version (a scaling factor of 1/2) of $\mathfrak{so}(3)$ representation (refer to (2.27)). Since $\mathfrak{so}(3)$ representation is already being used in the case of $SO(3) \otimes \mathcal{R}^3$, we chose to use the ambient space representation for unit quaternions.

$\boldsymbol{\mathcal{UD}}$ **:** In this case, each rigid body transformation $T_{i,j}(t)$ is represented using a unit dual quaternion

$$\boldsymbol{\zeta}_{i,j}(t) = \left(s^r_{i,j}(t),\ \boldsymbol{v}^r_{i,j}(t)\right) + \epsilon\left(s^d_{i,j}(t),\ \boldsymbol{v}^d_{i,j}(t)\right). \tag{3.9}$$

The set of unit dual quaternions does not form a vector space. Hence, similar to the quaternions, we use the 8-dimensional ambient space representation for unit dual quaternions, which gives the following $8M(M-1)$-dimensional vector space representation for the action curve $C(t)$:

$$\begin{aligned}\mathfrak{C}_4(t) = \big[&s^r_{1,2}(t), \boldsymbol{v}^r_{1,2}(t), s^d_{1,2}(t), \boldsymbol{v}^d_{1,2}(t), s^r_{2,1}(t), \boldsymbol{v}^r_{2,1}(t), s^d_{2,1}(t), \boldsymbol{v}^d_{2,1}(t), \ldots, \\ &s^r_{M-1,M}(t), \boldsymbol{v}^r_{M-1,M}(t), s^d_{M-1,M}(t), \boldsymbol{v}^d_{M-1,M}(t), \\ &s^r_{M,M-1}(t), \boldsymbol{v}^r_{M,M-1}(t), s^d_{M,M-1}(t), \boldsymbol{v}^d_{M,M-1}(t)\big].\end{aligned} \tag{3.10}$$

Table 3.1: The proposed family of R3DG features.

R3DG feature		Dimensionality	Needs scale normalization
Rigid body transformation-based	$\mathfrak{se}(3)$	$6M(M-1)$	Yes
	$\mathfrak{so}(3) \otimes \mathcal{R}^3$	$6M(M-1)$	Yes
	$\mathcal{UQ} \otimes \mathcal{R}^3$	$7M(M-1)$	Yes
	$\mathcal{UD}$	$8M(M-1)$	Yes
Rotation-based	$\mathfrak{so}(3)$	$3M(M-1)$	No
	$\mathcal{UQ}$	$4M(M-1)$	No

3.3.2 Scale-invariant R3DG Features

One of the standard ways to handle scale variations in skeletal data is to resize all the skeletons to a fixed size. This can be done by normalizing the skeletons (without changing the joint angles) such that their body part lengths are equal to the corresponding lengths of a reference skeleton. Interestingly, while the translations between different body parts vary with this scale normalization, the 3D rotations do not change. So, by using only the rotations between different body parts, we can get the following two scale-invariant R3DG features based on the $\mathfrak{so}(3)$ and $\mathcal{UQ}$ representations of rotations:

$$\begin{aligned}\mathfrak{C}_5(t) = \big[& \operatorname{vec}(\mathbf{log}(\boldsymbol{R}_{1,2}(t))), \operatorname{vec}(\mathbf{log}(\boldsymbol{R}_{2,1}(t))), \ldots, \\ & \operatorname{vec}(\mathbf{log}(\boldsymbol{R}_{M-1,M}(t))), \operatorname{vec}(\mathbf{log}(\boldsymbol{R}_{M,M-1}(t)))\big], \\ \mathfrak{C}_6(t) = \big[& s_{1,2}(t), \boldsymbol{v}_{1,2}(t), s_{2,1}(t), \boldsymbol{v}_{2,1}(t), \ldots, \\ & s_{M-1,M}(t), \boldsymbol{v}_{M-1,M}(t), s_{M,M-1}(t), \boldsymbol{v}_{M,M-1}(t)\big].\end{aligned} \tag{3.11}$$

Note that at any time instance t, $\mathfrak{C}_5(t)$ is a $3M(M-1)$-dimensional vector and $\mathfrak{C}_6(t)$ is a $4M(M-1)$-dimensional vector. Table 3.1 summarizes the proposed family of R3DG features.

3.4 Temporal Modeling and Classification

Classification of vector space curves into different action categories is not a straightforward task due to various issues like rate variations, temporal misalignment, noise, etc. Following [32, 82], we use DTW to handle rate variations. During training, for each action category, we compute a nominal curve using the algorithm described in Table 3.2, and warp all the training curves to this nominal curve. We use the squared Euclidean distance for DTW computations. Note that for computing a nominal curve all the curves should have equal number of samples. For this, we re-sample the action curves using the interpolation algorithms presented for $SO(3)$ and $SE(3)$ in Sections 2.1.1 and 2.1.2, respectively. In the case of quaternions, we first interpolate the rotations on $SO(3)$ and then convert them to unit quaternions. In the case of dual quaternions, we first interpolate the rigid body transformations on $SE(3)$ and then convert them to unit dual quaternions.

After the DTW step, we represent the warped curves by using the low-frequency FTP representation that was shown to be robust to temporal misalignment and noise in [13]. We apply FTP for each dimension separately and concatenate the low-frequency Fourier coefficients to obtain the final feature vector. Action recognition is performed by classifying the final feature vectors using a one-vs-all SVM

Table 3.2: Algorithm for computing a nominal curve.

```
Input: Curves $\mathcal{S}_1(t), \ldots, \mathcal{S}_J(t)$ at $t = 0, 1, \ldots, T'$.
Maximum number of iterations $max$ and threshold $\delta$.
Output: Nominal curve $\mathcal{S}(t)$ at $t = 0, 1, \ldots, T'$.
Initialization: $\mathcal{S}(t) = \mathcal{S}_1(t)$, iter = 0.
while iter < $max$
    Warp each curve $\mathcal{S}_j(t)$ to the nominal curve $\mathcal{S}(t)$ using DTW with
    squared Euclidean distance to get a warped curve $\mathcal{S}_j^w(t)$.
    Compute a new nominal $\mathcal{S}'(t)$ using $\mathcal{S}'(t) = \frac{1}{J}\sum_{j=1}^{J} \mathcal{S}_j^w(t)$.
    if $\sum_{t=0}^{T'} \|\mathcal{S}'(t) - \mathcal{S}(t)\|_2^2 \leq \delta$
        break
    end
    $\mathcal{S}(t) = \mathcal{S}'(t)$;  iter = iter + 1;
end
```

classifier. Figure 3.4 gives an overview of the proposed skeleton-based action recognition approach. The top row shows all the steps involved in training and the bottom row shows all the steps involved in testing.

3.5 Experimental Evaluation

In this section, we evaluate the proposed R3DG features on five action datasets: MSRAction3D [83], UTKinect-Action [57], Florence3D [84], MSRPairs [85] and G3D [86]. Please refer to Table 3.3 for details about these datasets.

Basic pre-processing: To make the skeletal data invariant to the absolute location of human in the scene, all the 3D joint coordinates were transformed from world

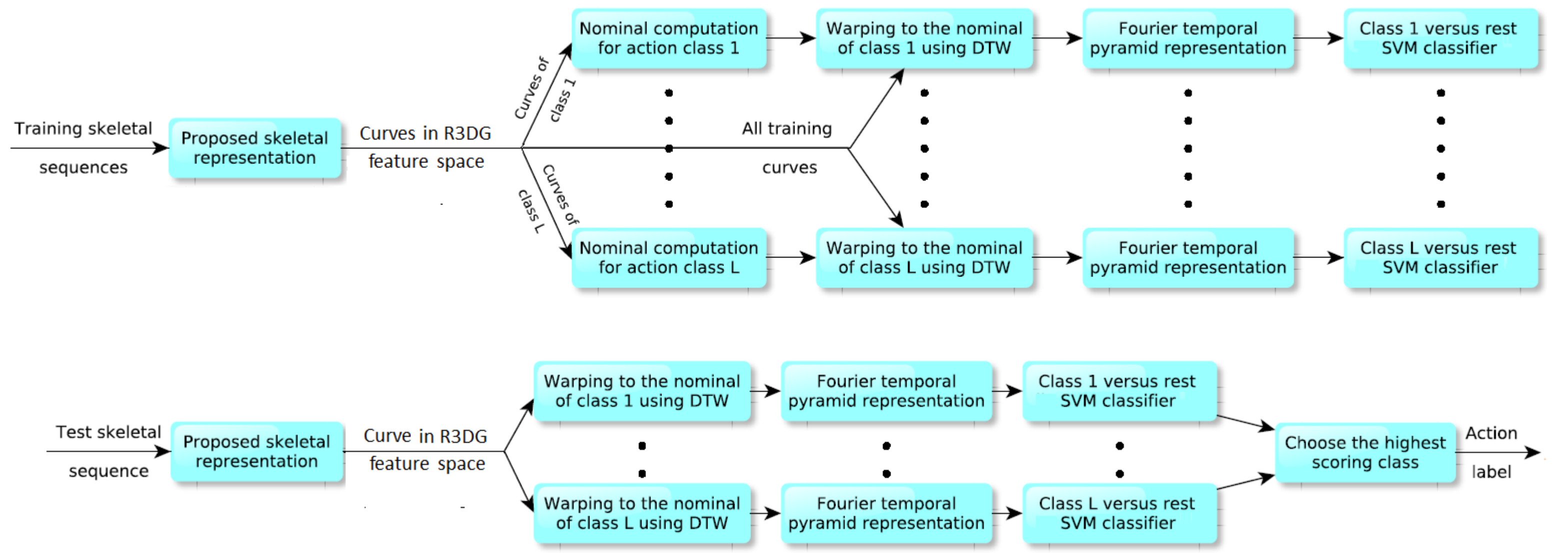
Training skeletal
sequences
Proposed skeletal representation
Curves in R3DG
feature space
Curves of
class 1
Curves of
class L
All training
curves
Nominal computation for action class 1
Warping to the nominal of class 1 using DTW
Fourier temporal pyramid representation
Class 1 versus rest SVM classifier
Nominal computation for action class L
Warping to the nominal of class L using DTW
Fourier temporal pyramid representation
Class L versus rest SVM classifier
Test skeletal
sequence
Proposed skeletal representation
Curve in R3DG
feature space
Warping to the nominal of class 1 using DTW
Fourier temporal pyramid representation
Class 1 versus rest SVM classifier
Warping to the nominal of class L using DTW
Fourier temporal pyramid representation
Class L versus rest SVM classifier
Choose the highest scoring class
Action
label

Figure 3.4: The proposed action recognition approach: Top row - Training, Bottom row - Testing.

Table 3.3: Datasets for skeleton-based human action recognition.

Dataset	MSRAction3D	UTKinect-Action	Florence3D	MSRPairs	G3D
Actions	20	10	9	12	20
Subjects	10	10	10	10	10
Sequences	557	199	215	353	663
Joints	20	20	15	20	20
Body parts	19	19	14	19	19

coordinate system to a person-centric coordinate system by placing the hip center at the origin. We rotated the skeletons such that the ground plane projection of the vector from left hip to right hip is parallel to the global x-axis. For each dataset, we took one of the subjects as reference, and normalized all the other skeletons (without changing their joint angles) such that their body part lengths are equal to the corresponding lengths of the reference skeleton. This normalization takes care of scale variations. We also performed experiments by varying the reference subject, but the results did not vary much. The standard deviation in the recognition accuracy was around 0.2-0.3%.

Alternative skeletal representations: To show the effectiveness of the proposed R3DG features, we compare them with the following alternative representations:

- **Joint positions (JP):** Concatenation of 3D coordinates of all the joints $v_1, \ldots, v_N$ (except the hip center).
- **Relative positions of the joints (RJP):** Concatenation of all the vectors $\overrightarrow{v_i v_j},\ 1 \leq i < j \leq N$.

- **Joint angles (JA):** Concatenation of the quaternion representations of all the joint angles. We also tried the $\mathfrak{so}(3)$ and Euler-angle representations for joint angles, but quaternions gave the best results. Here, we measure each joint angle in the local coordinate systems of both body parts associated with that angle.

- **Relation pose features (RP):** We use the joint distance, plane, normal plane, velocity and normal velocity features of [77] computed from a single human skeleton.

- **Individual body part locations (BPL):** Each body part e_m is represented using its rigid body transformation T_m with respect to the global x-axis (Figure 3.3(c)). Similar to R3DG features, we have six different BPL features: $\mathfrak{se}(3)$, $\mathfrak{so}(3) \otimes \mathcal{R}^3$, $\mathcal{UQ} \otimes \mathcal{R}^3$, $\mathcal{UD}$, $\mathfrak{so}(3)$, and $\mathcal{UQ}$.

- **Skeletal quads (SQ):** We use the skeletal quad descriptor of [72] to describe the relative geometry between every pair of body parts.

For a fair comparison, we use the same temporal modeling and classification approach described in Section 3.4 with all the representations. Table 3.4 summarizes the alternative skeletal representations used for comparison.

Parameters: For the FTP representation, we used a three-level temporal pyramid with one-fourth of the segment length as the number of low-frequency coefficients. While using one or two levels for the temporal pyramid produced inferior results, going beyond three did not improve the results significantly. Changing the number

Table 3.4: Alternative skeletal representations for comparison.

Representation	JP	RJP	JA	RP	SQ
Dimensionality	$3(N-1)$	$\frac{3}{2}N(N-1)$	$8M$	$N(75+\frac{N-1}{2})$	$6M(M-1)$
Requires scale normalization	Yes	Yes	No	Yes	Yes

Representation	BPL					
	$\mathfrak{se}(3)$	$\mathfrak{so}(3) \otimes \mathcal{R}^3$	$\mathcal{UQ} \otimes \mathcal{R}^3$	$\mathcal{UD}$	$\mathfrak{so}(3)$	$\mathcal{UQ}$
Dimensionality	$6M$	$6M$	$7M$	$8M$	$3M$	$4M$
Requires scale normalization	Yes	Yes	Yes	Yes	No	No

of low-frequency coefficients from one-fourth of the segment length to one-third or one-fifth did not significantly change the accuracy (around 0.2%). The value of SVM parameter C was chosen using cross-validation. For each dataset, all the curves were re-sampled to have same length. The reference length was chosen to be the maximum number of samples in any curve in the dataset before re-sampling.

Comparison with other skeletal representations: Table 3.5 shows the recognition accuracy for various skeletal representations on five action datasets when the same temporal modeling and classification pipeline (DTW + FTP + linear SVM) is used with all the representations. For all the datasets, we followed the cross-subject test setting, in which half of the subjects were used for training and the other half were used for testing. All the results reported in this table were averaged over ten different random combinations of training and test data. The best result in each column is shown in boldface style. We can see that all the proposed R3DG features

Table 3.5: Recognition accuracy for various skeletal representations.

Representation		MSRAction3D	UTKinect	Florence3D	MSRPairs	G3D
JP		88.75	95.08	85.26	92.90	87.28
RJP		88.87	95.48	85.17	93.91	90.03
JA		75.39	94.07	80.45	90.46	86.25
RP		87.25	93.46	76.86	84.76	88.19
SQ		83.44	95.18	88.89	90.70	89.79
BPL	$\mathfrak{se}(3)$	82.97	94.58	81.38	89.62	87.40
	$\mathfrak{so}(3) \otimes \mathcal{R}^3$	83.88	94.67	81.26	90.59	87.19
	$\mathcal{UQ} \otimes \mathcal{R}^3$	88.74	96.18	84.94	93.32	89.48
	$\mathcal{UD}$	87.76	95.48	83.95	92.75	88.73
	$\mathfrak{so}(3)$	82.46	94.37	80.52	90.70	86.64
	$\mathcal{UQ}$	86.30	95.18	83.46	92.41	87.76
R3DG	$\mathfrak{se}(3)$	89.55	**97.20**	90.71	93.65	91.60
	$\mathfrak{so}(3) \otimes \mathcal{R}^3$	89.37	**97.20**	90.87	93.82	91.60
	$\mathcal{UQ} \otimes \mathcal{R}^3$	90.24	97.09	**92.61**	93.60	91.51
	$\mathcal{UD}$	**90.69**	97.09	91.55	**94.33**	**92.12**
	$\mathfrak{so}(3)$	89.22	96.78	90.52	93.48	91.48
	$\mathcal{UQ}$	90.59	96.88	91.27	93.88	**92.12**

perform better than all the alternative skeletal representations on all the datasets except the MSRPairs dataset where the RJP representation performs slightly better than some of the R3DG features. Specifically, the accuracy of the best R3DG feature is better than the accuracy of the best competing skeletal representation by 1.82% on the MSRAction3D dataset, 1.02% on the UTKinect dataset, 3.72% on the Florence3D dataset, 0.42% on the MSRPairs dataset, and 2.09% on the G3D dataset. These results clearly show the superiority of the proposed R3DG features.

Table 3.6: Contribution of the FTP module in terms of recognition accuracy

Dataset		$\mathfrak{se}(3)$	$\mathfrak{so}(3) \otimes \mathcal{R}^3$	$\mathcal{UQ} \otimes \mathcal{R}^3$	$\mathcal{UD}$	$\mathfrak{so}(3)$	$\mathcal{UQ}$
MSRAction3D	DTW + SVM	87.71	87.52	90.30	90.59	86.96	90.23
	DTW + FTP + SVM	89.55	89.37	90.24	90.69	89.22	90.59
	FTP contribution	1.84	1.85	-0.06	0.10	2.26	0.36
G3D	DTW + SVM	88.13	88.28	89.73	89.73	87.92	89.52
	DTW + FTP + SVM	91.60	91.60	91.51	92.12	91.48	92.12
	FTP contribution	3.47	3.32	1.78	2.39	3.56	2.60

Contribution of the translational information: Comparing the recognition accuracy of rotation-based and full rigid body transformation-based R3DG features, we can see that on four out of five datasets, the contribution of translational information is not that significant. The difference between the best recognition accuracy of rotation-based and full rigid body transformation-based R3DG features is 0.1% for the MSRAction3D dataset, 0.32% for the UTKinect dataset, 0.45% for the MSR-Pairs dataset, and 0% for the G3D dataset. Only on the Florence3D dataset, there is a significance difference of around 1.34%.

Contribution of the DTW and FTP modules: Our temporal modeling consists of the DTW and FTP modules. To analyze the contribution of these modules to the final recognition accuracy, we performed experiments on the MSRAction3D and G3D datasets (the two largest datasets) by removing these modules from the action recognition pipeline. Table 3.6 compares the final accuracy with and without the FTP module in the action recognition pipeline. As we can see, the FTP module contributes significantly to the final accuracy in most of the cases.

Table 3.7: Contribution of the DTW module in terms of recognition accuracy

Dataset		$\mathfrak{se}(3)$	$\mathfrak{so}(3) \otimes \mathcal{R}^3$	$\mathcal{UQ} \otimes \mathcal{R}^3$	$\mathcal{UD}$	$\mathfrak{so}(3)$	$\mathcal{UQ}$
MSRAction3D	FTP + SVM	86.93	86.94	87.58	87.69	86.42	87.26
	DTW + FTP + SVM	89.55	89.37	90.24	90.69	89.22	90.59
	DTW contribution	2.62	2.43	2.66	3.00	2.80	3.33
G3D	FTP + SVM	91.75	91.75	91.48	92.12	91.60	92.12
	DTW + FTP + SVM	91.60	91.60	91.51	92.12	91.48	92.12
	DTW contribution	-0.15	-0.15	0.03	0	-0.12	0

Table 3.7 compares the final accuracy with and without the DTW module in the action recognition pipeline. While the DTW module contributes significantly to the final accuracy in the case of MSRAction3D dataset, it does not change the accuracy much in the case of G3D dataset. This variation is expected since the contribution of DTW module depends on the rate variations present in the dataset.

Comparison with state-of-the-art approaches: Table 3.8 compares the proposed approach with various existing skeleton-based action recognition approaches on MSRAction3D, UTKinect, Florence3D and G3D datasets (MSRPairs dataset is missing since all the results reported in the literature for this dataset are based on depth data). Since the focus of this work is on skeleton-based human action recognition, we use only skeleton-based approaches for comparison. We evaluated our approach using both linear and RBF kernel SVMs, and the kernel SVM performed slightly better on all datasets. When the RBF kernel was used, the $\mathcal{UQ}$ R3DG feature gave the best result (among all R3DG features) for the G3D dataset and the $\mathcal{UQ} \times \mathcal{R}^3$ R3DG feature gave the best result on all the remaining datasets.

Table 3.8: Comparison with other skeleton-based action recognition approaches.

MSRAction3D dataset	
Bag of key poses [67]†	89.62
Random forests [71]†	90.90
HOD features [10]†	91.26
Covariance descriptors [11]†	90.53
Spatial and temporal part-sets [74]†	90.22
Skeletal quads [72]†	89.86
Moving pose [69]†	91.70
Actionlets [13]	88.20
MMTW [12]	**92.70**
Motion Trajectories [87]	92.10
Hanklets [88]	89.00
Joint angle similarities [15]	83.53
Proposed approach (linear SVM)	89.74
Proposed approach (Kernel SVM)	90.11

† Easier three subset protocol.

UTKinect dataset	
Histograms of 3D joints [57]**	90.92
Hanklets [88]**	86.76
Motion Trajectories [87]**	91.50
Random forests [71]	87.90
Elastic functional coding [89]	94.87
Proposed approach (linear SVM)	97.20
Proposed approach (Kernel SVM)	**97.59**
Florence3D dataset	
Multi-Part Bag-of-Poses [84]*	82.00
Motion Trajectories [87]*	87.04
Elastic functional coding [89]	89.67
Proposed approach (linear SVM)	92.61
Proposed approach (Kernel SVM)	**93.06**
G3D dataset	
RBM + HMM [90]	86.40
Proposed approach (linear SVM)	92.12
Proposed approach (kernel SVM)	**92.39**

* Easier leave-one-actor-out scheme.

** Easier leave-one-action-out scheme.

For the MSRAction3D dataset, we followed the standard protocol of using subjects 1, 3, 5, 7, 9 for training and the remaining for testing. For G3D, UTKinect and Florence3D datasets, we followed the cross-subject setting and used half of the subjects for training and the remaining half for testing. Note that this is a more difficult setting compared to the leave-one-action-out scheme used for the UTKinect dataset in [57, 87, 88] and the leave-one-actor-out scheme used for the Florence3D dataset in [84,87], where more subjects were used for training. We report the results averaged over ten random combinations of training and test data.

The best accuracy on each dataset is shown in boldface style. We can see that the proposed approach gives the best recognition accuracy on three out of four datasets. Specifically, it is better than the state-of-the-art results by 2.72% on the UTKinect dataset, 3.39% on the Florence3D dataset and 5.99% on the G3D dataset. The proposed approach also outperforms many recent skeleton-based action recognition approaches on the MSRAction3D dataset. Note that the main focus of this work is on skeletal representation, and the proposed R3DG features clearly outperform various existing skeletal representations when the same classification pipeline is used with all the representations.

3.6 Conclusions

In this chapter, we introduced a family of body part-based 3D skeletal representations for human action recognition, which we refer to as R3DG features. The proposed representations explicitly model the relative 3D geometry between various body parts (though not directly connected by a joint) using rigid body transformations. We represented 3D rigid body transformations using $SE(3)$, $SO(3) \otimes \mathcal{R}^3$, $\mathcal{UQ} \otimes \mathcal{R}^3$, and $\mathcal{UD}$, resulting in four different R3DG features. We also introduced scale-invariant R3DG features by using only the 3D rotations between various body parts. Using the proposed representations, we modeled the human actions as curves in R3DG feature spaces. Finally, we performed action recognition by classifying these curves using a combination of DTW, the FTP representation and an SVM classifier. We experimentally showed that the proposed R3DG features perform better than various existing skeletal representations, and the proposed action recognition approach outperforms various existing skeleton-based action recognition approaches.

Chapter 4: Rolling the Special Orthogonal Group for Skeleton-based Human Action Recognition

4.1 Introduction

In Chapter 3, we introduced scale-invariant skeletal representations for human action recognition based on the special orthogonal group $SO(3)$ and unit quaternions. In this chapter, we further investigate the $SO(3)$-based representation. Given a skeletal sequence, we represent each skeleton as a point in the Lie group $SO(3) \otimes \ldots \otimes SO(3)$ using the relative 3D rotations between all pairs of body parts, and the entire sequence as a curve in the Lie group. A similar $SO(3)$-based representation was also used in [91, 92] to represent human skeletons. However, while [91, 92] used only the joint orientations, our skeletal representation includes the 3D rotations between all pairs of body parts.

Classification of curves in the Lie group $SO(3) \otimes \ldots \otimes SO(3)$ is a non-trivial task due to the non-Euclidean nature of the underlying space. In Chapter 3, we first mapped the action curves from the Lie group to its Lie algebra (which is the tangent space at the identity element) using the logarithm map, and then classified the Lie algebra curves. But, flattening the Lie group using the logarithm map at a

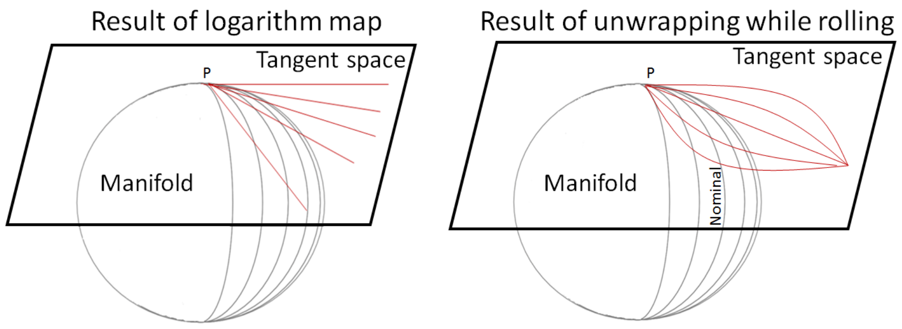

Figure 4.1: Left: Logarithm map at point P, Right: Unwrapping using the logarithm map while rolling along the nominal curve.

single point introduces distortions due to which curves that are nearby in the Lie group can move away from each other in the Lie algebra. Figure 4.1 (left) illustrates this pictorially with the example of a sphere. Here, the longitudinal curves moves away from each other when mapped to the tangent space at P using the logarithm map. Note that though we use a sphere for illustration in Figure 4.1, the manifold of interest here is $SO(3) \otimes \ldots \otimes SO(3)$.

To reduce the distortions introduced by flattening the Lie group using the logarithm map at a single point, we combine the logarithm map with rolling maps [93–95] in this chapter. Rolling maps can be used to flatten the Lie group $SO(3) \otimes \ldots \otimes SO(3)$ by unwrapping the action curves onto its Lie algebra using the logarithm map while rolling. Figure 4.1 (right) illustrates the effect of unwrapping (using the logarithm map) while rolling with the example of a sphere. When rolled along the middle longitudinal curve, referred to as the nominal curve in the figure, the other curves that are close to the nominal curve on the sphere remain close to it even after unwrapping onto the tangent space at P.

Though the rolling map is a mathematically well-defined concept, it has not been explored much by the computer vision community. Recently, Caseiro *et. al.* [96] introduced the rolling map to the vision community by using it for the classification of manifold features. In [96], the Grassmann manifold was first rolled as a rigid body over the tangent space at identity, and the data samples were unwrapped onto the tangent space. Then, classification was performed in the tangent space. Rolling maps have also been used for interpolation on $SO(3)$ [97,98] and Grassmann manifold [99].

In this chapter, we first compute a nominal curve for each action category in $SO(3) \otimes \ldots \otimes SO(3)$, and warp all the action curves to these nominal curves using DTW. This helps us to handle the rate variations. Then, we roll $SO(3) \otimes \ldots \otimes SO(3)$ (by rolling each $SO(3)$ individually) over its Lie algebra $\mathfrak{so}(3) \oplus \ldots \oplus \mathfrak{so}(3)$ along the nominal curves, and unwrap all the action curves (using the logarithm map) onto the Lie algebra while rolling. The main advantage of unwrapping while rolling is that the distances between the action curves and the nominal curves are preserved while mapping the curves from the Lie group to the Lie algebra. Finally, we represent the unwrapped Lie algebra curves either by directly concatenating the temporal samples into a single feature vector or by using the FTP representation of [13], and classify them using a one-vs-all linear SVM classifier. Experimental results show that flattening by unwrapping while rolling improves the recognition performance when compared to flattening by using the logarithm map at a single point.

In most of the existing works that use rolling maps, the rolling curve was chosen as a geodesic curve [96–98]. But, we are interested in rolling $SO(3)$ along the nominal action curves, which are non-geodesic. While [94,96–98] provide closed form expressions for the rolling map when the rolling curve is a geodesic, they do not explain how to compute the rolling map in closed form when the rolling curve is non-geodesic. In this chapter, we show how to obtain a piecewise smooth rolling map for a given (discrete) non-geodesic rolling curve in $SO(3)$. Specifically, we derive closed form expressions for a rolling map such that the rolling curve passes through a given set of points in $SO(3)$ at given instances of time.

Contributions: We combine the logarithm and rolling maps to flatten the special orthogonal group $SO(3)$ for performing human action recognition from 3D skeletal data. The rolling map is a mathematically well-defined concept that has not been explored much by the vision community. To the best of our knowledge, it was never used in the context of human action recognition. Most existing works on rolling maps use a geodesic curve as the rolling curve. They do not provide closed form expressions for the rolling map in the case of a non-geodesic rolling curve. In this chapter, we show how to compute a piecewise smooth rolling map corresponding to a given (discrete) non-geodesic rolling curve in $SO(3)$.

Organization: Section 4.2 provides relevant background information on group $SO(3)^2\mathcal{R}^9$ and rolling maps. Section 4.3 presents the rolling and unwrapping operations for $SO(3)$ and Section 4.4 presents the proposed action recognition approach. Section 4.5 presents the experimental results and Section 4.6 concludes the chapter.

4.2 Relevant Background

4.2.1 Group $\mathbf{SO(3)^2}\mathcal{R}^\mathbf{9}$

The group $SO(3)^2\mathcal{R}^9$ is the set of all matrix triplets (U, V, X), where $X \in \mathcal{R}^{3\times3}$ and $U, V \in SO(3)$. The group multiplication and group inversion operations for this group are defined as

$$(U_2, V_2, X_2) \star (U_1, V_1, X_1) = (U_2U_1, V_2V_1, U_2X_1V_2^\top + X_2),$$

$$(U, V, X)^{-1} = (U^\top, V^\top, -U^\top XV), \tag{4.1}$$

and the group identity element is given by $(\boldsymbol{I}_3, \boldsymbol{I}_3, \mathbf{0})$. The group $SO(3)^2\mathcal{R}^9$ acts on $\mathcal{R}^{3\times3}$ via

$$SO(3)^2\mathcal{R}^9 \circ \mathcal{R}^{3\times3} \to \mathcal{R}^{3\times3}, \quad (U, V, X) \circ Z = UZV^\top + X. \tag{4.2}$$

4.2.2 Rolling Motion

For two m-dimensional Riemannian manifolds $\mathcal{M}$ and $\bar{\mathcal{M}}$, both embedded in the same ambient Euclidean space $\mathcal{R}^n$ $(n \geq m)$, the rolling motion describes how $\mathcal{M}$ rolls over $\bar{\mathcal{M}}$ as a rigid body without slip and twist. A classical example of such a motion is the rolling of 2-dimensional sphere over the tangent plane at a point as shown in figure 4.2.

The curve $\{\alpha(t) \in \mathcal{M} \subset \mathcal{R}^n : t \in [0, T]\}$ along which the manifold $\mathcal{M}$ rolls is called the rolling curve and the curve $\{\bar{\alpha}(t) \in \bar{\mathcal{M}} \subset \mathcal{R}^n : t \in [0, T]\}$, where the rolling curve touches the manifold $\bar{\mathcal{M}}$ while rolling, is called the development curve

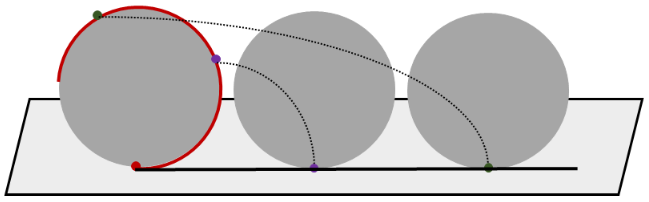

Figure 4.2: Sphere rolling over a tangent plane. The red curve is the rolling curve $\alpha(t)$ and the black curve is the development curve $\bar{\alpha}(t)$.

of α on $\bar{\mathcal{M}}$. In figure 4.2, the red curve on the sphere is the rolling curve and the black curve in the tangent space is the development curve. Since rolling is a rigid body motion in the ambient space $\mathcal{R}^n$, it can be mathematically described using a curve in the special Euclidean group $SE(n)$.

Definition 4.1 *A rolling map describing how $\mathcal{M}$ rolls over $\bar{\mathcal{M}}$, without slip and twist, along a smooth rolling curve $\alpha : [0,T] \to \mathcal{M}$, is a smooth map*

$$h : [0,T] \to SE(n),\ t \to h(t) = (\boldsymbol{R}(t), \boldsymbol{d}(t)), \tag{4.3}$$

satisfying the following conditions [94, 95]:

- *Rolling conditions*

$$\begin{aligned} \bar{\alpha}(t) &:= h(t) \circ \alpha(t) \in \bar{\mathcal{M}}, \\ T_{h(t)\circ\alpha(t)}(h(t) \circ \mathcal{M}) &= T_{\bar{\alpha}(t)}\bar{\mathcal{M}}, \end{aligned} \tag{4.4}$$

- *No-slip conditions*

$$(\dot{h}(t) \circ h(t)^{-1}) \circ \bar{\alpha}(t) = 0, \tag{4.5}$$

- *No-twist conditions*

$$\begin{aligned} (\dot{h}(t) \circ h(t)^{-1}) \circ T_{\bar{\alpha}(t)}\bar{\mathcal{M}} &\subset (T_{\bar{\alpha}(t)}\bar{\mathcal{M}})^{\perp}, \\ (\dot{h}(t) \circ h(t)^{-1}) \circ (T_{\bar{\alpha}(t)}\bar{\mathcal{M}})^{\perp} &\subset T_{\bar{\alpha}(t)}\bar{\mathcal{M}}, \end{aligned} \tag{4.6}$$

where for a point $x \in \mathcal{R}^n$ and a vector $\eta \in \mathcal{R}^n$ (i.e., there exists a curve $y : (-\epsilon, \epsilon) \to \mathcal{R}^n$ such that $\dot{y}(0) = \eta$), the operations $h(t) \circ x$, $\dot{h}(t) \circ x$, $(\dot{h}(t) \circ h(t)^{-1}) \circ x$ and $(\dot{h}(t) \circ h(t)^{-1}) \circ \eta$ are defined as

$$\begin{aligned} h(t) \circ x &:= \boldsymbol{R}(t)x + \boldsymbol{d}(t), \\ \dot{h}(t) \circ x &:= \frac{d}{ds}(h(s) \circ x)|_{s=t}, \\ (\dot{h}(t) \circ h(t)^{-1}) \circ x &:= \frac{d}{ds}((h(s)h(t)^{-1}) \circ x)|_{s=t}, \\ (\dot{h}(t) \circ h(t)^{-1}) \circ \eta &:= \frac{d}{ds}((\dot{h}(t) \circ h(t)^{-1}) \circ y(s))|_{s=t}. \end{aligned} \tag{4.7}$$

Result 4.1 *Given any piecewise smooth development or rolling curve, this definition ensures the existence and uniqueness of the corresponding rolling map [94, 95].*

4.3 Rolling Special Orthogonal Group

Here, we are interested in rolling $SO(3)$ over the tangent plane $T_{\boldsymbol{R}_0}SO(3)$ at a point $\boldsymbol{R}_0 \in SO(3)$. Note that both $SO(3)$ and $T_{\boldsymbol{R}_0}SO(3)$ are 3-dimensional manifolds embedded in the 9-dimensional Euclidean space $\mathcal{R}^{3\times3}$. Hence, we can describe the rolling of $SO(3)$ using a curve $h(t) \in SE(9)$. However, in [94], it has been shown that for rolling $SO(3)$ over a tangent plane, the rotational and translational components of the original special Euclidean group $SE(9)$ turn out to be $SO(3)^2$ and $\mathcal{R}^{3\times3}$, respectively. Therefore, the rolling map can be represented using a curve $c(t) \in SO(3)^2\mathcal{R}^9$.

Result 4.2 ***Rolling maps for*** $\mathbf{SO(3)}$***:*** *Let* $\{\boldsymbol{\Omega}(t) \in \mathfrak{so}(3) \mid t \in [0,T]\}$ *be any continuous curve. Let* $c(t) = (\boldsymbol{U}(t), \boldsymbol{V}(t), \boldsymbol{X}(t)) \in SO(3)^2\mathcal{R}^9$ *be the solution of*

$$\frac{d\boldsymbol{X}(t)}{dt} = \boldsymbol{\Omega}(t)\boldsymbol{R}_0, \qquad \frac{d\boldsymbol{U}(t)}{dt} = -\frac{1}{2}\boldsymbol{\Omega}(t)\boldsymbol{U}(t), \qquad \frac{d\boldsymbol{V}(t)}{dt} = \frac{1}{2}\boldsymbol{R}_0^\top\boldsymbol{\Omega}(t)\boldsymbol{R}_0\boldsymbol{V}(t), \quad (4.8)$$

satisfying $c(0) = (\boldsymbol{I}_3, \boldsymbol{I}_3, \boldsymbol{0})$*. Then, the action of* $c(t)$ *on* $SO(3) \subset \mathcal{R}^{3\times 3}$ *results in rolling of* $SO(3)$ *over the tangent plane* $T_{\boldsymbol{R}_0}SO(3)$ *with the rolling and development curves given by*

$$\begin{aligned} \alpha(t) &= \boldsymbol{U}(t)^\top\boldsymbol{R}_0\boldsymbol{V}(t) \in SO(3), \\ \bar{\alpha}(t) &= c(t) \circ \alpha(t) = \boldsymbol{R}_0 + \boldsymbol{X}(t) \in T_{\boldsymbol{R}_0}SO(3). \end{aligned} \quad (4.9)$$

The above result says that every continuous curve $\boldsymbol{\Omega}(t)$ in the Lie algebra of $SO(3)$ defines a rolling map $c(t)$ through the set of differential equations (4.8). Please refer to [94] for the proof.

Rolling along a geodesic: If $\boldsymbol{\Omega}(t) = \boldsymbol{\Omega} = \mathbf{log}(\boldsymbol{R}_1\boldsymbol{R}_0^\top)$, then the solution to (4.8) is given by

$$\boldsymbol{U}(t) = \mathbf{e}^{-\frac{1}{2}t\boldsymbol{\Omega}}, \ \boldsymbol{V}(t) = \boldsymbol{R}_0^\top\mathbf{e}^{\frac{1}{2}t\boldsymbol{\Omega}}\boldsymbol{R}_0, \ \boldsymbol{X}(t) = t\boldsymbol{\Omega}\boldsymbol{R}_0. \quad (4.10)$$

In this case, the rolling curve

$$\alpha(t) = \boldsymbol{U}(t)^\top\boldsymbol{R}_0\boldsymbol{V}(t) = \mathbf{e}^{t\boldsymbol{\Omega}}\boldsymbol{R}_0 = \mathbf{e}^{t\ \mathbf{log}(\boldsymbol{R}_1\boldsymbol{R}_0^\top)}\boldsymbol{R}_0 \quad (4.11)$$

is the geodesic from $\boldsymbol{R}_0$ to $\boldsymbol{R}_1$, and the development curve is given by

$$\bar{\alpha}(t) = \boldsymbol{R}_0 + t\boldsymbol{\Omega}\boldsymbol{R}_0. \quad (4.12)$$

4.3.1 Rolling along a Non-geodesic Curve

Note that Result 4.2 starts with a curve $\boldsymbol{\Omega}(t) \in \mathfrak{so}(3)$ and explains how to obtain the corresponding rolling map $c(t)$ and rolling curve $\alpha(t)$. It doesn't say anything about how to compute the rolling map $c(t)$ starting from a rolling curve $\alpha(t)$. But, in this chapter, we are interested in rolling $SO(3)$ along specific $\alpha(t)$, which are the nominal action curves obtained using DTW. If the rolling curve $\alpha(t)$ is a geodesic, then the corresponding rolling map $c(t)$ can be computed using (4.10). But, the nominal action curves along which we want to roll are usually non-geodesic.

Let $\{\boldsymbol{R}_0, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_T\}$ be the discrete representation of the curve along which we want to roll $SO(3)$. In Theorem 4.1, we show how to obtain a piecewise smooth rolling map $c(t)$ such that the corresponding rolling curve $\alpha(t)$ passes through $\boldsymbol{R}_t$ at time instance t for $t = 0, 1, \ldots, T$.

Theorem 4.1 *Let $\{\boldsymbol{R}_0, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_T\}$ be the given (discrete) rolling curve. Let $\boldsymbol{\Omega}_1$, $\boldsymbol{\Omega}_2$, ..., $\boldsymbol{\Omega}_T$ be T skew-symmetric matrices defined recursively using*

$$\boldsymbol{\Omega}_n = \boldsymbol{log}\left(\boldsymbol{e}^{-\frac{\boldsymbol{\Omega}_{n-1}}{2}} \ldots \boldsymbol{e}^{-\frac{\boldsymbol{\Omega}_1}{2}} \boldsymbol{R}_n \boldsymbol{R}_o^\top \boldsymbol{e}^{-\frac{\boldsymbol{\Omega}_1}{2}} \boldsymbol{e}^{-\frac{\boldsymbol{\Omega}_{n-1}}{2}}\right). \tag{4.13}$$

Let $c(t) = (\boldsymbol{U}(t), \boldsymbol{V}(t), \boldsymbol{X}(t))$ be a curve defined as

$$\boldsymbol{U}(t) = \boldsymbol{e}^{-\frac{(t-n+1)\boldsymbol{\Omega}_n}{2}} \boldsymbol{e}^{-\frac{\boldsymbol{\Omega}_{n-1}}{2}} \ldots \boldsymbol{e}^{-\frac{\boldsymbol{\Omega}_1}{2}}, \qquad \boldsymbol{V}(t) = \boldsymbol{R}_0^\top \boldsymbol{e}^{\frac{(t-n+1)\boldsymbol{\Omega}_n}{2}} \boldsymbol{e}^{\frac{\boldsymbol{\Omega}_{n-1}}{2}} \ldots \boldsymbol{e}^{\frac{\boldsymbol{\Omega}_1}{2}} \boldsymbol{R}_0,$$
$$\boldsymbol{X}(t) = \sum_{i=1}^{n-1} \boldsymbol{\Omega}_i \boldsymbol{R}_0 + (t-n+1)\boldsymbol{\Omega}_n \boldsymbol{R}_0, \qquad t \in [n-1, n], \quad n = 1, 2, \ldots, T. \tag{4.14}$$

Then, the action of $c(t) \in SO(3)^2\mathcal{R}^9$ on $SO(3)$ results in rolling of $SO(3)$ over the *tangent plane $T_{\boldsymbol{R}_0}SO(3)$ with a rolling curve $\alpha(t)$ that satisfies*

$$\alpha(n) = \boldsymbol{R}_n, \quad \textit{for} \;\; n = 1, 2, \ldots, T. \tag{4.15}$$

Proof: Let $\{\boldsymbol{\Omega}(t) \in \mathfrak{so}(3) \mid t \in [0, T]\}$ be a curve defined as

$$\boldsymbol{\Omega}(t) = 6\boldsymbol{\Omega}_n \left((t-n+1) - (t-n+1)^2\right), \quad t \in [n-1, n], \quad n = 1, 2, \ldots, T. \tag{4.16}$$

For this $\boldsymbol{\Omega}(t)$, the solution for differential equations (4.8) is given by (4.14). Hence by Result 4.2, the action of $c(t)$ on $SO(3)$ results in rolling of $SO(3)$ over the tangent space $T_{\boldsymbol{R}_0}SO(3)$ with the rolling curve given by

$$\alpha(t) = \boldsymbol{U}(t)^\top \boldsymbol{R}_0 \boldsymbol{V}(t) = \mathbf{e}^{\frac{\boldsymbol{\Omega}_1}{2}} \ldots \mathbf{e}^{\frac{\boldsymbol{\Omega}_{n-1}}{2}} \mathbf{e}^{(t-n+1)\boldsymbol{\Omega}_n} \mathbf{e}^{\frac{\boldsymbol{\Omega}_{n-1}}{2}} \ldots \mathbf{e}^{\frac{\boldsymbol{\Omega}_1}{2}} \boldsymbol{R}_0, \tag{4.17}$$

$$t \in [n-1, n], \quad n = 1, 2, \ldots, T.$$

which satisfies

$$\alpha(n) = \mathbf{e}^{\frac{\boldsymbol{\Omega}_1}{2}} \ldots \mathbf{e}^{\frac{\boldsymbol{\Omega}_{n-1}}{2}} \mathbf{e}^{\boldsymbol{\Omega}_n} \mathbf{e}^{\frac{\boldsymbol{\Omega}_{n-1}}{2}} \ldots \mathbf{e}^{\frac{\boldsymbol{\Omega}_1}{2}} \boldsymbol{R}_0 = \boldsymbol{R}_n, \quad \text{for} \;\; n = 0, 1, \ldots, T. \tag{4.18}$$

The above equation follows directly from the definition of $\boldsymbol{\Omega}_n$ in (4.13). ■

4.3.2 Unwrapping while Rolling

Rolling maps can be used to flatten $SO(3)$ by unwrapping the action curves (while rolling) onto the tangent space at a point using the logarithm map. Figure 4.3 illustrates this pictorially. In this figure, the blue curve is unwrapped onto a tangent space while rolling along the red curve.

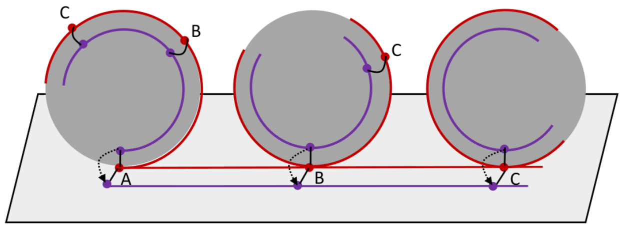

Figure 4.3: Unwrapping the blue curve while rolling along the red curve.

Let $c(t) = (\boldsymbol{U}(t), \boldsymbol{V}(t), \boldsymbol{X}(t)) \in SO(3)^2\mathcal{R}^9$ be the rolling map corresponding to the rolling curve $\alpha(t) \in SO(3)$. Let $\bar{\alpha}(t) \in T_{\alpha(0)}SO(3)$ be the development curve of $\alpha(t)$. Then, unwrapping (using the logarithm map) of a curve $\beta(t) \in SO(3)$ while rolling along $\alpha(t)$ gives the following curve $\bar{\beta}(t) \in T_{\alpha(0)}SO(3)$ [98]:

$$\begin{aligned}\bar{\beta}(t) &= \log_{SO(3)}(\alpha(0),\ c(t)o\beta(t) - \bar{\alpha}(t) + \alpha(0)) + \bar{\alpha}(t) \\ &= \log_{SO(3)}\left(\alpha(0),\ \boldsymbol{U}(t)\beta(t)\boldsymbol{V}(t)^\top\right) + \alpha(0) + \boldsymbol{X}(t).\end{aligned} \tag{4.19}$$

4.3.3 Advantage of Unwrapping while Rolling

The main motivation for using rolling maps in this chapter is that flattening of $SO(3)$ by unwrapping (using the logarithm map) the action curves while rolling is better than flattening it by using the logarithm map at a single point.

Theorem 4.2 *Let $\{\alpha(t),\ \beta(t) \in SO(3) : t \in [0, T]\}$ be two curves. Let $\bar{\alpha}(t), \bar{\beta}(t) \in T_{\alpha(0)}SO(3)$ respectively be the curves obtained by unwrapping (using the logarithm map) $\alpha(t)$ and $\beta(t)$ while rolling the $SO(3)$ over the tangent space at $\alpha(0)$ along the curve $\alpha(t)$. Then, we have*

$$d_{T_{\alpha(0)}SO(3)}\left(\bar{\beta}(t), \bar{\alpha}(t)\right) = d_{SO(3)}(\beta(t), \alpha(t)) \quad \forall t, \tag{4.20}$$

where $d_{SO(3)}$ represents the geodesic distance on $SO(3)$ and $d_{T_{\alpha(0)}SO(3)}$ represents the standard Euclidean distance in the tangent space $T_{\alpha(0)}SO(3)$.

Proof: Let $c(t) = (\boldsymbol{U}(t), \boldsymbol{V}(t), \boldsymbol{X}(t)) \in SO(3)^2\mathcal{R}^9$ be the rolling map corresponding to the rolling curve $\alpha(t)$. Then, by (4.19) we have

$$\bar{\beta}(t) = \log_{SO(3)}\left(\alpha(0),\ \boldsymbol{U}(t)\beta(t)\boldsymbol{V}(t)^\top\right) + \alpha(0) + \boldsymbol{X}(t). \tag{4.21}$$

Since $\alpha(t)$ is the rolling curve, $\bar{\alpha}(t) = \alpha(0) + \boldsymbol{X}(t)$ from Results 4.2. Hence, we have

$$\begin{aligned} d_{T_{\alpha(0)}SO(3)}\left(\bar{\beta}(t), \bar{\alpha}(t)\right) &= \|\bar{\beta}(t) - \bar{\alpha}(t)\|_{Fr} \\ &= \|\log_{SO(3)}\left(\alpha(0),\ \boldsymbol{U}(t)\beta(t)\boldsymbol{V}(t)^\top\right)\|_{Fr} \\ &= d_{SO(3)}\left(\alpha(0),\ \boldsymbol{U}(t)\beta(t)\boldsymbol{V}(t)^\top\right) \\ &= d_{SO(3)}\left(\boldsymbol{U}(t)^\top\alpha(0)\boldsymbol{V}(t),\ \beta(t)\right) = d_{SO(3)}\left(\alpha(t),\ \beta(t)\right). \end{aligned} \tag{4.22}$$

Here, the second last equality follows from the fact that $d_{SO(3)}$ is bi-invariant [100].■

As mentioned earlier, we first compute a nominal curve for each action category, and warp all the action curves to these nominal curves. Then, we roll the Lie group along the nominal curves and unwrap all the action curves onto the Lie algebra while rolling. As stated in the above theorem , the main advantage of flattening the action curves by unwrapping while rolling is that the distances between the action curves and the nominal curves are preserved. This is not the case with flattening using the logarithm map at a single point.

Alternative interpretation: The idea of *unwrapping while rolling along the nominal curve* can also be interpreted as the extension of the idea of *tangent plane* *mapping at the Karcher mean* from points to curves. When dealing with points, the Karcher mean is commonly used as the anchor point for tangent plane projection. Since we are dealing with curves rather than points, the Karcher mean is replaced by the mean/nominal curve. In the case of points, the logarithm map at Karcher mean is used to map the points to a common tangent space. Since we are dealing with curves (a curve can go through various points that are quite far apart), using the logarithm map at a single point to flatten entire curves is not a good idea because, as we move away from the anchor point (which will happen in the case of curves), the distortion due to the logarithm map increases. Instead, it is better to use the logarithm maps at multiple points spread over the nominal curve. This is exactly what we are doing while rolling and unwrapping.

4.4 Proposed Action Recognition Approach

Our human action recognition system consists of the following steps: (1) Skeletal representation, (2) Nominal curve computation using DTW, (3) Rolling and unwrapping, (4) Linear SVM classification (with concatenated or FTP representation).

Skeletal representation: We represent a 3D human skeleton using the relative 3D rotations between all pairs of body parts. Since 3D rotations are members of the Lie group $SO(3)$, our skeletal representation becomes a point in the Lie group $SO(3) \otimes \ldots \otimes SO(3)$.

Nominal curves: Using the above skeletal representation, we represent human actions as curves in the Lie group $SO(3) \otimes \ldots \otimes SO(3)$. During training, for each

Table 4.1: Algorithm for computing a nominal curve.

```
Input: Curves $\mathcal{S}_1(t), \ldots, \mathcal{S}_N(t)$ at $t = 0, 1, \ldots, T$.
Maximum number of iterations $max$ and threshold $\delta$.
Output: Nominal curve $\mathcal{S}(t)$ at $t = 0, 1, \ldots, T$.
Initialization: $\mathcal{S}(t) = \mathcal{S}_1(t)$, iter = 0.
while  iter < $max$
  Warp each curve $\mathcal{S}_j(t)$ to the nominal curve $\mathcal{S}(t)$ using DTW to get a warped
  curve $\mathcal{S}_j^w(t)$.
  Compute a new nominal $\mathcal{S}^{'}(t)$ using $\mathcal{S}^{'}(t) = \text{Karcher mean}\left(\{\mathcal{S}_i^w(t)\}_{i=1}^N\right)$
  if  $\sum_{t=0}^T$ Geodesic distance$(\mathcal{S}^{'}(t), \mathcal{S}(t)) \leq \delta$
      break
  end
  $\mathcal{S}(t) = \mathcal{S}^{'}(t)$;  iter = iter + 1;
end
```

action category, we compute a nominal curve using the algorithm summarized in Table 4.1, and warp all the curves to this nominal using dynamic time warping. This step helps in handling rate variations. For DTW computations, we use the squared Euclidean distance in the Lie algebra. We also performed DTW using the geodesic distance in $SO(3)$, but did not get any improvement in the final classification results. Hence, for faster computations, we use the Lie algebra distance in this chapter. Note that in order to compute the nominal curves, all the action curves must have same number of samples. For this, we use the interpolation algorithm presented in section 2.1.1 and re-sample the curves in $SO(3) \otimes \ldots \otimes SO(3)$. Interpolation on $SO(3) \otimes \ldots \otimes SO(3)$ is performed by simultaneously interpolating on individual $SO(3)$.

We note that the recently proposed transported square-root vector field [101] representation of curves, which is an extension of the earlier square-root velocity representation [102] to Riemannian manifolds, provides a distance metric that is invariant to temporal warping (i,.e., the distance between two curve does not change if both curves undergo the same temporal warping). Using this distance metric for DTW and nominal curve computations could further improve the performance.

Rolling and unwrapping: In this step, we roll the Lie group $SO(3) \otimes \ldots \otimes SO(3)$ over its lie algebra $\mathfrak{so}(3) \oplus \ldots \oplus \mathfrak{so}(3)$ (by rolling each $SO(3)$ individually over its Lie algebra) along each nominal action curve, and unwrap all the action curves onto the Lie algebra. The rolling map for a given (discrete) rolling curve can be obtained using Theorem 4.1 and the unwrapping operation can be performed using (4.19). Since a nominal action curve may not start from the identity element (remember that Lie algebra is the tangent space at the identity element), we first roll the Lie group from the identity element to the starting point of the nominal curve and then roll along the nominal curve.

SVM classification: In this step, we first represent the unwrapped action curves either by directly concatenating the temporal samples into a single feature vector or by using the FTP representation of [13], and then classify them using a one-vs-all linear SVM classifier. In the case of FTP representation, we apply FTP for each dimension separately and concatenate all the Fourier coefficients to obtain the final feature vector.

Figure 4.4 gives an overview of the proposed action recognition approach.

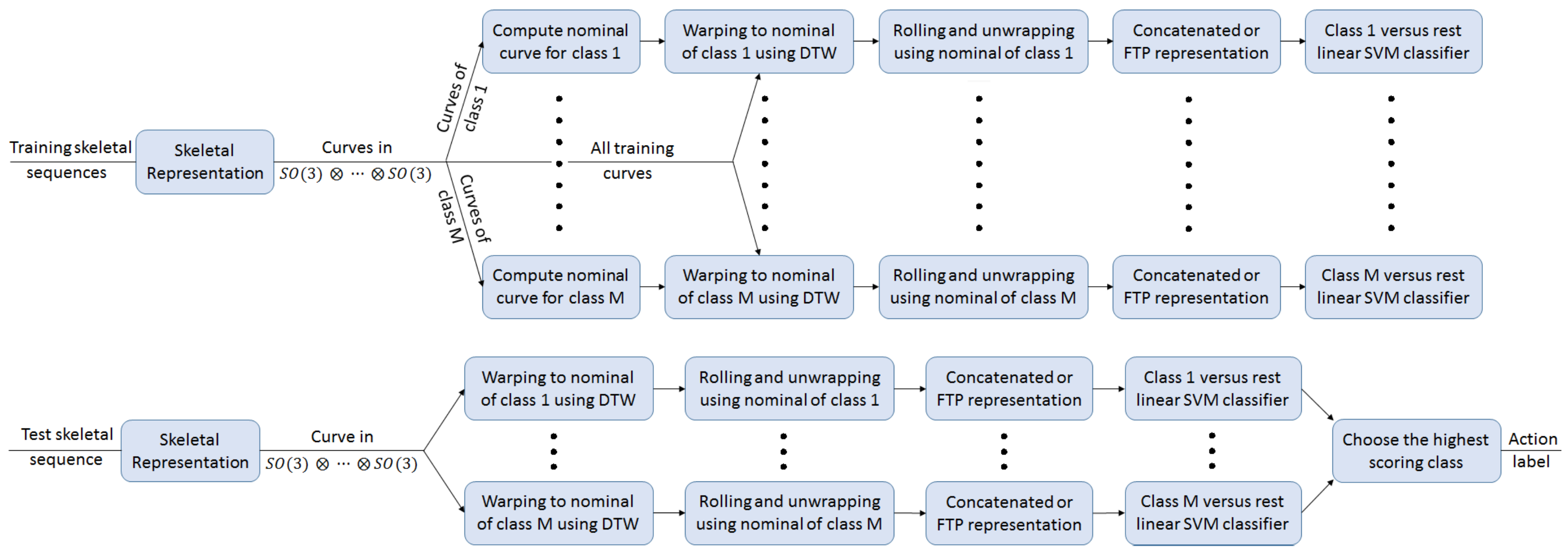
Training skeletal sequences
Skeletal Representation
Curves in $SO(3) \otimes \cdots \otimes SO(3)$
Curves of class 1
Curves of class M
All training curves
Compute nominal curve for class 1
Warping to nominal of class 1 using DTW
Rolling and unwrapping using nominal of class 1
Concatenated or FTP representation
Class 1 versus rest linear SVM classifier
Compute nominal curve for class M
Warping to nominal of class M using DTW
Rolling and unwrapping using nominal of class M
Concatenated or FTP representation
Class M versus rest linear SVM classifier
Test skeletal sequence
Skeletal Representation
Curve in $SO(3) \otimes \cdots \otimes SO(3)$
Warping to nominal of class 1 using DTW
Rolling and unwrapping using nominal of class 1
Concatenated or FTP representation
Class 1 versus rest linear SVM classifier
Warping to nominal of class M using DTW
Rolling and unwrapping using nominal of class M
Concatenated or FTP representation
Class M versus rest linear SVM classifier
Choose the highest scoring class
Action label

Figure 4.4: The proposed action recognition approach: Top row - Training, Bottom row - Testing.

4.5 Experimental Evaluation

In this section, we evaluate the proposed action recognition approach using three action datasets captured with Kinect sensor: Florence3D-Action [84], MSRAction Pairs [85] and G3D-Gaming [86].

Florence3D-Action [84] dataset consists of nine different daily actions like *drink water*, *answer phone*, *read watch*, *tight lace*, etc. performed by 10 different subjects. Each subject performed every action two or three times resulting in a total of 215 action sequences. The 3D locations of 15 joints are provided with the dataset.

MSRAction Pairs [85] dataset consists of six action pairs like *pick up a box/put down a box*, *wear a hat/take off a hat*, etc. performed by 10 different subjects. Each subject performed every action two or three times resulting in a total of 353 action sequences. The 3D locations of 20 joints are provided with the dataset.

G3D-Gaming [86] dataset consists of 20 different gaming actions like *golf swing*, *tennis serve*, *bowling*, *aim and fire gun*, etc. performed by 10 different subjects. Each subject performed every action three or more times resulting in a total of 663 action sequences. The 3D locations of 20 joints are provided with the dataset.

Evaluation setting: We followed the cross-subject test setting, in which half of the subjects were used for training and the other half were used for testing. All the results reported in this section were averaged over ten different random combinations of training and test subjects.

Table 4.2: Comparison between logarithm map at a point and rolling.

Dataset	Concatenated representation		FTP representation	
	Logarithm	Rolling	Logarithm	Rolling
Florence3D	86.83	**89.82**	90.89	**91.40**
MSRPairs	92.96	**94.09**	94.10	**94.67**
G3D	**87.89**	87.77	**91.48**	91.42

Basic pre-processing: To make the skeletal data invariant to the absolute location of human in the scene, all the 3D joint coordinates were transformed from world coordinate system to a person-centric coordinate system by placing the hip center at the origin. We rotated the skeletons such that the ground plane projection of the vector from left hip to right hip is parallel to the global x-axis.

Parameters: As explained in section 4.4, for each dataset, all the action curves were re-sampled to have same length. The reference length was chosen to be the maximum number of samples in any curve in the dataset before re-sampling. The value of SVM parameter C was chosen based on cross-validation. For the FTP representation, we used a three-level temporal pyramid with 1/4 length of each segment as low-frequency coefficients.

Unwrapping while rolling vs logarithm map: In this chapter, we are using rolling and unwrapping for flattening the Lie group $SO(3) \otimes \ldots \otimes SO(3)$. An alternative way to flatten this Lie group is to map the action curves to its Lie algebra using the logarithm map. Table 4.2 compares the performance of both these approaches in terms of action recognition accuracy when a linear SVM classifier is used with the concatenated and FTP representations.

Table 4.3: Comparison with the remaining R3DG features.

Dataset	$\mathfrak{se}(3)$	$\mathfrak{so}(3) \otimes \mathcal{R}^3$	$\mathcal{UQ} \otimes \mathcal{R}^3$	$\mathcal{UD}$	$\mathcal{UQ}$	$SO(3)$+ Rolling
Florence3D	90.71	90.87	**92.61**	91.55	91.27	91.40
MSRPairs	93.65	93.82	93.60	94.33	93.88	**94.67**
G3D	91.60	91.60	91.51	**92.12**	**92.12**	91.42

Note that the concatenated representation is nothing but the vectorized version of unwrapped curves. Hence, the results obtained using this representation directly compare the effects of using the logarithm map at a point and unwrapping while rolling. As we can see from Table 4.2, unwrapping while rolling outperforms the logarithm map by 3% on Florence3D dataset and by 1.1% on MSRPairs dataset. On G3D dataset, both rolling and logarithm map perform equally well. These results suggest that it is better to flatten $SO(3)$ by unwrapping while rolling instead of using the logarithm map at a point. When we use additional processing steps like the FTP representation, the performance gap decreases. This is probably because, by discarding the high frequency Fourier coefficients, the FTP representation is able to remove some of the distortions introduced by the logarithm map.

Comparison with the remaining R3DG features: Table 4.3 compares the performance of the proposed $SO(3)$-based approach with the remaining R3DG features introduced in Chapter 3. We use the DTW + FTP + linear SVM pipeline described in Chapter 3 with the other R3DG features. As we can see, while the proposed approach gives the best result on MSRPairs dataset, its accuracy is 1.2% and 0.7% less than the accuracy of best competing R3DG feature on Florence3D and G3D datasets, respectively.

4.6 Conclusions

In this chapter, we used the rolling maps for flattening $SO(3)$ to perform human action recognition from 3D skeletal data. We represented each human skeleton as a point in the Lie group $SO(3) \otimes \ldots \otimes SO(3)$ using the relative 3D rotations between all pairs of body parts. Using this skeletal representation, we represented human actions as curves in $SO(3) \otimes \ldots \otimes SO(3)$. For each action category, we computed a nominal curve and warped all the action curves to this nominal using DTW. Then, we rolled $SO(3) \otimes \ldots \otimes SO(3)$ over its Lie algebra along the nominal curves and unwrapped all the action curves onto the Lie algebra. Finally, we represented the unwrapped curves using either the concatenated representation or the FTP representation and classified them using a one-vs-all linear SVM classifier. By evaluating on three action datasets, we showed that flattening $SO(3)$ by unwrapping while rolling performs better than flattening $SO(3)$ by using logarithm map at a single point.

Note that in order to roll along the nominal curves, we should be able to compute the rolling map corresponding to a non-geodesic rolling curve. However, most of the existing works use a geodesic curve as the rolling curve and do not provide closed form expressions for the rolling map in the case of a non-geodesic rolling curve. In this chapter, we showed how to compute a piecewise smooth rolling map such that the rolling curve passes through a given set of points in $SO(3)$ at given instances of time.

Chapter 5: Kernel Learning for Extrinsic Classification of Manifold Features

5.1 Introduction

Many applications involving images and videos require classification of data that obey specific constraints. Such data often lie in non-Euclidean spaces. For instance, popular features in computer vision such as shapes [103], rotation matrices [35], linear subspaces [16], covariance features [17], etc., are known to lie on Riemannian manifolds. In such cases, one needs good classification techniques that make use of the underlying manifold structure.

For features that lie in Euclidean spaces, classifiers based on discriminative approaches such as Linear Discriminant Analysis (LDA), Partial Least Squares (PLS) and SVM have been successfully used in various applications. However, these approaches are not directly applicable to features that lie on Riemannian manifolds. Hence, classification is often performed in an extrinsic manner by first mapping the manifold to an Euclidean space, and then learning classifiers in the new space. One such popularly used Euclidean space is the tangent space at the mean sample [17, 104]. However, tangent spaces preserve only the local structure of the manifold and can often lead to sub-optimal performance. An alternative approach is to map the manifold to a reproducing kernel Hilbert space by using kernels [18, 20, 22–24, 105]. Though kernel-based methods have been successfully used in many computer vision applications, a poor choice of kernel can often result in reduced classification performance. This gives rise to an important question: *How to find good kernels for the classification of manifold features?*

In this chapter, we answer the above question using the kernel learning approach [106, 107], in which appropriate kernels are learned directly from the data. Since we are interested in learning good kernels for the purpose of classification, we jointly learn the kernel and the classifier by solving a single optimization problem. To learn a good kernel-classifier combination for features that lie on Riemannian manifolds, we propose the following two criteria: (i) Risk functional associated with the classifier in the mapped space should be minimized for good classification performance, (ii) The mapping should preserve the underlying manifold structure. The second criterion acts as a regularizer in learning the kernel. Our general framework for learning a good kernel-classifier combination can be represented as the following optimization problem

$$\min_{\boldsymbol{w},\mathcal{K}} \ \lambda \, \Gamma_s(\mathcal{K}) + \Gamma_c(\boldsymbol{w}, \mathcal{K}),$$

where $\Gamma_s(\mathcal{K})$, $\Gamma_c(\boldsymbol{w}, \mathcal{K})$ are respectively the manifold-structure and the classifier costs expressed as functions of classifier parameters $\boldsymbol{w}$ and kernel $\mathcal{K}$, and λ is a regularization parameter.

Due to its superior generalization properties, we focus on using the SVM classifier in this chapter. In order to preserve the manifold structure, we constrain the distances in the mapped space to be close to the manifold distances. Under this setting, we formulate the problem of learning a good kernel-classifier combination as a convex optimization problem. While the resulting formulation is an instance of SemiDefinite Programming (SDP) and can be solved using standard solvers such as SeDuMi [108], it is transductive in nature: both training and test data need to be present while learning the kernel matrix. Solving SDPs is also computationally expensive for large datasets. To solve both the issues, we follow the Multiple Kernel Learning (MKL) approach of [106, 107] and parameterize the kernel as a linear combination of known base kernels. This formulation results in a simpler convex optimization problem, that can be efficiently solved using gradient-based methods.

Organization: Section 5.2 provides a brief overview of existing approaches for the classification of manifold features. Section 5.3 briefly discuss the Riemannian geometry of two popularly-used manifold features, namely linear subspaces and covariance features, and Section 5.4 presents the proposed kernel learning-based extrinsic classification approach. Experimental results and conclusions are presented in Sections 5.5 and 5.6, respectively.

5.2 Related Work

Existing classification methods for manifold features can be broadly grouped into three main categories: nearest-neighbor methods, Bayesian methods, and Euclidean mapping-based methods.

Nearest neighbor: The simplest classifier on a manifold is the nearest-neighbor classifier based on some appropriately defined distance or similarity measure. In [109], the trajectories of human joint positions were represented as subspaces using LDS models, which were then classified using Martin and Finsler distances. In [104], LDS models were used to get subspace representations for shape deformations and the Frobenius distance was used for classification. In [19, 21, 110], image sets were modeled using linear subspaces, which were compared using the direct sum of canonical correlations in [19], a weighted sum of canonical correlations in [21], and the largest canonical correlation in [110].

Bayesian framework: Another possible approach for classification is to use the Bayesian framework by defining probability density functions (pdfs) on manifolds. In [16] parametric pdfs like Gaussian were defined on the tangent space and then wrapped back on to the manifold to define intrinsic pdfs for the Grassmann manifold. Alternatively, Parzen-window based non-parametric density estimation was used in [111] for the Stiefel manifold. Both these approaches along with the Bayes classifier were used for human activity recognition and video-based face recognition. In general, parametric approaches are sensitive to the model order, whereas the model-free non-parametric approaches are sensitive to the choice of window size.

Euclidean mapping: Discriminative approaches like LDA, PLS, SVM, Boosting, etc., can be extended to manifolds by mapping the manifolds to Euclidean spaces. One such Euclidean space is the tangent-space. In [17], a LogitBoost classifier was developed using weak classifiers learned on tangent spaces. This classifier was applied to the pedestrian detection task using covariance features. Alternatively, one can map manifolds to Euclidean spaces by defining Mercer kernels for manifolds. In [20,22], discriminant analysis was used for image set-based recognition tasks using Grassmann kernels. In [18], a kernel defined for the manifold of Symmetric Positive Definite (SPD) matrices was used with PLS for image set-based recognition tasks. In [105], Binet-Cauchy kernels defined for non-linear dynamical systems were used for human activity recognition. In general, the success of kernel-based methods is often determined by the choice of kernel. Hence, in this chapter, we address the issue of kernel-selection for the classification of manifold features.

The idea of using manifold structure as a regularizer was previously explored in the context of data manifolds [112, 113], where the given high dimensional data samples were simply assumed to lie on a lower dimensional manifold. Since the structure of the underlying manifold was unknown, a graph Laplacian-based empirical estimate of the data distribution was used in [112,113]. Contrary to this, in this chapter, we are interested in analytical manifolds such as the Grassmann manifold and the manifold of SPD matrices, whose underlying geometry is well-understood.

5.3 Relevant Background

In this section, we briefly discuss the Riemannian geometry of two representations that are popularly used in computer vision, namely linear subspaces and covariance features.

5.3.1 Linear Subspaces - Grassmann Manifold

Grassmann manifold, denoted by $\mathcal{G}_{n,d}$, is the set of all d-dimensional linear subspaces of $\mathcal{R}^n$. An element S of $\mathcal{G}_{n,d}$ can be represented by any $n \times d$ orthonormal matrix $\boldsymbol{Y}_S$ such that the column span of $\boldsymbol{Y}_S$ is the subspace S. The geodesic distance between two subspaces S_1 and S_2 on the Grassmann manifold is given by $\|\boldsymbol{\theta}\|_2$, where $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_d]$ are the principal angles between S_1 and S_2. The angles $\boldsymbol{\theta}$ can be computed using $\theta_i = \cos^{-1}(\alpha_i) \in [0, \frac{\pi}{2}]$, where α_i are the singular values of $\boldsymbol{Y}_{S1}^\top \boldsymbol{Y}_{S2}$. Other popularly-used distances for the Grassmann manifold are the Procrustes metric given by $2(\sum_{i=1}^d \sin^2(\theta_i/2))^{1/2}$, and the Projection metric given by $(\sum_{i=1}^d \sin^2\theta_i)^{1/2}$. We refer the interested readers to [114, 115] for further discussions on the Grassmann manifold.

Grassmann kernels: Grassmann manifold can be mapped to Euclidean spaces by using Mercer kernels. One popularly-used kernel [18, 20, 22] is the Projection kernel given by $\mathcal{K}_P(\boldsymbol{Y}_1, \boldsymbol{Y}_2) = \|\boldsymbol{Y}_1{}^\top \boldsymbol{Y}_2\|_{Fr}^2$. The feature mapping corresponding to the Projection kernel is given by $\Phi_P(\boldsymbol{Y}) = \boldsymbol{Y}\boldsymbol{Y}^\top$. Various kernels can be generated from $\mathcal{K}_P$ and Φ_P using

$$\begin{aligned} &\mathcal{K}_P^{\text{rbf}}(\boldsymbol{Y}_1, \boldsymbol{Y}_2) = \exp\left(-\gamma \|\Phi_P(\boldsymbol{Y}_1) - \Phi_P(\boldsymbol{Y}_2)\|_{Fr}^2\right), \\ &\mathcal{K}_P^{\text{poly}}(\boldsymbol{Y}_1, \boldsymbol{Y}_2) = \left(\gamma \mathcal{K}_P(\boldsymbol{Y}_1, \boldsymbol{Y}_2)\right)^d. \end{aligned} \tag{5.1}$$

We refer to the family of kernels K_P^{rbf} as projection-RBF kernels and the family of kernels K_P^{poly} as projection-polynomial kernels.

5.3.2 Covariance Features - SPD Manifold

The $d \times d$ SPD matrices, i.e., full-rank covariance matrices, form a Riemannian manifold [116], and the resulting Affine-Invariant Geodesic Distance (AIGD) is given by $(\sum_{i=1}^{d} \ln^2 \lambda_i(\boldsymbol{C}_1, \boldsymbol{C}_2))^{1/2}$, where $\lambda_i(\boldsymbol{C}_1, \boldsymbol{C}_2)$ are the generalized Eigenvalues of matrices $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$, and ln denotes the natural logarithm. Another popularly-used distance for SPD matrices is the Log-Euclidean Distance (LED) [117] given by $\|\mathbf{log}(\boldsymbol{C}_1) - \mathbf{log}(\boldsymbol{C}_2)\|_{Fr}$. We refer the readers to [116, 117] for further details.

Kernels for SPD matrices: Similar to the Grassmann manifold, we can define kernels for the set of SPD matrices. One such kernel based on the log-Euclidean distance was derived in [18]: $\mathcal{K}_{log}(\boldsymbol{C}_1, \boldsymbol{C}_2) = \text{trace}[\mathbf{log}(\boldsymbol{C}_1)^\top \mathbf{log}(\boldsymbol{C}_2)]$. The mapping corresponding to $\mathcal{K}_{log}$ is given by $\Phi_{log}(\boldsymbol{C}) = \mathbf{log}(\boldsymbol{C})$. Various kernels can be generated from $\mathcal{K}_{log}$ and Φ_{log} using

$$\begin{aligned} \mathcal{K}_{log}^{\text{rbf}}(\boldsymbol{C}_1, \boldsymbol{C}_2) &= \exp\left(-\gamma \|\Phi_{log}(\boldsymbol{C}_1) - \Phi_{log}(\boldsymbol{C}_2)\|_{Fr}^2\right), \\ \mathcal{K}_{log}^{\text{poly}}(\boldsymbol{C}_1, \boldsymbol{C}_2) &= \left(\gamma \mathcal{K}_{log}(\boldsymbol{C}_1, \boldsymbol{C}_2)\right)^d. \end{aligned} \tag{5.2}$$

We refer to the family of kernels K_{log}^{rbf} as LED-RBF kernels and the family of kernels K_{log}^{poly} as LED-polynomial kernels.

5.4 Extrinsic Support Vector Machines

Let $\mathcal{M}$ denote the underlying Riemannian manifold. Let $\{(x_i, y_i)\}_{i=1}^{N_{tr}}$ be the set of training samples where $x_i \in \mathcal{M}$, $y_i \in \{+1, -1\}$, and $\{x_i\}_{i=N_{tr}+1}^{N}$ be the set of test samples. Let Φ be the mapping to be learned from the manifold $\mathcal{M}$ to some inner product space $\mathcal{H}$. Let $\mathcal{K}(\cdot,\cdot)$ be the associated kernel function, and

$$\boldsymbol{K} = \begin{pmatrix} \boldsymbol{K}_{tr,tr} & \boldsymbol{K}_{tr,te} \\ \boldsymbol{K}_{te,tr} & \boldsymbol{K}_{te,te} \end{pmatrix} \in \mathcal{R}^{N \times N} \tag{5.3}$$

be the associated kernel matrix, with $\boldsymbol{K}_{ij} = \mathcal{K}(x_i, x_j) = \Phi(x_i)^\top \Phi(x_j), \forall x_i, x_j \in \mathcal{M}$.

Since we are interested in performing classification in the mapped space, we jointly learn the kernel and the classifier using a single optimization problem based on the following criteria:

- **Risk minimization:** For better classification performance, the risk functional associated with the classifier in the mapped space should be minimized.

- **Structure preservation:** Since the features lie on a Riemannian manifold with a well-defined structure, the mapping should be structure-preserving. This criterion can be seen as playing the role of a regularizer in kernel-learning.

Combining the above two criteria we formulate the problem of learning a good kernel-classifier combination as

$$\min_{\boldsymbol{w},\mathcal{K}} \ \lambda \ \Gamma_s(\mathcal{K}) + \Gamma_c(\boldsymbol{w}, \mathcal{K}), \tag{5.4}$$

where $\Gamma_s(\mathcal{K})$ and $\Gamma_c(\boldsymbol{w}, \mathcal{K})$ are respectively the manifold-structure cost and the classifier cost expressed as functions of classifier parameters $\boldsymbol{w}$ and kernel $\mathcal{K}$. Here, λ is the regularization parameter used to balance the two criteria. Since the mapped space is an inner product space, one can use standard machine learning techniques to perform classification. Due to its superior generalization properties, we focus on the SVM classifier in this chapter. However, it is important to note that the framework introduced here is general and can be applied to other classifiers as well.

SVM classifier in the mapped space: The SVM classifier in the mapped space is given by

$$f(x) = \boldsymbol{w}^{*\top}\Phi(x) + b^*, \tag{5.5}$$

where the weight vector $\boldsymbol{w}^*$ and the bias b^* are given by

$$\begin{aligned} \boldsymbol{w}^*, b^* = \underset{\boldsymbol{w},b,\eta}{\arg\min}\ & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N_{tr}} \eta_i \\ \text{subject to }\ & y_i(\boldsymbol{w}^\top\Phi(x_i) + b) \geq 1 - \eta_i,\ \eta_i \geq 0,\ i = 1, \ldots, N_{tr}. \end{aligned} \tag{5.6}$$

This problem is usually solved in its dual form

$$\max_{\boldsymbol{\alpha}\in\Omega} \left(\boldsymbol{\alpha}^\top\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top\left(\boldsymbol{y}\boldsymbol{y}^\top *\ \boldsymbol{K}_{tr,tr}\right)\boldsymbol{\alpha}\right), \tag{5.7}$$

where $\Omega = \{\boldsymbol{\alpha} \in \mathcal{R}^{N_{tr}} \mid \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1},\ \boldsymbol{\alpha}^\top\boldsymbol{y} = 0\}$, and $\boldsymbol{y}^\top = [y_1, \ldots, y_{N_{tr}}]$.

Preserving the manifold structure: To preserve the manifold structure, we constrain the distances in the mapped space to be close to the manifold distances. The squared Euclidean distance between two points x_i and x_j in the mapped space can be expressed in terms of kernel values as

$$\|\Phi(x_i) - \Phi(x_j)\|_2^2 = \boldsymbol{K}_{ii} + \boldsymbol{K}_{jj} - \boldsymbol{K}_{ij} - \boldsymbol{K}_{ji}. \tag{5.8}$$

Hence, we wish to minimize $\sum_{i=1}^{N}\sum_{j=i+1}^{N}\zeta_{ij}^2$, where

$$\zeta_{ij} = \boldsymbol{K}_{ii} + \boldsymbol{K}_{jj} - \boldsymbol{K}_{ij} - \boldsymbol{K}_{ji} - d_{ij}^2, \tag{5.9}$$

and d_{ij} is the manifold distance between the points x_i and x_j. Since ζ_{ij} can be positive or negative, we use ζ_{ij}^2 in the cost.

Combined formulation: Combining both the classifier and the structure costs, the joint optimization problem for learning a good kernel-classifier combination is given by

$$\begin{aligned}
&\min_{\boldsymbol{K}\succeq 0,\ \boldsymbol{\zeta}} \quad \max_{\boldsymbol{\alpha}\in\Omega} \quad \lambda\|\boldsymbol{\zeta}\|_2^2 + \left(\boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \left(\boldsymbol{y}\boldsymbol{y}^\top * \ \boldsymbol{K}_{tr,tr}\right)\boldsymbol{\alpha}\right), \\
&\text{subject to} \quad \sum_{i=1}^{N}\sum_{j=1}^{N} \boldsymbol{K}_{ij} = 0, \qquad\qquad (5.10) \\
&\qquad\qquad \boldsymbol{K}_{ii} + \boldsymbol{K}_{jj} - \boldsymbol{K}_{ij} - \boldsymbol{K}_{ji} - d_{ij}^2 = \zeta_{ij} \quad \text{for} \quad 1 \le i < j \le N,
\end{aligned}$$

where $\Omega = \{\boldsymbol{\alpha} \in \mathcal{R}^{N_{tr}} \mid \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1},\ \boldsymbol{\alpha}^\top \boldsymbol{y} = 0\}$, $\boldsymbol{\zeta}$ is the column vector of variables ζ_{ij} and $\boldsymbol{y} \in \mathcal{R}^{N_{tr}}$ is the column vector of class labels. The centering constraint $\sum_{ij} \boldsymbol{K}_{ij} = 0$ in (5.10) is added simply to remove the ambiguity associated with the origin in the mapped space [118]. Note that in (5.10) we are learning the entire kernel matrix $\boldsymbol{K}$ directly in a non-parametric fashion, and the classifier term has only $\boldsymbol{K}_{tr,tr}$. Therefore, to ensure meaningful values for $\boldsymbol{K}_{tr,te}$ and $\boldsymbol{K}_{te,te}$, we need additional constraints between the training and test samples [106]. For this, we use both the training and test samples in the structure-preserving constraints.

By following [106], it can be easily shown that the optimal $\boldsymbol{K}$ for (5.10) can be found by solving a semidefinite programming problem. SDPs are convex in nature and can be solved using standard solvers such as SeDuMi. Once the kernel matrix $\boldsymbol{K}$ is obtained, the SVM classifier in the mapped space can be obtained by solving the SVM dual problem (5.7). Note that the above formulation is transductive in nature: both training and test data need to be present while learning the kernel matrix. Also in general, solving SDPs can be computationally expensive for large datasets. Both these issues can be addressed by using the MKL approach.

5.4.1 Extrinsic SVM using MKL Framework

Instead of learning a non-parametric kernel matrix $\boldsymbol{K}$, following [107], we parameterize the kernel function $\mathcal{K}$ as a linear combination of fixed base kernels $\mathcal{K}^1, \mathcal{K}^2, ..., \mathcal{K}^M$:

$$\mathcal{K} = \sum_{m=1}^{M} \mu_m \mathcal{K}^m, \tag{5.11}$$

where $\boldsymbol{\mu}^\top = [\mu_1, \ldots, \mu_m]$ are positive weights to be learned. Since we use the same linear combination model for both training and test data, the weights $\boldsymbol{\mu}$ can be learned using only the training data, and the kernel values for test data can be computed using the known base kernels and the learned weights. Hence, the formulation becomes inductive. Under this linear combination model, the optimization problem (5.10) becomes

$$\begin{aligned} &\min_{\boldsymbol{\zeta},\ \boldsymbol{\mu}} \quad \max_{\boldsymbol{\alpha}\in\Omega} \lambda\|\boldsymbol{\zeta}\|_2^2 + \left(\boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top(\boldsymbol{y}\boldsymbol{y}^\top * \sum_{m=1}^{M}\mu_m \boldsymbol{K}^m_{tr,tr})\boldsymbol{\alpha}\right), \\ &\text{subject to} \quad \sum_{m=1}^{M}\mu_m(\boldsymbol{K}^m_{ii} + \boldsymbol{K}^m_{jj} - \boldsymbol{K}^m_{ij} - \boldsymbol{K}^m_{ji}) - d^2_{ij} = \zeta_{ij},\ \text{for } 1 \le i < j \le N_{tr}, \\ &\qquad\qquad \boldsymbol{\mu} \ge \mathbf{0}, \end{aligned} \tag{5.12}$$

where $\Omega = \{\boldsymbol{\alpha} \in \mathcal{R}^{N_{tr}} \mid \mathbf{0} \le \boldsymbol{\alpha} \le C\mathbf{1},\ \boldsymbol{\alpha}^\top\boldsymbol{y} = 0\}$. Note that the centering constraint $\sum_{i,j}\boldsymbol{K}_{ij} = 0$ in (5.10) is not required for the MKL approach as the origin is automatically decided based on the base kernels and their weights.

Let p^m_{ij} denote the squared distance between samples x_i and x_j induced by the base kernel $\mathcal{K}^m$, i.e., $p^m_{ij} = \boldsymbol{K}^m_{ii} + \boldsymbol{K}^m_{jj} - \boldsymbol{K}^m_{ij} - \boldsymbol{K}^m_{ji}$. Let $J_1(\boldsymbol{\mu})$ and $J_2(\boldsymbol{\mu})$ represent the manifold-structure cost and the classifier cost respectively in (5.12). Then,

$$\begin{aligned} J_1(\boldsymbol{\mu}) &= \sum_{i=1}^{N_{tr}} \sum_{j=i+1}^{N_{tr}} \zeta_{ij}^2 = \sum_{i=1}^{N_{tr}} \sum_{j=i+1}^{N_{tr}} \left(\sum_{m=1}^{M} \mu_m p_{ij}^m - d_{ij}^2 \right)^2, \\ J_2(\boldsymbol{\mu}) &= \max_{\boldsymbol{\alpha} \in \Omega} \left(\boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2} \boldsymbol{\alpha}^\top (\boldsymbol{y}\boldsymbol{y}^\top * \sum_{m=1}^{M} \mu_m \boldsymbol{K}_{tr,tr}^m) \boldsymbol{\alpha} \right). \end{aligned} \tag{5.13}$$

Let Φ_m be the mapping corresponding to the kernel $\mathcal{K}^m$ and $\ell_h(f)$ be the hinge loss function: $\ell_h(f) = \max(0, 1 - f)$.

Result 5.1 $J_2(\boldsymbol{\mu}) = J_3(\boldsymbol{\mu})$, *where*

$$J_3(\boldsymbol{\mu}) = \min_{\boldsymbol{V}_m,\ b} \ \frac{1}{2} \sum_{m=1}^{M} \frac{\|\boldsymbol{V}_m\|_2^2}{\mu_m} + C \sum_{i=1}^{N_{tr}} \ell_h \left(y_i \left(\sum_{m=1}^{M} \boldsymbol{V}_m^\top \Phi_m(x_i) + b \right) \right). \tag{5.14}$$

Please refer to [107] for the proof. Let $h(\boldsymbol{\mu}) = \lambda J_1(\boldsymbol{\mu}) + J_3(\boldsymbol{\mu})$. Then, using Result 5.1, the optimization problem (5.12) can be written as

$$\min_{\boldsymbol{\mu}} \ h(\boldsymbol{\mu}) \quad \text{subject to} \ \ \boldsymbol{\mu} \geq \mathbf{0}. \tag{5.15}$$

Theorem 5.1 *$h(\boldsymbol{\mu})$ is differentiable and convex if $\boldsymbol{K}_{tr,tr}^m \succ 0$ for $m = 1, ...M$.*

Proof: $J_1(\boldsymbol{\mu})$ is a convex quadratic term and hence differentiable with respect to $\boldsymbol{\mu}$. As shown in [107], $J_3(\boldsymbol{\mu})$ is also convex and differentiable if all the base kernel matrices $\boldsymbol{K}_{tr,tr}^m$ are strictly positive definite. Hence, $h(\boldsymbol{\mu})$ is a differentiable convex function of $\boldsymbol{\mu}$.

Using Theorem 5.1, the optimization problem (5.15) can be efficiently solved using the reduced gradient descent method [107] or any other standard algorithm used for solving constrained convex optimization problems. For any given $\boldsymbol{\mu}$, $J_1(\boldsymbol{\mu})$ can be evaluated directly using (5.13) and its gradient can be computed as

$$\frac{\partial J_1}{\partial \mu_m} = \sum_{i=1}^{N_{tr}} \sum_{j=i+1}^{N_{tr}} \left(2 p_{ij}^m \left(\sum_{k=1}^{M} \mu_k p_{ij}^k - d_{ij}^2 \right) \right). \tag{5.16}$$

Since $J_3(\boldsymbol{\mu}) = J_2(\boldsymbol{\mu})$, it can be computed by solving a standard SVM dual problem with $\boldsymbol{K}_{tr,tr} = \sum_{m=1}^{M} \mu_m \boldsymbol{K}^m_{tr,tr}$. The gradient of J_3 can be computed using [107]

$$\frac{\partial J_3}{\partial \mu_m} = -\frac{1}{2} \sum_{i=1}^{N_{tr}} \sum_{j=1}^{N_{tr}} \alpha_i^* \alpha_j^* y_i y_j \boldsymbol{K}^m_{ij}, \tag{5.17}$$

where $\boldsymbol{\alpha}^*$ is the optimal solution for the SVM dual problem used for computing $J_3(\boldsymbol{\mu})$. Once the optimal $\boldsymbol{\mu}^*$ is computed, the classifier in the mapped space can be obtained by solving the SVM dual problem (5.7) with $\boldsymbol{K}_{tr,tr} = \sum_{m=1}^{M} \mu_m^* \boldsymbol{K}^m_{tr,tr}$. Note that Theorem 5.1 requires the Gram matrices $\boldsymbol{K}^m_{tr,tr}$ to be positive definite. To enforce this property a small ridge may be added to their diagonals.

5.5 Experimental Evaluation

In this section, we evaluate the proposed extrinsic classification approach by applying it to image set-based face and object recognition tasks using two manifold features, namely linear subspaces and covariance features.

5.5.1 Recognition using Image Sets

Given multiple images of the same face or object, they can be collectively represented using a lower dimensional subspace spanned by the feature vectors representing individual images. Let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]$ be the mean-subtracted data matrix of an image set, where $\boldsymbol{x}_i \in \mathcal{R}^n$ is an n-dimensional feature descriptor of i-th image. Let $\boldsymbol{C} = \boldsymbol{X}\boldsymbol{X}^\top/N$ be the data covariance matrix. The linear subspace spanned by the top d Eigenvectors of $\boldsymbol{C}$ can be used to represent the image set by a d-dimensional linear subspace. Alternatively, the image set can also be represented using its natural second-order statistic [18], i.e., the covariance matrix $\boldsymbol{C}$. Since covariance matrices are positive semi-definite in general, we add a small ridge to their diagonals to make them positive definite.

5.5.2 Datasets and Feature Extraction

Face recognition – YouTube Celebrities [119]: This dataset has 1910 video clips of 47 subjects collected from the YouTube. Most of them are low resolution and highly compressed videos, making it a challenging dataset for face recognition. The face region in each image was extracted using a cascaded face detector, resized into 30×30 intensity image, and histogram equalized to eliminate lighting effects. Each video generated an image set of faces. Figure 5.1 shows some of the variations in an image set from this dataset.

Object recognition – ETH80 [120]: This dataset has images of eight object categories with each category containing ten different object instances. Each object instance has 41 images captured under different views, which form an image set. All the images were resized into 20×20 intensity images. Figure 5.2 shows typical variations in an image set from this dataset.

For both of these datasets, we performed experiments with two different manifold features: covariance matrices and linear subspaces. To avoid matrix singularity, we added a small ridge $\delta \boldsymbol{I}$ to each covariance matrix $\boldsymbol{C}$, where $\delta = 10^{-3} \times trace(\boldsymbol{C})$. For subspace representation, we used twenty dimensional linear subspaces spanned by the top twenty Eigenvectors of $\boldsymbol{C}$.

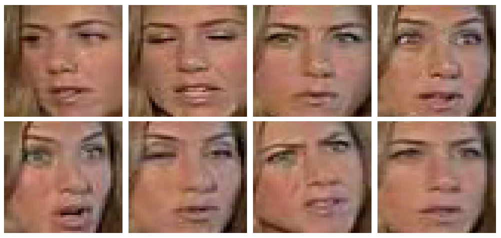

Figure 5.1: An image set from YouTube dataset.

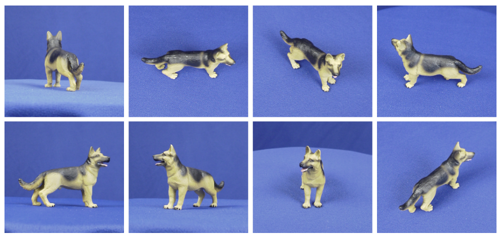

Figure 5.2: An image set from ETH80 dataset.

5.5.3 Comparative Methods and Evaluation Settings

We compare the proposed approach with the following methods:

- **Nearest neighbor(NN):** We used three different distances for the Grassmann manifold, namely the geodesic distance, the Procrustes distance and the Projection metric. We report the best results among the three. For covariance features, we used two distances, namely the AIGD and the LED and report the best results among the two.

- **Grassmann discriminant analysis (GDA) [20]:** Performs discriminant analysis followed by NN classification for the Grassmann manifold using the Projection kernel.

- **PLS with the Projection kernel (Proj+PLS) [18]:** Uses PLS combined with the Projection kernel for the Grassmann manifold.

- **Covariance discriminative learning (CDL) [18]:** Uses LDA and PLS for covariance features using a kernel derived from the LED metric.

- **Standard MKL (S-MKL) [107]:** In standard MKL, the kernel is learned as a convex combination of base kernels ($\mathcal{K} = \sum_{m=1}^{M} \mu_m \mathcal{K}^m,\ \boldsymbol{\mu} \geq \mathbf{0},\ \boldsymbol{\mu}^\top \mathbf{1} = 1$), by minimizing the SVM cost without manifold-based regularization.

Following [18], for the YouTube dataset, for each person, we used three randomly chosen image sets for training and six for testing, and for the ETH80 dataset, for each category, we used five randomly chosen image sets for training and five for testing. We report the recognition accuracy averaged over ten random trials. For GDA, Proj+PLS and CDL approaches, we report the recognition accuracy from [18].

5.5.4 Base Kernels and Parameters

For both the S-MKL and the proposed approaches, we used several base kernels. For experiments with linear subspaces, we used multiple projection-RBF and projection-polynomial kernels defined in (5.1). For each dataset, the values for the RBF parameter γ and the polynomial degree d were chosen based on their individual cross-validation accuracy on the training data. Specifically, for the YouTube dataset, we used ten projection-polynomial kernels and fifteen projection-RBF kernels, and for the ETH80 dataset, we used ten projection-polynomial kernels and thirteen projection-RBF kernels. The values for RBF kernel parameter γ were taken as $\frac{1}{n}2^{\delta}$, where n is the number of dimensions of Φ_P defined in Section 5.3.1, $\delta = \{-14, -12, \ldots, 12, 14\}$ for the YouTube dataset, and $\delta = \{-5, -3, \ldots, 17, 19\}$ for the ETH80 dataset. Polynomial kernels were generated by taking $\gamma = \frac{1}{n}$ and varying the degree from one to ten for both datasets.

For experiments with covariance features, we used the LED-RBF and LED-polynomial kernels defined in (5.2), whose parameters were chosen based on their individual cross-validation performance. Specifically, for the YouTube dataset, we used ten LED-polynomial kernels and fifteen LED-RBF kernels. For the ETH80 dataset, we used ten LED-polynomial kernels and twenty LED-RBF kernels. The values for the RBF parameter γ were taken as $\frac{1}{n}2^{\delta}$, where n is the number of dimensions of Φ_{log} defined in Section 5.3.2, $\delta = \{-7, -6, \ldots, 6, 7\}$ for the YouTube dataset, and $\delta = \{-10, -9, \ldots, 8, 9\}$ for the ETH80 dataset. For both datasets, polynomial kernels were generated by taking $\gamma = \frac{1}{n}$ and varying the degree from one to ten.

For both linear subspaces and covariance features, manifold geodesic distances were used in the distance preserving constraints. In all the experiments, the parameters for the S-MKL method (SVM parameter C) and the proposed approach (SVM parameter C and the regularization parameter λ) were chosen using cross-validation. For multi-class classification using SVM, we followed one-vs-all approach.

5.5.5 Results

Tables 5.1 and 5.2 show the recognition accuracy for YouTube and ETH80 datasets using linear subspaces and covariance features, respectively. We can see that the proposed approach clearly outperforms various existing approaches for the classification of manifold features. When compared to the NN baseline method, the proposed approach performs better with an average increase of 12.2% in the

Table 5.1: Recognition accuracy using linear subspaces.

dataset	NN	S-MKL [107]	GDA [20]	Proj + PLS [18]	Proposed approach
YouTube	62.8	64.3	65.7	67.7	**70.8**
ETH80	93.2	93.7	92.8	95.3	**96.0**

Table 5.2: Recognition accuracy using covariance features.

dataset	NN	S-MKL [107]	CDL-LDA [18]	CDL-PLS [18]	Proposed approach
YouTube	40.7	69.7	67.5	70.1	**73.2**
ETH80	92.7	93.7	94.5	96.5	**98.2**

recognition accuracy. This is expected as the simple NN-based classifier may not be powerful enough to handle the complex visual recognition tasks considered here. When compared to the S-MKL approach, the proposed approach performs better with an average increase of 4.2% in the recognition accuracy. This shows that the proposed manifold-based regularization is effective in finding a better kernel for classification. Recently, covariance features combined with PLS have been shown [18] to perform better than various other recent methods for image set-based recognition tasks. Our results show that the classification performance can be further improved by combining the covariance features with the proposed approach.

5.6 Conclusions

In this chapter, we introduced a general extrinsic framework for the classification of manifold features using kernel learning approach. We proposed two criteria for learning a good kernel-classifier combination for manifold features. In the case of SVM classifier, based on the proposed criteria, we formulated the problem of learning a good kernel-classifier combination as a convex optimization problem, and solved it efficiently following the multiple kernel learning approach. We evaluated the proposed approach for the image set-based classification task using linear subspaces and covariance features, and obtained superior performance compared to other relevant approaches.

Chapter 6: Deep Gaussian Conditional Random Field Network for Image Denoising

6.1 Introduction

In the recent past, deep networks have been successfully used in various image processing and computer vision applications [7, 28, 121]. Their success can be attributed to several factors such as their ability to represent complex input-output relationships, feed-forward nature of their inference (no need to solve an optimization problem during run time), availability of large training datasets, etc. One of the positive aspects of deep networks is that fairly general architectures composed of fully-connected or convolutional layers have been shown to work reasonably well across a wide range of applications. However, these general architectures do not use problem domain knowledge which could be very helpful in many applications.

For example, in the case of image denoising, it has been recently shown that conventional Multi-Layer Perceptrons (MLP) are not very good at handling multiple levels of input noise [28]. When a single MLP was trained to handle multiple input noise levels (by providing the noise variance as an additional input to the network), it produced inferior results compared to the widely-used Block-Matching and 3D filtering (BM3D) [122] approach. Contrary to this, the Expected Patch Log Likelihood (EPLL) framework of [123], which is a model-based approach, has been shown to work well across a range of noise levels. These results suggest that we should work towards bringing deep networks and model-based approaches together. Motivated by this, we propose a new deep network architecture for image denoising based on a Gaussian conditional random field model. The proposed network explicitly models the input noise variance and hence is capable of handling a range of noise levels.

Gaussian Markov Random Fields (MRFs) [9] are popular models for various structured inference tasks such as denoising, inpainting, super-resolution and depth estimation, as they model continuous quantities and can be efficiently solved using linear algebra routines. However, the performance of a Gaussian MRF model depends on the choice of pairwise potential functions. For example, in the case of image denoising, if the potential functions for neighboring pixels are homogeneous (i.e., identical everywhere), then the Gaussian MRF model can result in blurred edges and over-smoothed images. Therefore, to improve the performance of a Gaussian MRF model, the pairwise potential function parameters should be chosen according to the image being processed. A Gaussian MRF model that uses data-dependent potential function parameters is referred to as Gaussian conditional random field [124].

Image denoising using a Gaussian CRF model consists of two steps: a *parameter selection step* in which the potential function parameters are chosen based on the input image, and an *inference step* in which energy minimization is performed for the chosen parameters. In this chapter, we propose a novel model-based deep network architecture, which we refer to as *deep Gaussian CRF network*, by converting both the parameter selection and inference steps into feed-forward networks.

The proposed deep Gaussian CRF network consists of two sub-networks: a *Parameter Generation Network (PGNet)* that generates appropriate potential function parameters based on the input image, and an *Inference Network (InfNet)* that performs energy minimization using the potential function parameters generated by the PGNet. Since directly generating the potential function parameters for an entire image is very difficult (as the number of pixels could be very large), we construct a full-image pairwise potential function indirectly by combining the potential functions defined on image patches. If we use $d \times d$ patches, then our construction defines a graphical model in which each pixel is connected to its $(2d-1) \times (2d-1)$ spatial neighbors. This construction is motivated by the recent EPLL framework of [123]. Our PGNet directly operates on each $d \times d$ input image patch and chooses appropriate parameters for the corresponding potential function.

Though the energy minimizer can be obtained in closed form for a Gaussian CRF, it involves solving a linear system with number of variables equal to the number of image pixels (usually of the order of 10^6). Solving such a large linear system could be computationally prohibitive, especially for dense graphs (each pixel is connected to 224 neighbors when 8×8 image patches are used). Hence, we use an iterative optimization approach based on *Half Quadratic Splitting* (HQS) [123, 125–127] for designing our inference network. Recently, this approach has been shown to work very well for image restoration tasks even with a few iterations [123]. Our inference network consists of a new type of layer, which we refer to as the HQS layer, that performs the computations involved in a HQS iteration.

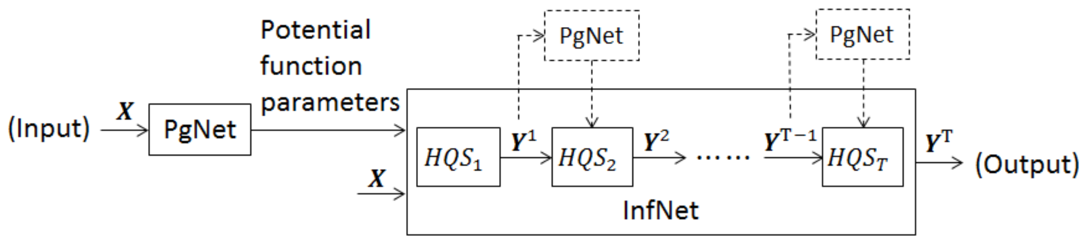

Figure 6.1: The proposed deep Gaussian CRF network: Parameter generation network followed by inference network. The PGNets in dotted boxes are the additional parameter generation networks introduced after each HQS iteration.

Combining the parameter generation and inference networks, we get our deep Gaussian CRF network shown in Figure 6.1. Note that using appropriate pairwise potential functions is crucial for the success of a Gaussian CRF model. Since PGNet operates on the noisy input image, it becomes increasingly difficult to generate good potential function parameters as the image noise increases. To address this issue, we introduce an additional PGNet after each HQS iteration as shown in dotted boxes in Figure 6.1. Since we train this deep Gaussian CRF network discriminatively in an end-to-end fashion, even if the first PGNet fails to generate good potential function parameters, the later PGNets can learn to generate appropriate parameters based on partially restored images.

Contributions: We propose a new end-to-end trainable deep network architecture for image denoising based on a Gaussian CRF model. Contrary to existing discriminative denoising methods that train a separate model for each noise level, the proposed network explicitly models the input noise variance and hence is capable of handling a range of noise levels. We propose a differentiable parameter generation network that generates the Gaussian CRF pairwise potential parameters based on the noisy input image. We unroll a half quadratic splitting-based iterative Gaussian CRF inference procedure into a deep network and train it jointly with our parameter generation network. We show that the proposed approach produces results on par with the state-of-the-art without training a separate network for each noise level.

Organization: Section 6.2 provides an overview of existing works on Gaussian CRFs, image denoising and inference unfolding. Section 6.3 presents the Gaussian CRF model used in this chapter, and Section 6.4 presents the proposed deep Gaussian CRF network. Experimental results and conclusions are presented in Sections 6.5 and 6.6, respectively.

6.2 Related Work

Gaussian CRFs were first introduced in [124] by modeling the parameters of the conditional distribution of output given input as a function of the input image. The precision matrix associated with each image patch was modeled as a linear combination of twelve derivative filter-based matrices. The combination weights were chosen as a parametric function of the responses of the input image to a set of oriented edge and bar filters, and the parameters were learned using discriminative training. This Gaussian CRF model was extended to Regression Tree Fields (RTFs) in [128], where regression trees were used for selecting the parameters of Gaussians defined over image patches. These regression trees used responses of the input image to various hand-chosen filters for selecting an appropriate leaf node for each image patch. This RTF-based model was trained by iteratively growing the regression trees and optimizing the Gaussian parameters at leaf nodes. Recently, a cascade of RTFs [129] has also been used for image restoration tasks. Contrary to the RTF-based approaches, all the components of our network are differentiable, and hence it can be trained end-to-end using standard gradient-based techniques.

Recently, [130] proposed a cascade of shrinkage fields for image restoration tasks. They learned a separate filter bank and shrinkage function for each stage of their cascade using discriminative training. Though this model can also be seen as a cascade of Gaussian CRFs, the filter banks and shrinkage functions used in the cascade do not depend on the noisy input image during test time. Contrary to this, the pairwise potential functions used in our Gaussian CRF model are generated by our PGNets based on the noisy input image.

Our approach is also related to the EPLL framework of [123], which decomposes the full-image Gaussian model into patch-based Gaussians, and uses HQS iterations for Gaussian CRF inference. Following are the main differences between EPLL and this work: (i) We propose a new deep network architecture which combines HQS iterations with a differentiable parameter generation network. (ii) While the EPLL chooses the potential parameters for each image patch as one of the K possible matrices, we construct each potential parameter matrix as a convex combination of K base matrices. (iii) While the EPLL learns the K possible potential parameter matrices in a generative fashion by fitting a Gaussian Mixture Model (GMM) to clean image patches, we learn the K base matrices in a discriminative fashion by the end-to-end training of our deep network. As shown later in the experiments section, our discriminative model clearly outperforms the generatively trained EPLL.

Gaussian CRFs have also been used recently for other applications such as depth estimation [131], facial landmark detection [132] and document retrieval [133].

Denoising: Image denoising is one of the oldest problems in image processing and various denoising algorithms have been proposed over the past several years. Some of the popular algorithms include fields of experts [1], BM3D [122], wavelet shrinkage [134], Gaussian scale mixtures [135], non-linear diffusion process-based approaches [136–138], sparse coding-based approaches [139–142], weighted nuclear norm minimization [143], and non-local Bayesian denoising [144]. Among these, BM3D is currently the most widely-used state-of-the-art denoising approach.

Denoising with neural networks: Recently, various deep neural network-based approaches have also been proposed for image denoising [27–29,145–147]. While [145] used a convolutional neural network, [28,146] used multilayer perceptrons, and [27, 29] used stacked sparse denoising autoencoders. Among these, the MLP [28] approach has been shown to work very well outperforming the BM3D approach. However, none of these deep networks explicitly model the input noise variance, and hence are not good at handling multiple noise levels. In all these works, a different network was trained for each noise level.

Unfolding inference as a deep network: The proposed approach is also related to a class of algorithms that discriminatively learn the model parameters by backpropagating the gradient through a fixed number of inference steps. In [148], the fields of experts [1] MRF model was discriminatively trained for image denoising by unfolding a fixed number of gradient descent inference steps. In [149], message-passing inference machines were trained for structured prediction tasks by considering the belief propagation-based inference of a discrete graphical model as a sequence of predictors. In [150], a feed-forward sparse code predictor was trained by unfolding a coordinate descent based sparse coding inference algorithm. In [151, 152], deep CNNs and discrete graphical models were jointly trained by unfolding the discrete mean-field inference. In [153], a new kind of non-negative deep network was introduced by deep unfolding of non-negative matrix factorization model. Recently, [136] revisited the classical non-linear diffusion process [154] by modeling it using several parameterized linear filters and influential functions. The parameters of this diffusion process were learned discriminatively by back-propagating the gradient through a fixed number of diffusion process iterations. Though this diffusion process-based approach has been shown to work well for the task of image denoising, it uses a separate model for each noise level.

In this chapter, we design our inference network using HQS-based inference of a Gaussian CRF model, resulting in a different network architecture compared to the above unfolding works. In addition to this inference network, our deep Gaussian CRF network also consists of other sub-networks used for modeling the Gaussian CRF pairwise potentials.

6.3 Gaussian Conditional Random Field Model

Let $\mathbf{X}$ be the given (noisy) input image and $\mathbf{Y}$ be the (clean) output image that needs to be inferred. Let $\mathbf{X}(i,j)$ and $\mathbf{Y}(i,j)$ represent the pixel (i,j) in images $\mathbf{X}$ and $\mathbf{Y}$, respectively. We model the conditional probability density $p(\mathbf{Y}|\mathbf{X})$ as a Gaussian distribution given by $p\left(\mathbf{Y}|\mathbf{X}\right) \propto \exp\left\{-E\left(\mathbf{Y}|\mathbf{X}\right)\right\}$, where

$$\begin{aligned} E\left(\mathbf{Y}|\mathbf{X}\right) = \frac{1}{2\sigma^2}\sum_{ij}\left[\mathbf{Y}(i,j) - \mathbf{X}(i,j)\right]^2 & \left.\vphantom{\sum_{ij}}\right\} \coloneqq E_d\left(\mathbf{Y}|\mathbf{X}\right) \\ + \frac{1}{2}\text{vector}(\mathbf{Y})^\top \mathbf{Q}(\mathbf{X})\text{vector}(\mathbf{Y}) & \left.\vphantom{\frac{1}{2}}\right\} \coloneqq E_p\left(\mathbf{Y}|\mathbf{X}\right). \end{aligned} \tag{6.1}$$

Here, σ^2 is the input noise variance and $\mathbf{Q}(\mathbf{X}) \succeq 0$ are the input-dependent parameters of the quadratic pairwise potential function $E_p\left(\mathbf{Y}|\mathbf{X}\right)$ defined over the image $\mathbf{Y}$. Note that if the pairwise potential parameters $\boldsymbol{Q}$ are constant, then this model can be interpreted as a generative model with E_d as the data term, E_p as the prior term and $p(\mathbf{Y}|\mathbf{X})$ as the posterior. Hence, our Gaussian CRF is a discriminative model inspired by a generative Gaussian model.

6.3.1 Patch-based Pairwise Potential Functions

Directly choosing the pairwise potential parameters $\mathbf{Q}(\mathbf{X})$ for an entire image $\mathbf{Y}$ is very challenging since the number of pixels in an image could be of the order of 10^6. Hence, motivated by [123], we construct the (full-image) pairwise potential function E_p by combining patch-based pairwise potential functions.

Let $\mathbf{x}_{ij}$ and $\mathbf{y}_{ij}$ be $d^2 \times 1$ column vectors representing the $d \times d$ patches centered on pixel (i,j) in images $\mathbf{X}$ and $\mathbf{Y}$, respectively. Let $\bar{\mathbf{x}}_{ij} = \mathbf{G}\mathbf{x}_{ij}$ and $\bar{\mathbf{y}}_{ij} = \mathbf{G}\mathbf{y}_{ij}$ be the mean-subtracted versions of vectors $\mathbf{x}_{ij}$ and $\mathbf{y}_{ij}$, respectively, where $\mathbf{G} = \mathbf{I}_{d^2} - \frac{1}{d^2}\mathbf{1}$ is the mean subtraction matrix. Let

$$V\left(\bar{\mathbf{y}}_{ij}|\bar{\mathbf{x}}_{ij}\right) = \frac{1}{2}\bar{\mathbf{y}}_{ij}^{\top}\left(\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\right)^{-1}\bar{\mathbf{y}}_{ij},\ \mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij}) \succeq 0, \tag{6.2}$$

be a quadratic pairwise potential function defined on patch $\bar{\mathbf{y}}_{ij}$, with $\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})$ being the corresponding (input) data-dependent parameters. Combining the patch-based potential functions at all the pixels, we get the following full-image pairwise potential function:

$$E_p\left(\mathbf{Y}|\mathbf{X}\right) = \frac{1}{d^2}\sum_{ij} V\left(\bar{\mathbf{y}}_{ij}|\bar{\mathbf{x}}_{ij}\right) = \frac{1}{2d^2}\sum_{ij}\mathbf{y}_{ij}^{\top}\mathbf{G}^{\top}\left(\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\right)^{-1}\mathbf{G}\mathbf{y}_{ij}. \tag{6.3}$$

Note that since we are using all $d \times d$ image patches, each pixel appears in d^2 patches that are centered on its $d \times d$ neighbor pixels. In every patch, each pixel interacts with all the d^2 pixels in that patch. This effectively defines a graphical model of neighborhood size $(2d-1) \times (2d-1)$ on image $\mathbf{Y}$.

6.3.2 Inference

Given the (input) data-dependent parameters $\{\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\}$ of the pairwise potential function $E_p\left(\mathbf{Y}|\mathbf{X}\right)$, the Gaussian CRF inference solves the following optimization problem:

$$\mathbf{Y}^* = \underset{\mathbf{Y}}{\operatorname{argmin}} \sum_{ij}\left(\frac{d^2}{\sigma^2}\left[\mathbf{Y}(i,j) - \mathbf{X}(i,j)\right]^2 + \mathbf{y}_{ij}^{\top}\mathbf{G}^{\top}\left(\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\right)^{-1}\mathbf{G}\mathbf{y}_{ij}\right). \tag{6.4}$$

Note that the optimization problem (6.4) is an unconstrained quadratic program and hence can be solved in closed form. However, the closed form solution for $\mathbf{Y}$ requires solving a linear system of equations with number of variables equal to the number of image pixels. Since solving such linear systems could be computationally prohibitive for large images, we use a half quadratic splitting-based iterative optimization method, that has been recently used in [123] for solving the above optimization problem. This approach allows for efficient optimization by introducing auxiliary variables.

Let $\mathbf{z}_{ij}$ be an auxiliary variable corresponding to the patch $\mathbf{y}_{ij}$. In half quadratic splitting method, the cost function in (6.4) is modified to

$$J(\mathbf{Y}, \{\mathbf{z}_{ij}\}, \beta) = \sum_{ij} \left\{ \begin{array}{l} \frac{d^2}{\sigma^2} \left[\mathbf{Y}(i,j) - \mathbf{X}(i,j)\right]^2 + \ \beta \|\mathbf{y}_{ij} - \mathbf{z}_{ij}\|_2^2 \\ \\ + \ \mathbf{z}_{ij}^\top \mathbf{G}^\top \left(\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\right)^{-1} \mathbf{G}\mathbf{z}_{ij} \end{array} \right\}. \tag{6.5}$$

Note that as $\beta \to \infty$, the patches $\{\mathbf{y}_{ij}\}$ are restricted to be equal to the auxiliary variables $\{\mathbf{z}_{ij}\}$, and the solutions of (6.4) and (6.5) converge.

For a fixed value of β, the cost function J can be minimized by alternatively optimizing for $\mathbf{Y}$ and $\{\mathbf{z}_{ij}\}$. If we fix $\mathbf{Y}$, then the optimal $\mathbf{z}_{ij}$ is given by

$$\begin{aligned} f(\mathbf{y}_{ij}) &= \underset{\mathbf{z}_{ij}}{\operatorname{argmin}} \ \left(\mathbf{z}_{ij}^\top \mathbf{G}^\top \left(\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\right)^{-1} \mathbf{G}\mathbf{z}_{ij} + \ \beta \|\mathbf{y}_{ij} - \mathbf{z}_{ij}\|_2^2\right) \\ &= \left(\mathbf{G}^\top \left(\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\right)^{-1} \mathbf{G} + \beta \mathbf{I}_{d^2}\right)^{-1} \beta \mathbf{y}_{ij} \\ &= \left(\mathbf{I}_{d^2} - \mathbf{G}^\top \left(\beta \mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij}) + \mathbf{G}\mathbf{G}^\top\right)^{-1} \mathbf{G}\right) \mathbf{y}_{ij}. \end{aligned} \tag{6.6}$$

The last equality in (6.6) follows from Woodbury matrix identity. If we fix $\{\mathbf{z}_{ij}\}$, then the optimal $\mathbf{Y}(i,j)$ is given by

$$\begin{aligned} g(\{\mathbf{z}_{ij}\}) &= \underset{\mathbf{Y}(i,j)}{\operatorname{argmin}} \ \left(\frac{d^2}{\sigma^2} \left[\mathbf{Y}(i,j) - \mathbf{X}(i,j)\right]^2 + \ \beta \sum_{p,q=-\lfloor \frac{d-1}{2} \rfloor}^{\lceil \frac{d-1}{2} \rceil} \left[\mathbf{Y}(i,j) - \mathbf{z}_{pq}(i,j)\right]^2 \right) \\ &= \frac{\mathbf{X}(i,j)}{1+\beta\sigma^2} + \frac{\beta\sigma^2}{(1+\beta\sigma^2)d^2} \sum_{p,q=-\lfloor \frac{d-1}{2} \rfloor}^{\lceil \frac{d-1}{2} \rceil} \mathbf{z}_{pq}(i,j), \end{aligned} \tag{6.7}$$

where $\lfloor \quad \rfloor, \lceil \quad \rceil$ are the floor and ceil operators, respectively, and $\mathbf{z}_{pq}(i,j)$ is the intensity value of pixel (i,j) according to the auxiliary patch $\mathbf{z}_{pq}$.

In half quadratic splitting approach, the optimization steps (6.6) and (6.7) are repeated while increasing the value of β in each iteration. This iterative approach has been shown to work well in [123] for image restorations tasks even with few (5-6) iterations.

6.4 Deep Gaussian CRF network

As mentioned earlier, the proposed deep Gaussian CRF network consists of the following two components:

- **Parameter generation network:** This network takes the noisy image $\mathbf{X}$ as input and generates the parameters $\{\boldsymbol{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\}$ of pairwise potential function $E_p(\mathbf{Y}|\mathbf{X})$.
- **Inference network:** This network performs Gaussian CRF inference using the pairwise potential parameters $\{\boldsymbol{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\}$ given by the parameter generation network.

6.4.1 Parameter Generation Network

We model the pairwise potential parameters $\{\boldsymbol{\Sigma}_{ij}\}$ as convex combinations of K symmetric positive semidefinite matrices $\boldsymbol{\Psi}_1, \ldots, \boldsymbol{\Psi}_K$:

$$\boldsymbol{\Sigma}_{ij} = \sum_k \gamma_{ij}^k \boldsymbol{\Psi}_k, \ \gamma_{ij}^k \geq 0, \ \sum_k \gamma_{ij}^k = 1. \tag{6.8}$$

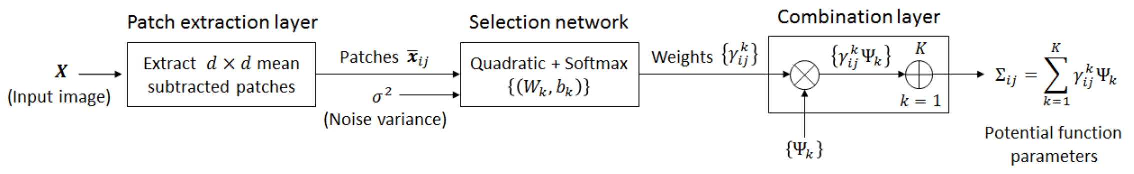
Patch extraction layer
Extract $d \times d$ mean subtracted patches
X
(Input image)
Patches $\bar{x}_{ij}$
σ^2
(Noise variance)
Selection network
Quadratic + Softmax $\{(W_k, b_k)\}$
Weights $\{\gamma_{ij}^k\}$
Combination layer
$\{\gamma_{ij}^k \Psi_k\}$
K
$k = 1$
$\{\Psi_k\}$
$\Sigma_{ij} = \sum_{k=1}^{K} \gamma_{ij}^k \Psi_k$
Potential function parameters

Figure 6.2: Parameter generation network: Mean subtracted patches $\bar{\mathbf{x}}_{ij}$ extracted from the input image $\mathbf{X}$ are used to compute the combination weights $\{\gamma_{ij}^k\}$, which are used for generating the pairwise potential parameters $\{\boldsymbol{\Sigma}_{ij}\}$.

The combination weights $\{\gamma_{ij}^k\}$ are computed from the mean-subtracted input image patches $\{\bar{\mathbf{x}}_{ij}\}$ using the following two layer selection network:

$$\begin{gathered}\text{Layer 1 - Quadratic layer : For } k = 1, 2, \ldots, K \\ s_{ij}^k = -\frac{1}{2}\bar{\mathbf{x}}_{ij}^\top \left(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2}\right)^{-1} \bar{\mathbf{x}}_{ij} + b_k.\end{gathered} \tag{6.9}$$

$$\begin{gathered}\text{Layer 2 - Softmax layer : For } k = 1, 2, \ldots, K \\ \gamma_{ij}^k = e^{s_{ij}^k} \Big/ \sum_{p=1}^{K} e^{s_{ij}^p}.\end{gathered} \tag{6.10}$$

Figure 6.2 shows the overall parameter generation network which includes a patch extraction layer, a selection network and a combination layer. Here, σ^2 is the noise variance, and $\{(\mathbf{W}_k \succeq 0, \boldsymbol{\Psi}_k \succeq 0, b_k)\}$ are the network parameters.

Our choice of the above quadratic selection function is motivated by the following two reasons: (i) Since the selection network operates on mean-subtracted patches, it should be symmetric, i.e., both $\bar{\mathbf{x}}$ and $-\bar{\mathbf{x}}$ should have the same combination weights $\{\gamma^k\}$. To achieve this, we compute each s^k as a quadratic function of $\bar{\mathbf{x}}$. (ii) Since we are computing the combination weights using the noisy image patches, the selection network should be robust to input noise. To achieve this, we include the input noise variance σ^2 in the computation of $\{s^k\}$. We choose the particular form $(\mathbf{W}_k + \sigma^2\mathbf{I}_{d^2})^{-1}$ because in this case, we can (roughly) interpret the computation of $\{s^k\}$ as evaluating the Gaussian log likelihoods. If we interpret $\{\mathbf{W}_k\}$ as covariance matrices associated with clean image patches, then $\{\mathbf{W}_k + \sigma^2\mathbf{I}_{d^2}\}$ can be interpreted as covariance matrices associated with noisy image patches.

6.4.2 Inference Network

We use the half quadratic splitting method described in Section 6.3.2 to create our inference network. Each layer of the inference network, also referred to as the HQS layer, implements one half quadratic splitting iteration. Each HQS layer consists of the following two sub-layers:

- **Patch inference layer (PI):** This layer uses the current image estimate $\mathbf{Y}^t$ and computes the auxiliary patches $\{\mathbf{z}_{ij}\}$ using $f(\mathbf{y}_{ij})$ given in (6.6).

- **Image formation layer (IF):** This layer uses the auxiliary patches $\{\mathbf{z}_{ij}\}$ given by the PI layer and computes the next image estimate $\mathbf{Y}^{t+1}$ using $g(\{\mathbf{z}_{ij}\})$ given in (6.7).

Let $\{\beta_1, \beta_2, \ldots, \beta_T\}$ be the β schedule for half quadratic splitting. Then, our inference network consists of T HQS layers as shown in Figure 6.3. Here, $\mathbf{X}$ is the input image with noise variance σ^2, and $\{\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\}$ are the (data-dependent) pairwise potential parameters generated by the PGNet.

Remark 6.1 *Since our inference network implements a fixed number of HQS iterations, its output may not be optimal for* (6.4)*. However, since we train our parameter generation and inference networks jointly in a discriminative fashion, the*

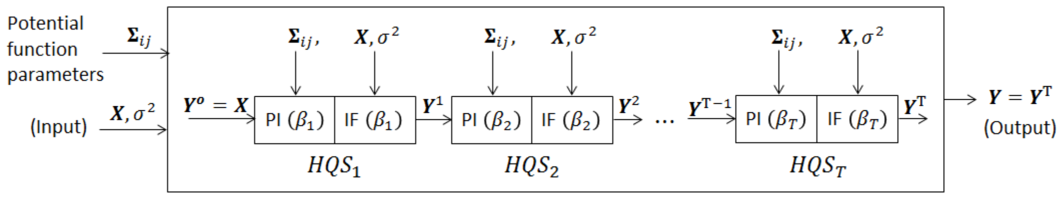

Figure 6.3: Inference network uses the pairwise potential parameters $\{\mathbf{\Sigma}_{ij}(\bar{\mathbf{x}}_{ij})\}$ generated by the PGNet and performs T HQS iterations.

PGNet will learn to generate appropriate pairwise potential parameters such that the output after a fixed number of HQS iterations would be close to the desired output.

6.4.3 Gaussian CRF Network

Combining the above parameter generation and inference networks, we get our full Gaussian CRF network with parameters $\{(\mathbf{W}_k \succeq 0, \mathbf{\Psi}_k \succeq 0, b_k)\}$. Note that this Gaussian CRF network has various new types of layers that use quadratic functions, matrix inversions and multiplicative interactions, which are quite different from the computations used in standard deep networks.

Additional PGNets: Note that using appropriate pairwise potential functions is crucial for the success of a Gaussian CRF model. Since the parameter generation network operates on the noisy input image $\mathbf{X}$, it is very difficult to generate good parameters at high noise levels (even after incorporating the noise variance σ^2 into the selection network). To overcome this issue, we introduce an additional PGNet after each HQS iteration (shown with dotted boxes in Figure 6.1). The rationale behind adding these additional PGNets is that even if the first PGNet fails to generate good parameters, the later PGNets could generate appropriate parameters using the partially restored images. Our final deep Gaussian CRF network consists of T PGNets and T HQS layers as shown in Figure 6.1.

6.4.4 Training

Since all the components of the proposed deep Gaussian CRF network are differentiable, it can be trained end-to-end in a discriminative fashion. Here, we show how to back-propagate the loss derivatives through the layers of the proposed deep network. Please refer to Appendix B for detailed derivations. Let L be the final loss function.

Backpropagation through the combination layer: Given the derivatives $dL/d\mathbf{\Sigma}_{ij}$ of the loss function L with respect to the pairwise potential parameters $\mathbf{\Sigma}_{ij}$, we can compute the derivatives of L with respect to the combination weights γ_{ij}^k and the matrices $\mathbf{\Psi}_k$ using

$$\frac{dL}{d\gamma_{ij}^k} = \text{trace}\left(\mathbf{\Psi}_k^\top \frac{dL}{d\mathbf{\Sigma}_{ij}}\right), \quad \frac{dL}{d\mathbf{\Psi}_k} = \sum_{ij} \gamma_{ij}^k \frac{dL}{d\mathbf{\Sigma}_{ij}}. \tag{6.11}$$

Backpropagation through the quadratic layer: Given the derivatives dL/ds_{ij}^k of the loss function L with respect to the quadratic layer output s_{ij}^k, we can compute the derivatives of L with respect to the selection network parameters $(\mathbf{W}_k, b_k)$ and the input patches $\bar{\mathbf{x}}_{ij}$ using:

$$\begin{aligned}
\frac{dL}{d\mathbf{W}_k} &= \left(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2}\right)^{-1} \left(\sum_{ij} \frac{dL}{ds_{ij}^k} \frac{\bar{\mathbf{x}}_{ij}\bar{\mathbf{x}}_{ij}^\top}{2}\right) \left(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2}\right)^{-1} \\
\frac{dL}{db_k} &= \sum_{ij} \frac{dL}{ds_{ij}^k}, \quad \frac{dL}{d\bar{\mathbf{x}}_{ij}} = -\sum_k \frac{dL}{ds_{ij}^k} \left(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2}\right)^{-1} \bar{\mathbf{x}}_{ij}.
\end{aligned} \tag{6.12}$$

Backpropagation through the patch inference layer: Given the derivatives $dL/d\mathbf{z}_{ij}$ of the loss function L with respect to the output of a patch inference layer, we can compute the derivatives of L with respect to its input patches $\mathbf{y}_{ij}$ and the pairwise potential parameters $\mathbf{\Sigma}_{ij}$ using

$$\begin{aligned} \frac{dL}{d\mathbf{y}_{ij}} &= \left(\mathbf{I}_{d^2} - \mathbf{G}^\top(\beta\mathbf{\Sigma}_{ij} + \mathbf{G})^{-1}\mathbf{G}\right)\frac{dL}{d\mathbf{z}_{ij}}, \\ \frac{dL}{d\mathbf{\Sigma}_{ij}} &= \beta\left(\beta\mathbf{\Sigma}_{ij} + \mathbf{G}\right)^{-1}\mathbf{G}\frac{dL}{d\mathbf{z}_{ij}}\mathbf{y}_{ij}^\top\mathbf{G}^\top\left(\beta\mathbf{\Sigma}_{ij} + \mathbf{G}\right)^{-1}. \end{aligned} \tag{6.13}$$

We skip the derivative formulas for other computations such as softmax, extracting mean-subtracted patches from an image, averaging in the image formation layer, etc., as they are standard operations. Note that we have a constrained optimization problem here because of the symmetry and positive semi-definiteness constraints on the network parameters $\{\mathbf{W}_k\}$ and $\{\mathbf{\Psi}_k\}$. We convert this constrained problem into an unconstrained one by parametrizing $\mathbf{W}_k$ and $\mathbf{\Psi}_k$ as $\mathbf{W}_k = \mathbf{P}_k\mathbf{P}_k^\top$, $\mathbf{\Psi}_k = \mathbf{R}_k\mathbf{R}_k^\top$, where $\mathbf{P}_k$ and $\mathbf{R}_k$ are lower triangular matrices.

6.5 Experimental Evaluation

In this section, we use the proposed deep Gaussian CRF network for image denoising. We trained our network using a dataset of 400 images (200 images from BSD300 [155] training set and 200 images from PASCALVOC 2012 [156] dataset) by maximizing the Peak Signal-to-Noise Ratio (PSNR) measure. We used limited memory BFGS [157] for optimization. For testing, we used a dataset of 300 images (100 images from BSD300 test set and 200 images from PASCALVOC 2012 dataset). We used white Gaussian noise of various standard deviations in our experiments.

For realistic evaluation, all the images were quantized to [0-255] range after adding the noise. We use the standard PSNR measure for quantitative evaluation.

Though we use Gaussian noise, due to quantization (clipping to 0-255 range), the noise characteristics deviate from being a Gaussian as the noise variance increases. To cope up with this variation in noise characteristics, we trained two different networks, one for low input noise levels ($\sigma \leq 25$, noise reasonably close to a Gaussian after quantization) and one for high input noise levels ($25 < \sigma < 60$, noise far from being a Gaussian after quantization). When we tried training a single network for all noise levels, the training was mainly focusing on high noise data. For training the low noise network, we used $\sigma = [8, 13, 18, 25]$ and for training the high noise network, we used $\sigma = [30, 35, 40, 50]$. Note that both the networks were trained to handle a range of input noise levels. For testing, we varied the σ from 10 to 60 in intervals of 5.

We performed experiments with two patch sizes (5×5 and 8×8), and the number of matrices $\mathbf{\Psi}_k$ was chosen as 200. Following [123], we used six HQS iterations with β values given by $\frac{1}{\sigma^2}[1, 4, 8, 16, 32, 64]$. Optimizing the β values using a validation set may further improve our performance. To avoid overfitting, we regularized the network, by sharing the parameters $\{\mathbf{W}_k, \mathbf{\Psi}_k\}$ across all PGNets. We initialized the network parameters using the parameters of a GMM learned on clean image patches.

Table 6.1 compares the proposed deep Gaussian CRF network with various image denoising approaches on 300 test images. Here, DGCRF$_5$ and DGCRF$_8$ refer to the deep Gaussian CRF networks that use 5×5 and 8×8 patches, respectively.

Table 6.1: Comparison of various denoising approaches on 300 test images.

Test σ	10	15	20	25
ClusteringSR [139]	33.27	30.97	29.41	28.22
EPLL [123]	33.32	31.06	29.52	28.34
BM3D [122]	33.38	31.09	29.53	28.36
NL-Bayes [144]	33.46	31.11	29.63	28.41
NCSR [140]	33.45	31.20	29.56	28.39
WNNM [143]	**33.57**	31.28	29.70	28.50
CSF [130]	-	-	-	28.43
MLP [28]	33.43	-	-	**28.68**
DGCRF$_5$ Low noise network	33.53	**31.29**	**29.76**	28.58
DGCRF$_8$ Low noise network	**33.56**	**31.35**	**29.84**	**28.67**

Test σ	30	35	40	45	50	55	60
ClusteringSR [139]	27.25	26.30	25.56	24.89	24.28	23.72	23.21
EPLL [123]	27.36	26.52	25.76	25.08	24.44	23.84	23.27
BM3D [122]	27.42	26.64	25.92	25.19	24.63	24.11	23.62
NL-Bayes [144]	27.42	26.57	25.76	25.05	24.39	23.77	23.18
NCSR [140]	27.45	26.32	25.59	24.94	24.35	23.85	23.38
WNNM [143]	27.51	26.67	25.92	25.22	24.60	24.01	23.45
MLP [28]	-	**27.13**	-	-	**25.33**	-	-
DGCRF$_5$ High noise network	**27.68**	26.95	**26.30**	**25.73**	25.23	**24.76**	**24.33**
DGCRF$_8$ High noise network	**27.80**	**27.08**	**26.44**	**25.88**	**25.38**	**24.90**	**24.45**

For each noise level, the top two PSNR values are shown in boldface style. Note that the CSF [130] and MLP [28] approaches train a different model for each noise level. Hence, for these approaches, we report the results only for those noise levels for which the corresponding authors have provided their trained models. As we can see, the proposed deep Gaussian CRF network clearly outperforms the ClusteringSR [139], EPLL [123], BM3D [122], NL-Bayes [144], NCSR [140] and CSF approaches on all noise levels, and the WNNM [143] approach on all noise levels except $\sigma = 10$ (where it performs equally well). Specifically, it produces significant improvement in the PSNR compared to the ClusteringSR (0.29 - 1.24 dB), EPLL (0.24 - 1.18 dB), BM3D (0.18 - 0.83 dB), NL-Bayes (0.10 - 1.27 dB), NCSR (0.11 - 1.07 dB) and WNNM (upto 1.0 dB) approaches. The CSF approach of [130], which also uses Gaussian CRFs, performs poorly (0.24 dB for $\sigma = 25$) compared to our deep network.

When compared with MLP [28], which is the state-of-the-art deep networks-based denoising approach, we perform better for $\sigma = [10, 50]$, worse for $\sigma = 35$, and equally well for $\sigma = 25$. However, note that while [28] uses a different MLP for each specific noise level, we trained only two networks, each of which can handle a range of noise levels. In fact, our single low noise network is able to outperform the MLP trained for $\sigma = 10$ and perform as good as the MLP trained for $\sigma = 25$. This ability to handle a range of noise levels is one of the major benefits of the proposed deep network. Note that though we did not use the noise levels $\sigma = 10, 15, 20, 45$ during training, our networks performs very well for these σ. This shows that our networks are able to handle a range of noise levels rather than just being effective for the trained σ. Also, our high noise network performs well for $\sigma = 55$ and 60

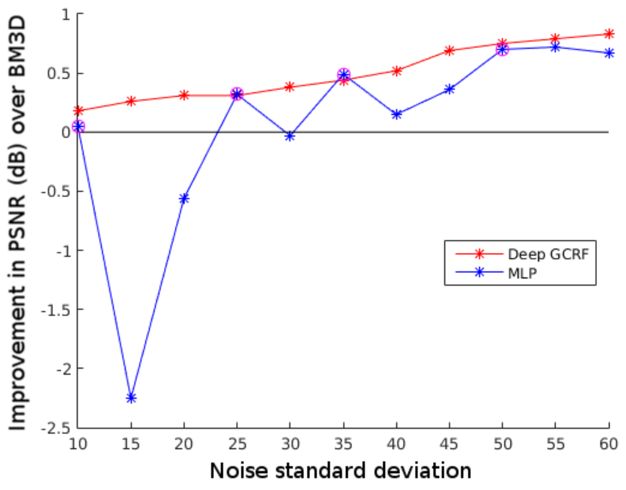

Figure 6.4: Sensitivity analysis of the MLP and the proposed approach. The noise levels for which MLP was trained are indicated using a circular marker.

even though these values are out of its training range. This shows that the proposed *model-based* deep network can also generalize reasonably well for out-of-range noise levels.

We acknowledge that the comparisons in Table 6.1 may be biased since some of the competing methods are not designed for denoising quantized images. However, we believe that, for the denoising problem, using quantized images is a more realistic experimental setting than using unquantized images. Please refer to Table 6.2 for additional results on a benchmark dataset under the unquantized setting.

To analyze the sensitivity of the *non-model* based MLP approach to the deviation from training noise, we evaluated it on noise levels that are slightly (± 5) different from the training σ. The authors of [28] trained separate MLPs for $\sigma = 10, 25, 35, 50$ and 65. As reported in [28], training a single MLP to handle multiple noise levels gave inferior results. Figure 6.4 shows the improvement of the

MLP approach over BM3D in terms of PSNR. For each noise level, we used the best performing model among $\sigma = 10, 25, 35, 50, 65$. As we can see, while the MLP approach does very well for the exact noise levels for which it was trained, it performs poorly if the test σ deviates from the training σ even by 5 units. This is a major limitation of the MLP approach since training a separate model for each noise level is not practical. Contrary to this, the proposed approach is able to cover a wide range of noise levels just using two networks.

Please note that the purpose of Figure 6.4 is not to compare the performance of our approach with MLP on noise levels that were not used in MLP training, which would be an unfair comparison. The only purpose of this figure is to show that, although very powerful, a network trained for a specific noise level is very sensitive.

Apart from our test set of 300 images, we also evaluated our low noise DGCRF$_8$ network on a smaller dataset of 68 images [1] which has been used in various existing works. Tables 6.2 and 6.3 compare the proposed deep Gaussian CRF network with various approaches on this dataset under the unquantized and quantized settings, respectively. For each noise level, the top two PSNR values are shown in boldface style. As we can see, the proposed approach outperforms all the other approaches except RTF$_5$ [129] and MLP [28] under the quantized setting, and TRD [136] under the unquantized setting. However, note that while we use a single network for both $\sigma = 15$ and $\sigma = 25$, the MLP, TRD and RTF$_5$ approaches trained their models specifically for individual noise levels.

Table 6.2: Comparison of various denoising approaches on 68 images (dataset of [1]) under the unquantized setting.

Test σ	ARF [148]	LLSC [142]	EPLL [123]	opt-MRF [158]	ClusteringSR [139]	NCSR [140]	BM3D [122]
15	30.70	31.27	31.19	31.18	31.08	31.19	31.08
25	28.20	28.70	28.68	28.66	28.59	28.61	28.56

Test σ	MLP [28]	WNNM [143]	CSF [130]	RTF_5 [129]	TRD [136]	$DGCRF_8$
15	-	31.37	31.24	-	**31.43**	**31.43**
25	28.85	28.83	28.72	28.75	**28.95**	**28.89**

Table 6.3: Comparison of various denoising approaches on 68 images (dataset of [1]) under the quantized setting.

Test σ	LLSC [142]	EPLL [123]	opt-MRF [158]	ClusteringSR [139]	NCSR [140]	BM3D [122]
15	31.09	31.11	31.06	30.93	31.13	31.03
25	28.24	28.46	28.40	28.26	28.41	28.38

Test σ	NL-Bayes [144]	MLP [28]	WNNM [143]	CSF [130]	RTF_5 [129]	$DGCRF_8$
15	31.06	-	31.20	-	-	**31.36**
25	28.43	**28.77**	28.48	28.53	**28.74**	28.73

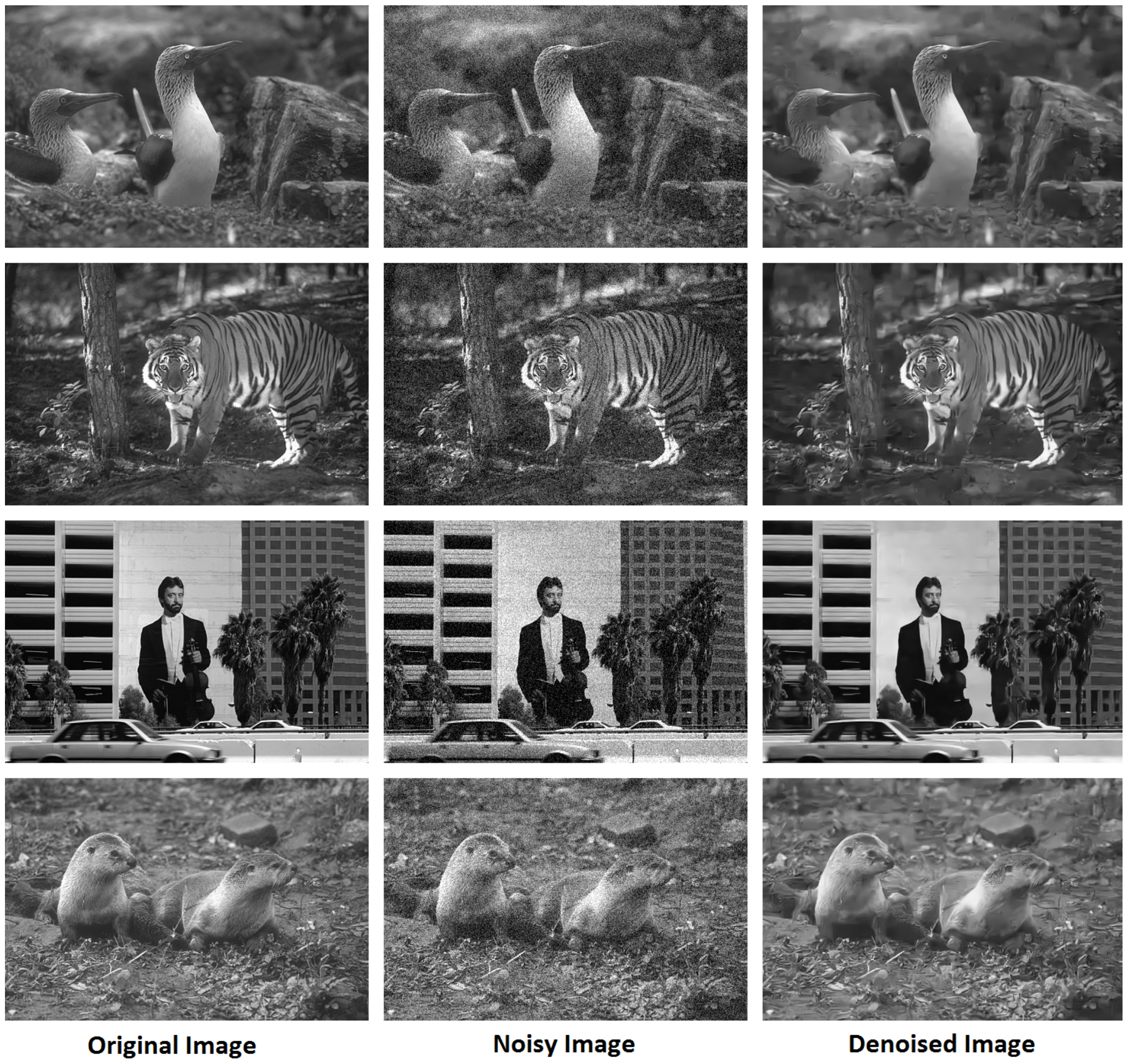

Figure 6.5: Denoising results by the proposed approach for noise $\sigma = 25$.

Figure 6.5 shows some example denoising results produced by the proposed approach for noise standard deviation $\sigma = 25$. As we can see, the proposed approach is able to retain the image content while suppressing the noise.

Denoising time: The proposed DGCRF$_8$ network takes 4.4s for a 321×481 image on an NVIDIA Titan GPU using a MATLAB implementation.

6.6 Conclusions

In this chapter, we proposed a new end-to-end trainable deep network architecture for image denoising using a Gaussian CRF model. The proposed network consists of a parameter generation network that generates appropriate potential function parameters based on the input image, and an inference network that performs approximate Gaussian CRF inference. Unlike existing discriminative denoising approaches that train a separate model for each noise level, the proposed network can handle a range of noise levels as it explicitly models the input noise variance. We achieved results on par with the state-of-the-art by training two deep Gaussian CRF networks, one for low input noise levels and one for high input noise levels.

Chapter 7: Gaussian Conditional Random Field Network for Semantic Segmentation

7.1 Introduction

Semantic segmentation, which aims to predict a category label for every pixel in the image, is an important task for scene understanding. Though it has received significant attention from the vision community over the past few years, it still remains a challenging problem due to large variations in the visual appearance of the semantic classes and complex interactions between various classes in the visual world. Recently, CNNs have been shown to work very well for this challenging task [30, 31, 159–161]. Their success can be attributed to several factors such as their ability to represent complex input-output relationships, feed-forward nature of their inference, availability of large training datasets and fast computing hardware like GPUs, etc.

However, CNNs may not be optimal for structured prediction tasks such as semantic segmentation as they do not model the interactions between output variables directly. Acknowledging this, various semantic segmentation approaches have been proposed in the recent past that use CRF models [162] on top of CNNs [2, 30, 151, 152, 163–165], and all these approaches have shown significant improvement in the segmentation results by using CRFs. By combining CNNs and CRFs, these approaches get the best of both worlds: the ability of CNNs to model complex input-output relationships and the ability of CRFs to directly model the interactions between output variables. While some of these approaches use CRF as a separate post-processing step [2, 30, 163–165], some other approaches train the CNNs along with the CRFs in an end-to-end fashion [151, 152].

All of the above approaches use discrete graphical models, and hence end up using graph-cuts or mean field-based approximate inference procedures. Though these inference procedures do not have global optimum guarantees, they have been successfully used for the semantic segmentation task in conjunction with CNNs. Different from discrete graphical models, Gaussian graphical models [9, 124] are simpler models, and have inference procedures that are guaranteed to converge to the global optimal solution. Gaussian graphical models have been used in the past for various applications such as image denoising [124, 128], depth estimation [131, 166], deblurring [123, 129], edge detection [167], texture classification [168], texture segmentation [169], etc.

While a discrete CRF is a natural fit for labeling tasks such as semantic segmentation, one needs to use inference techniques that do not have optimality guarantees. While exact inference is tractable in the case of a Gaussian CRF, it is not clear if this model is a good fit for discrete labeling tasks. This leads us to the following question: *Should we use a better model with approximate inference or an approximate model with better inference?*

To answer this question, in this chapter, we use a Gaussian CRF model for the task of semantic segmentation. To use a Gaussian CRF model for this discrete labeling task, we first replace each discrete variable with a vector of K mutually exclusive binary variables, where K is the number of possible values the discrete variable can take, and then model all the variables jointly as a multivariate Gaussian by relaxing the mutual exclusivity and binary constraints. After the Gaussian CRF inference, the discrete label assignment is done based on which of the K corresponding variables has the maximum value.

Though the Maximum a Posteriori (MAP) solution can be obtained in closed form in the case of Gaussian CRFs, it involves solving a linear system with number of variables equal to the number of nodes in the graph times the dimensionality of node variables (which is equal to the number of spatial locations times the number of classes in the case of semantic segmentation). Solving such a large linear system could be computationally prohibitive, especially for dense graphs where each node is connected to several other nodes. Hence, instead of exactly solving a large linear system, we unroll a fixed number of Gaussian Mean Field (GMF) inference steps as layers of a deep network, which we refer to as the GMF network. Note that the GMF inference is different from the mean field inference used in [170] for discrete CRFs with Gaussian edge potentials.

While GMF updates are guaranteed to give the MAP solution upon convergence, parallel updates are guaranteed to converge only under certain constraints such as diagonal dominance of the precision matrix of the joint Gaussian [171]. If the nodes are updated serially, then the GMF inference is equivalent to an alternating minimization approach in which each subproblem is solved optimally, and hence it will converge (as finding the MAP solution for a Gaussian CRF is a convex problem with a smooth cost function). But, using serial updates would be very slow when the number of variables is large. To avoid both these issues, we use a bipartite graph structure that allows us to update half of the nodes in parallel in each step without loosing the convergence guarantee even when the diagonal dominance constraint is not satisfied. Using this bipartite structure, we ensure that each layer of our GMF network produces an output that is closer to the MAP solution compared to its input.

By combining the proposed GMF network with CNNs, we propose a new end-to-end trainable deep network, which we refer to as Gaussian CRF network (Figure 7.1), for the task of semantic segmentation. The proposed Gaussian CRF network consists of a CNN-based unary network for generating the unary potentials, a CNN-based pairwise network for generating the pairwise potentials and a GMF network for performing the Gaussian CRF inference.

Contributions: Different from existing approaches that use discrete CRF models, we propose to use a Gaussian CRF model for the task of semantic segmentation. Compared to discrete CRFs, Gaussian CRFs are simpler models that can be solved optimally. We propose a novel deep network for Gaussian CRF inference by unfolding a fixed number of GMF iterations. Using a bipartite graph structure, we ensure that each layer in our inference network produces an output that is closer to

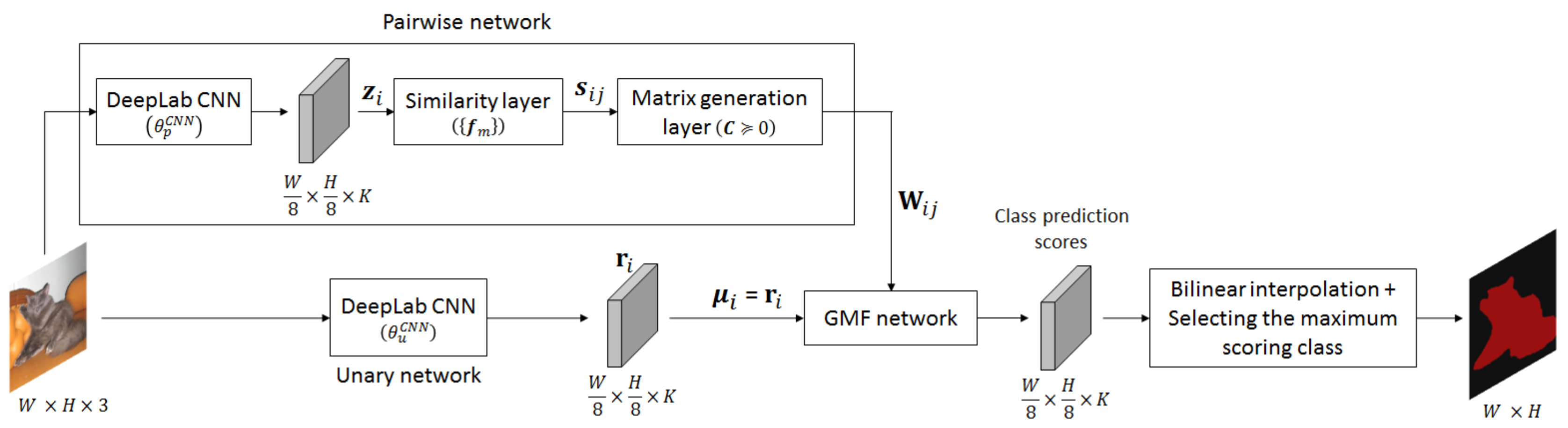
Pairwise network
DeepLab CNN (θ_p^{CNN})
$\frac{W}{8} \times \frac{H}{8} \times K$
z_i
Similarity layer ($\{f_m\}$)
s_{ij}
Matrix generation layer ($C \succcurlyeq 0$)
$\mathbf{W}_{ij}$
$W \times H \times 3$
DeepLab CNN (θ_u^{CNN})
Unary network
$\mathbf{r}_i$
$\frac{W}{8} \times \frac{H}{8} \times K$
$\boldsymbol{\mu}_i = \mathbf{r}_i$
GMF network
Class prediction scores
$\frac{W}{8} \times \frac{H}{8} \times K$
Bilinear interpolation + Selecting the maximum scoring class
$W \times H$

Figure 7.1: Proposed Gaussian CRF network: The GMF network performs Gaussian CRF inference using the outputs of unary and pairwise networks. The output of GMF network is upsampled to full image resolution using bilinear interpolation. Note that the parameters of this network are the unary network parameters θ_u^{CNN} and the pairwise network parameters $\{\theta_p^{CNN}, \{\mathbf{f}_m\}, \mathbf{C} \succeq 0\}$.

the optimal solution compared to its input. We propose a new end-to-end trainable deep network that combines the Gaussian CRF model with CNNs for the task of semantic segmentation. We show that the proposed Gaussian CRF network outperforms various discrete CRF-based approaches on the challenging PASCALVOC 2012 test set [156] (when trained with ImageNet [172] and PASCALVOC data).

Organization: Section 7.2 provides an overivew of existing works on semantic segmentation, Gaussian CRFs, and inference unfolding. Section 7.3 presents the Gaussian CRF model used in this chapter, and Section 7.4 presents the proposed Gaussian CRF network. Experimental results and conclusions are presented in Sections 7.5 and 7.6, respectively.

7.2 Related Work

Semantic segmentation using CNNs: In the recent past, numerous semantic segmentation approaches have been proposed based on CNNs. In [121, 173], each region proposal was classified into one of the semantic classes by using CNN features. Instead of applying a CNN to each region independently as in [121, 173], [174] applied the convolutional layers only once to the entire image, and generated region features by using pooling after the final convolutional layer.

Different from the above approaches, [30] trained a CNN to directly extract features at each pixel. To capture the information present at multiple scales, CNN was applied to the input image multiple times at different resolutions, and the features from all the resolutions were concatenated to get the final pixel features.

This multiscale feature was then classified using a two-layer neural network. Finally, post-processing steps like CRF and segmentation tree were used to further improve the results. Building on top of these CNN features, [175,176] introduced a recursive context propagation network that enriched the CNN features by adding image level contextual information. Instead of using a CNN multiple times, [2,160,161] proposed to use the features extracted by the intermediate layers of a deep CNN to capture the multi-scale information. Recently, [177] trained a deconvolution network for the task of semantic segmentation. This network was applied separately to each region proposal, and all the results were aggregated to get the final predictions.

Most of the CNN-based methods mentioned above use superpixels or region proposals, and hence the errors in the initial proposals will remain no matter how good the CNN features are. Different from these methods, [31] directly produced dense segmentation maps by upsampling the predictions produced by a CNN using a trainable deconvolution layer. To obtain the finer details in the upsampled output, they combined the final layer predictions with predictions from lower layers.

Combining CNNs and CRFs for semantic segmentation: Though CNNs have been shown to work very well for the task of semantic segmentation, they may not be optimal as they do not model the interactions between the output variables directly, which is important for semantic segmentation. To overcome this issue, various recent approaches [2, 30, 163–165] have used discrete CRF [162] models on top of CNNs. While [30] defined a CRF on superpixels and used graph-cuts based inference, [2, 163–165] defined a CRF directly on image pixels and used the efficient mean field inference proposed in [170]. Instead of using CRF as a post-processing step, [152] trained a CNN along with a CRF in an end-to-end fashion by converting the mean field inference procedure of [170] into a recurrent neural network. Similar joint training strategy was also used in [151].

In all these approaches, the CRF edge potentials were designed using hand-chosen features like image gradients, pixel color values, spatial locations, etc. and the potential function parameters were manually tuned. Contrary to this, recently, [178] has learned both unary and pairwise potentials using CNNs. While all these approaches learn CNN-based potentials and use message passing algorithms to perform CRF inference, [179] has recently proposed to use CNNs to directly learn the messages in message passing inference.

The idea of jointly training a CNN and graphical model has also been used for other applications such as sequence labeling [180, 181], text recognition [182], human pose estimation [183], predicting words from images [184], handwritten word recognition [185]. Recently, various CNN-based semantic segmentation approaches have also been proposed for semi and weakly supervised settings [164, 186–188].

Unrolling inference as a deep network: The proposed approach is also related to a class of algorithms that learn model parameters discriminatively by back-propagating the gradient through a fixed number of inference steps. In [148], the fields of experts [1] model was discriminatively trained for image denoising by unrolling a fixed number of gradient descent inference steps. In [184, 189–191] discrete graphical models were trained by back-propagating through either the mean field or the belief propagation inference iterations. In [149], message passing inference machines were trained by considering the belief propagation-based inference of a discrete graphical model as a sequence of predictors. In [150], a feed-forward sparse code predictor was trained by unrolling a coordinate descent-based sparse coding inference algorithm. In [153], a new non-negative deep network was introduced by deep unfolding of non-negative factorization model. Different from these approaches, we unroll the mean filed inference of a Gaussian CRF model as a deep network, and train our CNN-based potential functions along with the Gaussian CRF inference network in an end-to-end fashion.

Gaussian conditional random fields: Gaussian CRFs [124] are popular models for structured inference tasks like denoising [123, 124, 128–130], deblurring [123, 129, 130], depth estimation [131, 166], etc., as they model continuous quantities and can be efficiently solved using linear algebra routines.

Gaussian CRF was also used for discrete labeling tasks earlier in [192], where a Logistic Random Field (LRF) was proposed by combining a quadratic model with logistic function. While the LRF used a logistic function on top of a Gaussian CRF to model the output, we directly model the output using a Gaussian CRF. Unlike [192], which used hand-chosen features like image gradients, color values, etc. to model the potentials, we use CNN-based potential functions.

Recently, [166] trained a CNN along with a Gaussian CRF model for image-based depth prediction. The Gaussian CRF model of [166] was defined on super-pixels and had edges only between adjacent superpixels. As the resulting graph was sparse with few nodes, [166] performed exact Gaussian CRF inference by solving a linear system. Different from [166], we define our Gaussian CRF model directly on top of the dense CNN output and connect each node to several neighbors. Since the number of variables in our Gaussian CRF model is very large, exactly solving a linear system would be computationally expensive. Hence, we unfold a fixed number of GMF inference steps into a deep network. Also, while [166] used hand-designed features like color histogram, local binary patterns, etc. for designing their pairwise potentials, we use CNN-based pairwise potentials.

7.3 Gaussian Conditional Random Field Model

In semantic segmentation, we are interested in assigning each pixel in an image $\mathbf{X}$ to one of the K possible classes. As mentioned earlier, we use K variables (one for each class) to model the output at each pixel, and the final label assignment is done based on which of these K variables has the maximum value. Let $\mathbf{y}_i = [y_{i1}, \ldots, y_{iK}]$ be the vector of K output variables associated with the i^{th} pixel, and $\mathbf{y}$ be the vector of all output variables. We model the conditional probability density $P(\mathbf{y}|\mathbf{X})$ as a Gaussian distribution given by $P(\mathbf{y}|\mathbf{X}) \propto \exp\ \left\{-\frac{1}{2}\ E(\mathbf{y}|\mathbf{X})\right\}$, where

$$E\left(\mathbf{y}|\mathbf{X}\right) = \sum_i \|\mathbf{y}_i - \mathbf{r}_i(\mathbf{X};\theta_u)\|_2^2 + \sum_{ij} \left(\mathbf{y}_i - \mathbf{y}_j\right)^\top \mathbf{W}_{ij}\left(\mathbf{X};\theta_p\right)\left(\mathbf{y}_i - \mathbf{y}_j\right). \tag{7.1}$$

The first term in the above energy function E is the unary term and the second term is the pairwise term. Here, both $\mathbf{r}_i$ and $\mathbf{W}_{ij} \succeq 0$ are functions of the input image $\mathbf{X}$ with θ_u and θ_p being the respective function parameters. Note that when $\mathbf{W}_{ij} \succeq 0$ for all pairs of pixels, the unary and pairwise terms can be combined together into a single positive semidefinite quadratic form.

The optimal $\mathbf{y}$ that minimizes the energy function E can be obtained in closed form since the minimization of E is an unconstrained quadratic program. However, this closed form solution involves solving a linear system with number of variables equal to the number of pixels times the number of classes. Since solving such a large linear system could be computationally prohibitive, we use the iterative mean field inference approach.

7.3.1 Gaussian Mean Field Inference

The standard mean field approach approximates the joint distribution $P(\mathbf{y}|\mathbf{X})$ using a simpler distribution $Q(\mathbf{y}|\mathbf{X})$ which can be written as a product of independent marginals, i.e, $Q(\mathbf{y}|\mathbf{X}) = \prod_i Q_i(\mathbf{y}_i|\mathbf{X})$. This approximate distribution is obtained by minimizing the KL-divergence between the distributions P and Q. In the case of Gaussian, the mean field approximation Q and the original distribution P have the same mean [171]. Hence, finding the MAP solution $\mathbf{y}$ is equivalent to finding the mean $\boldsymbol{\mu}$ of the distribution Q.

For the Gaussian distribution in (7.1), the mean field updates for computing the mean $\boldsymbol{\mu}$ are given by

$$\boldsymbol{\mu}_i \leftarrow \Big(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\Big)^{-1}\Big(\mathbf{r}_i + \sum_j \mathbf{W}_{ij}\boldsymbol{\mu}_j\Big). \tag{7.2}$$

Here, $\boldsymbol{\mu}_i$ is the mean of marginal Q_i. Please refer to Appendix A for detailed derivations. It is easy to see that if we use the standard alternating minimization approach (in which we update one pixel at a time) to find the optimal $\mathbf{y}$ that minimizes the energy function in (7.1), we would end up with the same update equation. Since the energy function is a convex quadratic in the case of Gaussian CRF and update (7.2) solves each subproblem optimally, i.e., finds the optimal $\mathbf{y}_i$ (or $\boldsymbol{\mu}_i$) when all the other $\mathbf{y}_j$ (or $\boldsymbol{\mu}_j$) are fixed, performing serial updates is guaranteed to give us the MAP solution. However, it would be very slow since we are dealing with a large number of variables.

While using parallel updates seems to be a reasonable alternative, convergence of parallel updates is guaranteed only under certain constraints like diagonal dominance of the precision matrix of the distribution P [171]. Imposing such constraints could restrict the model capacity in practice. For example, in our Gaussian CRF model (7.1), we can satisfy the diagonal dominance constraint by making all $\mathbf{W}_{ij}$ diagonal. However, this can be very restrictive, as making the non-diagonal entries of $\mathbf{W}_{ij}$ zero will remove the direct inter-class interactions between pixels i and j, i.e., there will not be any interaction term in the energy function between the variables y_{ip} and y_{jq} for $p \neq q$.

7.3.2 Bipartite Graph Structure for Parallel Updates

While we want to avoid the diagonal dominance constraint, we also want to update as many variables as possible in parallel. To address this problem, we use a bipartite graph structure, which allows us to update half of the variables in parallel in each step, and still guarantees convergence without any constraints.

Note that our graphical model has a node for each pixel, and each node rep-

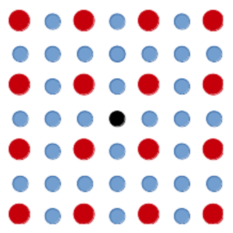

Figure 7.2: Each pixel in our CRF is connected to every other pixel along both rows and columns within a spatial neighborhood. Here, all the pixels that are connected to the center black pixel are shown in red. If the black pixel is on odd column, all the pixels connected to it will be on even columns and vice versa.

resents a vector of K variables. In order to update the i^{th} node using (7.2), we need to keep all the other nodes connected to the i^{th} node (i.e., all the nodes with non-zero $\mathbf{W}_{ij}$) fixed. If we partition the image into odd and even columns (or odd and even rows) and avoid edges within the partitions, then we can optimally update all the odd columns (or rows) in parallel using (7.2) while keeping the even columns (or rows) fixed and vice versa. This is again nothing but an alternating minimization approach in which each subproblem (corresponding to half of the nodes in the graph) is optimally solved, and hence is guaranteed to converge to the global optimum (since we are dealing with a convex problem).

Generally when using graphical models, each pixel is connected to all the pixels within a spatial neighborhood. Here, instead of using all the neighbors, we connect each pixel to every other neighbor along both rows and columns. Figure 7.2 illustrates this for a 7×7 spatial neighborhood. It is easy to see that with this connectivity, we can partition the image into even and odd columns (or even and odd rows) without any edges within the partitions.

7.4 Gaussian CRF network

The proposed Gaussian CRF network consists of three components: *Unary network*, *Pairwise network* and *GMF network*. While the unary and pairwise networks generate the $\mathbf{r}_i$ and $\mathbf{W}_{ij}$ that are respectively used in the unary and pairwise terms of the energy function (7.1), the GMF network performs Gaussian mean field inference using the outputs of unary and pairwise networks. Figure 7.1 gives an overview of the proposed Gaussian CRF network.

Unary network: To generate the $\mathbf{r}_i$ used in the unary term of the energy function (7.1), we use the DeepLab-MSc-LargeFov network of [2] (along with the softmax layer), which is a modified version of the popular VGG-16 network [6]. Modifications compared to VGG-16 include converting the fully-connected layers into convolutional layers, skipping downsampling after the last two pooling layers, modifying the convolutional layers after the fourth pooling layer, and using the multi-scale features. Please refer to [2] for further details. For brevity, we will refer to this DeepLab-MSc-LargeFov network as DeepLab CNN in the rest of this chapter. We will denote the parameters of this unary DeepLab network using θ_u^{CNN}.

Pairwise network: Our pairwise network generates the matrices $\mathbf{W}_{ij}$ that are used in the pairwise term of the energy function (7.1). We compute each $\mathbf{W}_{ij}$ as

$$\mathbf{W}_{ij} = s_{ij}\mathbf{C},\ \ \mathbf{C} \succeq 0, \tag{7.3}$$

where $s_{ij} \in [0,1]$ is a measure of similarity between pixels i and j, and the learned matrix $\mathbf{C}$ encodes the class compatibility information. We compute the similarity measure s_{ij} using

$$s_{ij} = e^{-(\mathbf{z}_i - \mathbf{z}_j)^\top \mathbf{F}(\mathbf{z}_i - \mathbf{z}_j)}, \tag{7.4}$$

where $\mathbf{z}_i$ is the feature vector extracted at i^{th} pixel using a DeepLab CNN (with parameters θ_p^{CNN}), and the learned matrix $\mathbf{F} \succeq 0$ defines a Mahalanobis distance function. Note that the exponent of s_{ij} can be written as

$$(\mathbf{z}_i - \mathbf{z}_j)^\top \mathbf{F}(\mathbf{z}_i - \mathbf{z}_j) = \sum_{m=1}^{M} (\mathbf{f}_m^\top \mathbf{z}_i - \mathbf{f}_m^\top \mathbf{z}_j)^2, \tag{7.5}$$

where $\mathbf{F} = \sum_{m=1}^{M} \mathbf{f}_m \mathbf{f}_m^\top$. Hence, we implement the Mahalanobis distance computation as convolutions (of $\mathbf{z}_i$ with filters $\mathbf{f}_m$) followed by an Euclidean distance computation.

The overall pairwise network consists of a DeepLab CNN that generates the pixel features $\mathbf{z}_i$, a similarity layer that computes s_{ij} for every pair of connected pixels using (7.4) and (7.5), and a matrix generation layer that computes the matrices $\mathbf{W}_{ij}$ using (7.3). Note that here $\{\mathbf{f}_m\}$ are the parameters of the similarity layer and $\mathbf{C} \succeq 0$ are the parameters of the matrix generation layer.

GMF network: The proposed GMF network performs a fixed number of Gaussian mean field updates using the outputs of unary and pairwise networks. The input to the network is initialized using the unary output, $\mu^1 = \mathbf{r} = \{\mathbf{r}_i\}$. The network consists of several sequential GMF layers, where each GMF layer has two sub-layers (an even update layer followed by an odd update layer, See Figure 7.3):

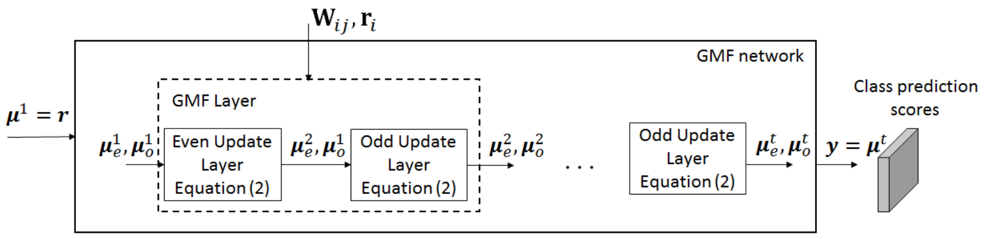
$\mathbf{W}_{ij}, \mathbf{r}_i$
GMF network
GMF Layer
$\boldsymbol{\mu}^1 = \boldsymbol{r}$
$\boldsymbol{\mu}_e^1, \boldsymbol{\mu}_o^1$
Even Update Layer Equation (2)
$\boldsymbol{\mu}_e^2, \boldsymbol{\mu}_o^1$
Odd Update Layer Equation (2)
$\boldsymbol{\mu}_e^2, \boldsymbol{\mu}_o^2$
Odd Update Layer Equation (2)
$\boldsymbol{\mu}_e^t, \boldsymbol{\mu}_o^t$
$\boldsymbol{y} = \boldsymbol{\mu}^t$
Class prediction scores

Figure 7.3: GMF Network. μ_e^t and μ_o^t are even and odd column nodes respectively where t indexes the layers, $\mu^t = \{\mu_e^t, \mu_o^t\}$. Network is initialized with unary network output $\mu^1 = \mathbf{r}$.

- **Even update layer:** This sublayer takes the output of previous layer as input, and updates the even column nodes using (7.2) while keeping the odd column nodes fixed.

- **Odd update layer:** This sublayer takes the output of even update layer as input, and updates the odd column nodes using (7.2) while keeping the even column nodes fixed.

As explained in the previous section, because of the bipartite graph structure, the update performed by each of the above sublayers is an optimal update. Hence, each layer of our GMF network is guaranteed to generate an output that is closer to the MAP solution compared to its input (unless the input itself is the MAP solution, in which case the output will be equal to the input).

Combining the unary, pairwise and GMF networks, we get the proposed Gaussian CRF network, which can be trained in an end-to-end fashion. The parameters of the network are the unary network parameters $\theta_u = \theta_u^{CNN}$, and the pairwise network parameters $\theta_p = \{\theta_p^{CNN}, \{\mathbf{f}_m\}, \mathbf{C} \succeq 0\}$. Note that since we use a fixed number of layers in our GMF network, the final output is not guaranteed to be the MAP solution of our Gaussian CRF model. However, since we train the entire network discriminatively in an end-to-end fashion, the unary and pairwise networks would learn to generate appropriate $\mathbf{r}_i$ and $\mathbf{W}_{ij}$ such that the output after a fixed number of mean field updates would be close to the desired output.

Note that the DeepLab network has three downsampling layers, and hence the size of its output is 1/8 times the input image size. We apply our Gaussian CRF model to this low resolution output and upsample the GMF network output to the input image resolution by using bilinear interpolation.

Discrete label assignment: Note that the final output at each pixel is a K-dimensional vector where K is the number of classes. Let $\mathbf{y}_i^* = [y_{i1}^*, \dots, y_{iK}^*]$ be the final output at i^{th} pixel. Then the predicted class label of i^{th} pixel is given by $\text{argmax}_k \; y_{ik}^*$.

7.4.1 Training

We train the proposed Gaussian CRF network discriminatively by minimizing the following loss function at each pixel

$$L\left(\mathbf{y}_i^*, l_i\right) = -\text{min}\left(0,\; y_{il_i}^* - \text{max}_{k \neq l_i}\; y_{ik}^* - T\right), \tag{7.6}$$

where l_i is the true class label. This loss function basically encourages the output associated with the true class to be greater than the output associated with all the other classes by a margin T.

We use standard backpropagation to compute the gradient of the network parameters. Here, we show how to backpropagate the loss derivatives through the layers of the proposed network. Please refer to Appendix C for detailed derivations.

Backpropagating through the odd update layer: Given the derivatives $dL/d\boldsymbol{\mu}_i^{out}$ of the loss function with respect to the output of an odd update layer, we can compute the derivatives of L with respect to its inputs $\mathbf{r}_i$, $\mathbf{W}_{ij}$ and $\boldsymbol{\mu}_j^{in}$ using

$$
\begin{aligned}
\frac{dL}{d\mathbf{r}_i} &= \begin{cases} \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1} \frac{dL}{d\boldsymbol{\mu}_i^{out}} & \text{if node } i \text{ is in an odd column} \\ 0 & \text{elsewise,} \end{cases} \\
\frac{dL}{d\mathbf{W}_{ij}} &= \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1} \frac{dL}{d\boldsymbol{\mu}_i^{out}} \left(\boldsymbol{\mu}_j^{in} - \boldsymbol{\mu}_i^{out}\right)^\top, \text{ for } i \text{ in odd columns,} \\
\frac{dL}{d\boldsymbol{\mu}_j^{in}} &= \begin{cases} \frac{dL}{d\boldsymbol{\mu}_j^{out}} + \sum_i \left(\mathbf{W}_{ij} \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1} \frac{dL}{d\boldsymbol{\mu}_i^{out}}\right) & \text{if node } j \text{ is in an even column} \\ 0 & \text{elsewise.} \end{cases}
\end{aligned}
\tag{7.7}
$$

Backpropagating through the similarity layer: Given the derivatives dL/ds_{ij} of the loss function with respect to the output of the similarity layer, we can compute the derivatives of L with respect to its input $\mathbf{z}_i$ and parameters $\mathbf{f}_m$ using

$$
\begin{aligned}
\frac{dL}{d\mathbf{z}_i} &= 2\left(\sum_{m=1}^{M} \mathbf{f}_m \mathbf{f}_m^\top\right)\left(\sum_j s_{ij} \frac{dL}{ds_{ij}} \left(\mathbf{z}_j - \mathbf{z}_i\right)\right), \\
\frac{dL}{d\mathbf{f}_m} &= -2\left(\sum_{ij} s_{ij} \frac{dL}{ds_{ij}} \left(\mathbf{z}_i - \mathbf{z}_j\right)\left(\mathbf{z}_i - \mathbf{z}_j\right)^\top\right)\mathbf{f}_m.
\end{aligned}
\tag{7.8}
$$

Backpropagating through the even update layer: Given the derivatives $dL/d\boldsymbol{\mu}_i^{out}$ of the loss function with respect to the output of an even update layer, we can compute the derivatives of L with respect to its inputs $\mathbf{r}_i$, $\mathbf{W}_{ij}$ and $\boldsymbol{\mu}_j^{in}$ using

$$\frac{dL}{d\mathbf{r}_i} = \begin{cases} \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1} \frac{dL}{d\boldsymbol{\mu}_i^{out}} & \text{if node } i \text{ is in an even column} \\ 0 & \text{elsewise,} \end{cases}$$

$$\frac{dL}{d\mathbf{W}_{ij}} = \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1} \frac{dL}{d\boldsymbol{\mu}_i^{out}} \left(\boldsymbol{\mu}_j^{in} - \boldsymbol{\mu}_i^{out}\right)^\top, \text{ for } i \text{ in even columns,}$$

$$\frac{dL}{d\boldsymbol{\mu}_j^{in}} = \begin{cases} \frac{dL}{d\boldsymbol{\mu}_j^{out}} + \sum_i \left(\mathbf{W}_{ij} \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1} \frac{dL}{d\boldsymbol{\mu}_i^{out}}\right) & \text{if node } j \text{ is in an odd column} \\ 0 & \text{elsewise.} \end{cases} \tag{7.9}$$

Backpropagating through the matrix generation layer: Given the derivatives $dL/d\mathbf{W}_{ij}$ of the loss function with respect to the output of the matrix generation layer, we can compute the derivatives of L with respect to its input s_{ij} and parameters $\mathbf{C}$ using

$$\frac{dL}{ds_{ij}} = \text{trace}\left(\left(\frac{dL}{d\mathbf{W}_{ij}}\right)^\top \mathbf{C}\right), \quad \frac{dL}{d\mathbf{C}} = \sum_{ij} s_{ij} \frac{dL}{d\mathbf{W}_{ij}}. \tag{7.10}$$

We skip the derivative formulas for CNNs since they are composed of standard layers. Note that we have a constrained optimization problem here due to the symmetry and positive semidefiniteness constraints on the parameter $\mathbf{C}$. We convert this constrained problem into an unconstrained one by parametrizing $\mathbf{C}$ as $\mathbf{C} = \mathbf{R}\mathbf{R}^\top$, where $\mathbf{R}$ is a lower triangular matrix.

7.5 Experimental Evaluation

We evaluate the proposed deep network using the PASCALVOC 2012 segmentation dataset [156], which consists of 20 object classes and one background class. The original dataset consists of 1464, 1449 and 1456 training, validation and test images, respectively. Similar to [2], we augment the training set with the additional annotations provided by [193], resulting in a total of 10,582 training images.

Parameters: In our Gaussian CRF model, each node was connected to every other node along both rows and columns (Figure 7.2) within a 23×23 spatial neighborhood. Note that since our Gaussian CRF model is applied to the CNN output whose resolution is 1/8 times the input resolution, the effective neighborhood size in the input image is 184×184. For our experiments, we used a five layer GMF network, which performs five full-image updates in the forward pass. During training, we used a value of 0.5 for the margin T used in our loss function. The number of filters M used in the similarity layer was set to be equal to the number of classes.

7.5.1 Training

We used the open source Caffe framework [194] for our experiments. We initialized both of our CNNs with the trained model provided by the authors of [2]. Note that this model was finetuned using only the PASCALVOC segmentation data starting from the ImageNet-trained VGG-16 model [6]. For training, we used stochastic gradient descent with a weight decay of 5×10^{-3} and a momentum of 0.9.

Pretraining: Before training the full Gaussian CRF network, we pre-trained the similarity layer and CNN of the pairwise network such that the output s_{ij} of the similarity layer is high for a pair of pixels that have the same class label and low for a pair of pixels that have different class labels. For pre-training, we used the following loss function for each pair of connected pixels:

$$L_{ij} = -\mathbb{1}[l_i = l_j]s_{ij} + \mathbb{1}[l_i \neq l_j]\min(0, s_{ij} - h), \tag{7.11}$$

where l_i and l_j are respectively the class labels of pixel i and j, and h is a threshold parameter. This loss function encourages s_{ij} to be high for similar pairs and below a threshold h for dissimilar pairs. The value of h was chosen as e^{-10}. For training, we used a mini-batch of 15 images and a starting learning rate of 10^{-3} for the similarity layer parameters $\{\mathbf{f}_m\}$ and 10^{-4} for the CNN parameters θ_p^{CNN}. After training for 8000 iterations, we multiplied the learning rate of the similarity layer parameters by 0.1 and trained for additional 5000 iterations.

Finetuning: After the pre-training stage, we finetuned the entire Gaussian CRF network using a mini-batch of 5 images and a starting learning rate of 10^{-2} for all parameters except θ_u^{CNN}, for which we used a small learning rate of 10^{-6}. Since the Unary DeepLab CNN was trained by [2] using PASCALVOC segmentation data, it was already close to a good local minima. Hence, we finetuned it with a small learning rate. After training for 6000 iterations, we multiplied the learning rate by 0.01 and trained for additional 25000 iterations.

Table 7.1: Comparison with various approaches on PASCALVOC 2012 test set (when trained using ImageNet and PASCALVOC data).

Method	bkg	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbk	person	plant	sheep	sofa	train	tv	mean
MSRA-CFM [174]	87.7	75.7	26.7	69.5	48.8	65.6	81.0	69.2	73.3	30.0	68.7	51.5	69.1	68.1	71.7	67.5	50.4	66.5	44.4	58.9	53.5	61.8
FCN-8s [31]	91.2	76.8	34.2	68.9	49.4	60.3	75.3	74.7	77.6	21.4	62.5	46.8	71.8	63.9	76.5	73.9	45.2	72.4	37.4	70.9	55.1	62.2
Hypercolumns [160]	89.3	68.7	33.5	69.8	51.3	70.2	81.1	71.9	74.9	23.9	60.6	46.9	72.1	68.3	74.5	72.9	52.6	64.4	45.4	64.9	57.4	62.6
DeepLab CNN [2]	91.6	78.7	51.5	75.8	59.5	61.9	82.5	76.6	79.4	26.9	67.7	54.7	74.3	70.0	79.8	77.3	52.6	75.2	46.6	66.9	57.3	67.0
ZoomOut [161]	91.1	85.6	37.3	83.2	62.5	66.0	85.1	80.7	84.9	27.2	73.2	57.5	78.1	79.2	81.1	77.1	53.6	74.0	49.2	71.7	63.3	69.6
Deep message passing [179]	93.9	90.1	38.6	77.8	61.3	74.3	89.0	83.4	83.3	36.2	80.2	56.4	81.2	81.4	83.1	82.9	59.2	83.4	54.3	80.6	70.8	73.4
Approaches that use CNNs and CRFs																						
DeconvNet + CRF [177]	92.9	87.8	41.9	80.6	63.9	67.3	88.1	78.4	81.3	25.9	73.7	61.2	72.0	77.0	79.9	78.7	59.5	78.3	55.0	75.2	61.5	70.5
object clique potentials [165]	92.8	80.0	53.8	80.8	62.5	64.7	87.0	78.5	83.0	29.0	82.0	60.3	76.3	78.4	83.0	79.8	57.0	80.0	53.1	70.1	63.1	71.2
DeepLab CNN-CRF [2]	93.3	84.4	54.5	81.5	63.6	65.9	85.1	79.1	83.4	30.7	74.1	59.8	79.0	76.1	83.2	80.8	59.7	82.2	50.4	73.1	63.7	71.6
CRF-RNN [152]	94.0	87.5	39.0	79.7	64.2	68.3	87.6	80.8	84.4	30.4	78.2	60.4	80.5	77.8	83.1	80.6	59.5	82.8	47.8	78.3	67.1	72.0
DeconvNet + FCN + CRF [177]	93.1	89.9	39.3	79.7	63.9	68.2	87.4	81.2	86.1	28.5	77.0	62.0	79.0	80.3	83.6	80.2	58.8	83.4	54.3	80.7	65.0	72.5
Proposed Gaussian CRF network	93.4	85.2	43.9	83.3	65.2	68.3	89.0	82.7	85.3	31.1	79.5	63.3	80.5	79.3	85.5	81.0	60.5	85.5	52.0	77.3	65.1	73.2

7.5.2 Results

For quantitative evaluation, we use the standard mean intersection-over-union measure (averaged across the 21 classes). Table 7.1 compares the proposed Gaussian CRF network with state-of-the-art semantic segmentation approaches on the challenging PASCALVOC 2012 test set. We can infer the following from these results:

- The proposed Gaussian CRF network performs significantly (6.2 points) better than the DeepLab CNN, which was used for initializing the unary network. This shows that Gaussian CRFs can be successfully used for discrete labeling problems even though they are continuous models.

- The proposed approach outperforms several recent approaches that use discrete CRF models with CNNs. This shows that, despite being a continuous model, the Gaussian CRF model can be a strong competitor to discrete CRFs in discrete labeling tasks.

Figure 7.4 provides a visual comparison of the proposed approach with DeepLab CNN (which is same as our unary network) and DeepLab CNN + discrete CRF. As we can see, the proposed Gaussian CRF model is able to correct the errors made by the unary network, and also produces more accurate segmentation maps compared to the discrete CRF-based DeepLab approach.

Computation time: The proposed Gaussian CRF network takes around 0.6 seconds to segment a 505×505 image on an NVIDIA TITAN GPU.

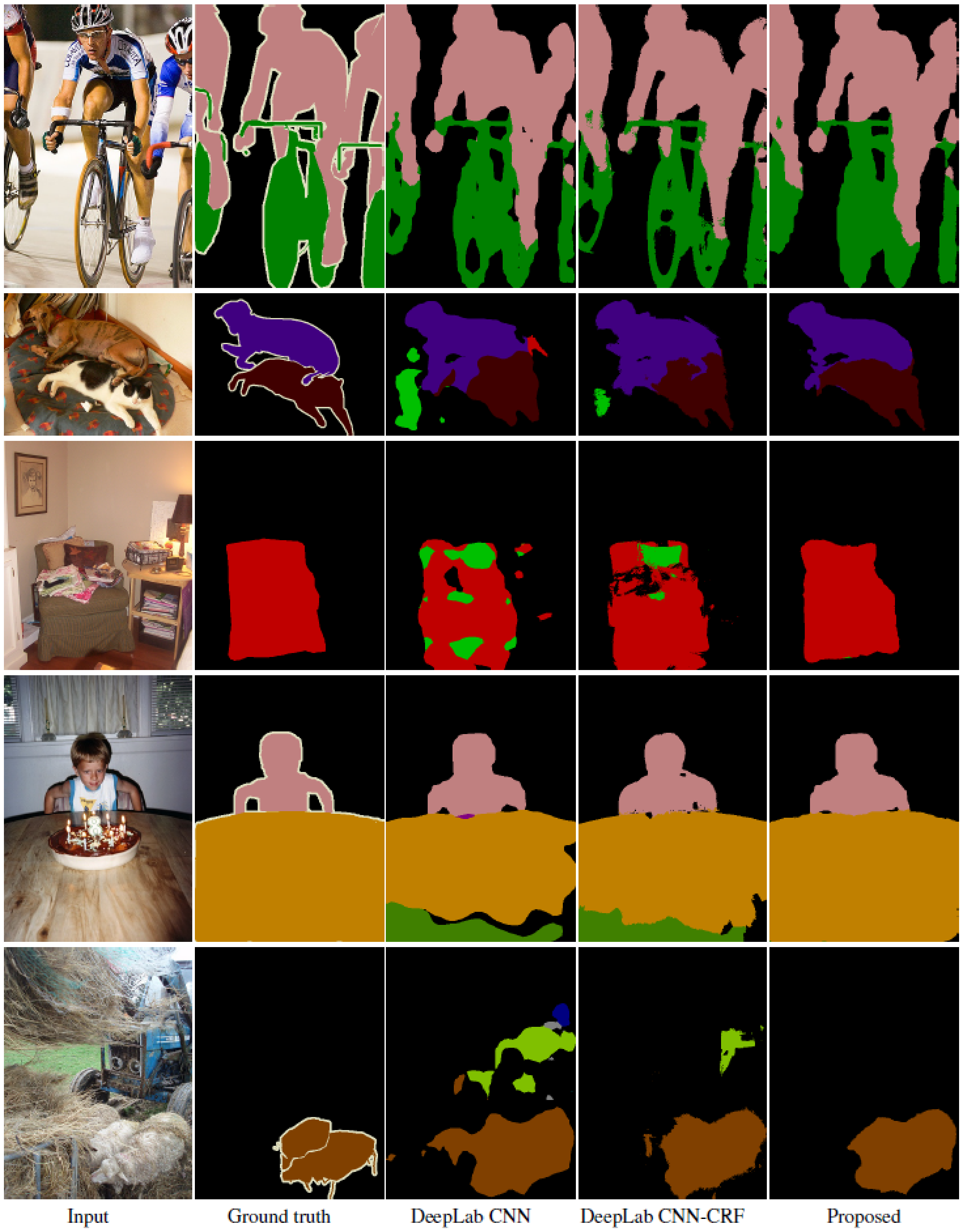

Figure 7.4: Comparison of the proposed approach with DeepLab CNN [2] and DeepLab CNN + discrete CRF [2].

7.6 Conclusions

In this chapter, we proposed a Gaussian CRF model for the discrete labeling task of semantic segmentation. We proposed a novel deep network for Gaussian CRF inference, which we refer to as GMF network, by unfolding a fixed number of Gaussian mean field inference steps. By combining this GMF network with CNNs, we proposed an end-to-end trainable Gaussian CRF network. When trained discriminatively, the proposed Gaussian CRF network outperformed various recent discrete CRF-based semantic segmentation approaches on the challenging PASCALVOC 2012 segmentation dataset. Our results suggest that, despite being a continuous model, Gaussian CRF can be successfully used for discrete labeling tasks.

Chapter 8: Conclusions and Directions for Future Work

8.1 Summary

In this dissertation, we focused on two important factors that are critical in the design of computer vision algorithms, namely *representation* and *context modeling*, and made novel contributions by proposing new 3D geometry-based representations for recognizing human actions from skeletal sequences, and introducing Gaussian conditional random field model-based deep network architectures that explicitly model the spatial context by considering the interactions among the output variables. In addition, we also proposed a kernel learning-based framework for the classification of manifold features such as linear subspaces and covariance matrices.

In the first part of this dissertation, we introduced a family of body part-based 3D skeletal representations for human action recognition, which we refer to as R3DG features. The proposed representations explicitly model the relative 3D geometry between various body parts using rigid body transformations. We represented 3D rigid body transformations using $SE(3)$, $SO(3) \otimes \mathcal{R}^3$, $\mathcal{UQ} \otimes \mathcal{R}^3$, and $\mathcal{UD}$, resulting in four different R3DG features. We also introduced two scale-invariant R3DG features by using only the 3D rotations between various body parts. Using the proposed representations, we modeled the human actions as curves in R3DG feature spaces. Finally, we performed action recognition by classifying these curves using a combination of DTW, FTP representation and SVM classifier. We experimentally showed that the proposed R3DG features perform better than various existing skeletal representations, and the proposed action recognition approach outperforms various existing skeleton-based action recognition approaches.

In the second part of this dissertation, we used rolling maps for flattening $SO(3)$ to perform human action recognition from 3D skeletal data. We represented each human skeleton as a point in the Lie group $SO(3) \otimes \ldots \otimes SO(3)$ using the relative 3D rotations between all pairs of body parts. Using this skeletal representation, we represented human actions as curves in $SO(3) \otimes \ldots \otimes SO(3)$. For each action category, we computed a nominal curve and warped all the action curves to this nominal using DTW. Then, we rolled $SO(3) \otimes \ldots \otimes SO(3)$ over its Lie algebra along the nominal curves and unwrapped all the action curves onto the Lie algebra. Finally, we represented the unwrapped curves using either the concatenated representation or the FTP representation and classified them using a one-vs-all linear SVM classifier. We experimentally showed that flattening $SO(3)$ by unwrapping while rolling performs better than flattening $SO(3)$ by using logarithm map at a single point. In this part of the dissertation, we also showed how to compute a piecewise smooth rolling map such that the corresponding rolling curve passes through a given set of points in $SO(3)$ at given instances of time.

In the third part of this dissertation, we introduced a general extrinsic framework for the classification of manifold features using kernel learning approach. We proposed two criteria for learning a good kernel-classifier combination for manifold features. In the case of SVM classifier, based on the proposed criteria, we formulated the problem of learning a good kernel-classifier combination as a convex optimization problem, and solved it efficiently following the multiple kernel learning approach. We evaluated the proposed approach for the image set-based classification task using linear subspaces and covariance features, and obtained superior performance compared to other relevant approaches.

In the fourth part of this dissertation, we proposed a new end-to-end trainable deep network architecture for image denoising based on a Gaussian CRF model. The proposed network consists of a parameter generation network that generates appropriate potential function parameters based on the input image, and an inference network that performs approximate Gaussian CRF inference. Unlike existing discriminative denoising approaches that train a separate model for each noise level, the proposed network can handle a range of noise levels as it explicitly models the input noise variance. We achieved results on par with the state-of-the-art by training two deep Gaussian CRF networks, one for low input noise levels and one for high input noise levels.

In the last part of this dissertation, we proposed to use a Gaussian CRF model for the discrete labeling task of semantic segmentation. We proposed a novel deep network for Gaussian CRF inference, which we refer to as GMF network, by unfolding a fixed number of Gaussian mean field inference steps. By combining this GMF network with CNNs, we proposed an end-to-end trainable Gaussian CRF network for semantic segmentation. When trained discriminatively, the proposed Gaussian CRF network outperformed various recent discrete CRF-based semantic segmentation approaches on the challenging PASCALVOC 2012 segmentation dataset. Our results suggest that, despite being a continuous model, Gaussian CRF can be successfully used for discrete labeling tasks.

8.2 Directions for Future Work

In Chapter 3, we used the relative 3D geometry between all pairs of body parts in the skeletal representation. However, each action is usually characterized by the interactions of a specific set of body parts. Hence, using feature selection approaches such as multiple kernel learning to automatically identify the set of body parts that differentiates a given action from the rest could further improve the action recognition performance.

In Chapter 4, we used the concept of rolling maps for mapping temporal sequences from the Lie group $SO(3)$ to its Lie algebra. Though we focused on $SO(3)$ in this dissertation, the rolling map is a general concept that can be used with any Riemannian manifold. So, the proposed approach can also be used for the classification of time series data on other manifolds like Grassmann manifold and the manifold of SPD matrices. While we focused only on actions performed by a single person in this dissertation, the proposed representations and action recognition approaches can also be used for classifying multi-person interactions.

In Chapter 5, we focused on the SVM classifier and formulated the problem of learning a good kernel-classifier combination as a convex optimization problem. The proposed framework can also be extended to discriminant analysis as kernel learning can be formulated as a convex optimization problem in the case of Fisher discriminant analysis [195,196]. Another possible direction of future work is to explore more sophisticated regularizers that can make use of the underlying manifold structure instead of the simple distance-preserving constraints used in this dissertation.

In Chapter 6, we proposed a Gaussian CRF-based deep network architecture for image denoising. Although we focused on image denoising in this dissertation, the proposed network design strategy can also be used for other full-image inference tasks like super-resolution, depth estimation, etc. We used half quadratic splitting and Gaussian mean field based inference approaches to design our inference networks in Chapters 6 and 7, respectively. Instead, one can also consider other inference approaches such as belief propagation.

Appendix A: Gaussian Mean Field Inference

In this appendix, we derive the Gaussian mean field inference update equation for the Gaussian CRF model presented in Section 7.3. We modeled the conditional probability density $P(\mathbf{y}|\mathbf{X})$ as a Gaussian distribution given by

$$\begin{aligned} P(\mathbf{y}|\mathbf{X}) &\propto \exp\left\{-\frac{1}{2}\, E(\mathbf{y}|\mathbf{X})\right\}, \text{ where} \\ E\left(\mathbf{y}|\mathbf{X}\right) &= \sum_i \|\mathbf{y}_i - \mathbf{r}_i\|_2^2 + \sum_{ij} \left(\mathbf{y}_i - \mathbf{y}_j\right)^\top \mathbf{W}_{ij} \left(\mathbf{y}_i - \mathbf{y}_j\right) \\ &= \sum_i \mathbf{y}_i^\top \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right) \mathbf{y}_i - 2\sum_i \mathbf{r}_i^\top \mathbf{y}_i + \sum_i \mathbf{r}_i^\top \mathbf{r}_i - 2\sum_{ij} \mathbf{y}_i^\top \mathbf{W}_{ij} \mathbf{y}_j. \end{aligned} \tag{A.1}$$

The standard mean field approach approximates the joint Gaussian distribution $P(\mathbf{y}|\mathbf{X})$ using a simpler Gaussian distribution $Q(\mathbf{y}|\mathbf{X})$ which can be written as a product of independent marginals, i.e, $Q(\mathbf{y}|\mathbf{X}) = \prod_i Q_i(\mathbf{y}_i|\mathbf{X})$, where $Q(\mathbf{y}_i|\mathbf{X})$ is a Gaussian distribution with mean $\boldsymbol{\mu}_i \in \mathcal{R}^K$ and covariance $\boldsymbol{\Sigma}_i \in \mathcal{R}^{K\times K}$. The parameters $\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$ of Q are obtained by minimizing the KL-divergence between the distributions Q and P.

$$\begin{aligned} KL(Q||P) &= \int Q(\mathbf{y}|\mathbf{X})\, \log\left[Q(\mathbf{y}|\mathbf{X})\right] - \int Q(\mathbf{y}|\mathbf{X})\, \log\left[P(\mathbf{y}|\mathbf{X})\right] \\ &= \sum_i \int Q_i(\mathbf{y}_i|\mathbf{X})\, \log\left[Q_i(\mathbf{y}_i|\mathbf{X})\right] - \int Q(\mathbf{y}|\mathbf{X})\, \log\left[P(\mathbf{y}|\mathbf{X})\right] \qquad \text{(A.2)} \\ &= -\sum_i \frac{1}{2}\log\left[(2\pi e)^K|\boldsymbol{\Sigma}_i|\right] - \int Q(\mathbf{y}|\mathbf{X})\, \log\left[P(\mathbf{y}|\mathbf{X})\right]. \end{aligned}$$

$$
\begin{aligned}
\{\boldsymbol{\mu}_i^*, \boldsymbol{\Sigma}_i^*\} &= \underset{\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}}{\operatorname{argmin}} \quad KL(Q||P) \\
&= \underset{\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}}{\operatorname{argmin}} \quad -\sum_i \frac{1}{2}\log\left[(2\pi e)^K |\boldsymbol{\Sigma}_i|\right] - \int Q(\mathbf{y}|\mathbf{X}) \log\left[P(\mathbf{y}|\mathbf{X})\right] \\
&= \underset{\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}}{\operatorname{argmin}} \quad -\sum_i \log\left[|\boldsymbol{\Sigma}_i|\right] + \sum_i \int Q(\mathbf{y}|\mathbf{X})\ \mathbf{y}_i^\top \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right) \mathbf{y}_i \\
&\qquad - 2\sum_i \int Q(\mathbf{y}|\mathbf{X})\ \mathbf{r}_i^\top \mathbf{y}_i - 2\sum_{ij} \int Q(\mathbf{y}|\mathbf{X})\ \mathbf{y}_i^\top \mathbf{W}_{ij} \mathbf{y}_j \\
&= \underset{\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}}{\operatorname{argmin}} \quad -\sum_i \log\left[|\boldsymbol{\Sigma}_i|\right] + \sum_i \mathbb{E}\left[\mathbf{y}_i^\top \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right) \mathbf{y}_i\right] \\
&\qquad - 2\sum_i \mathbb{E}\left[\mathbf{r}_i^\top \mathbf{y}_i\right] - 2\sum_{ij} \mathbb{E}\left[\mathbf{y}_i^\top \mathbf{W}_{ij} \mathbf{y}_j\right] \\
&= \underset{\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}}{\operatorname{argmin}} \quad -\sum_i \log\left[|\boldsymbol{\Sigma}_i|\right] + \sum_i \mathbb{E}\left[\operatorname{trace}\left(\mathbf{y}_i \mathbf{y}_i^\top \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right)\right)\right] \\
&\qquad - 2\sum_i \mathbb{E}\left[\mathbf{r}_i^\top \mathbf{y}_i\right] - 2\sum_{ij} \mathbb{E}\left[\operatorname{trace}\left(\mathbf{y}_j \mathbf{y}_i^\top \mathbf{W}_{ij}\right)\right] \\
&= \underset{\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}}{\operatorname{argmin}} \quad -\sum_i \log\left[|\boldsymbol{\Sigma}_i|\right] + \sum_i \operatorname{trace}\left(\mathbb{E}\left[\mathbf{y}_i \mathbf{y}_i^\top\right] \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right)\right) \\
&\qquad - 2\sum_i \mathbb{E}\left[\mathbf{r}_i^\top \mathbf{y}_i\right] - 2\sum_{ij} \operatorname{trace}\left(\mathbb{E}\left[\mathbf{y}_j \mathbf{y}_i^\top\right] \mathbf{W}_{ij}\right) \\
&= \underset{\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}}{\operatorname{argmin}} \quad -\sum_i \log\left[|\boldsymbol{\Sigma}_i|\right] + \sum_i \operatorname{trace}\left(\left(\boldsymbol{\Sigma}_i + \boldsymbol{\mu}_i \boldsymbol{\mu}_i^\top\right) \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right)\right) \\
&\qquad - 2\sum_i \mathbf{r}_i^\top \boldsymbol{\mu}_i - 2\sum_{ij} \operatorname{trace}\left(\boldsymbol{\mu}_j \boldsymbol{\mu}_i^\top \mathbf{W}_{ij}\right)
\end{aligned} \tag{A.3}
$$

Note that in the last step, we have used the fact that $\mathbf{y}_i$ and $\mathbf{y}_j$ are independent under the distribution Q. From (A.3) we have,

$$
\boldsymbol{\Sigma}_i^* = \underset{\boldsymbol{\Sigma}_i}{\operatorname{argmin}} \quad \operatorname{trace}\left(\boldsymbol{\Sigma}_i \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right)\right) - \log\left[|\boldsymbol{\Sigma}_i|\right] \tag{A.4}
$$

Note that (A.4) is a convex problem. Differentiating the cost function and setting the gradient to zero, we get $\boldsymbol{\Sigma}_i^* = \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right)^{-1}$. From (A.3) we have,

$$\begin{aligned}\boldsymbol{\mu}_i^* &= \underset{\boldsymbol{\mu}_i}{\operatorname{argmin}} \;\; \operatorname{trace}\left(\boldsymbol{\mu}_i\boldsymbol{\mu}_i^\top\left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right)\right) - 2\mathbf{r}_i^\top\boldsymbol{\mu}_i - 2\sum_j \operatorname{trace}\left(\boldsymbol{\mu}_j^*\boldsymbol{\mu}_i^\top\mathbf{W}_{ij}\right) \\ &= \underset{\boldsymbol{\mu}_i}{\operatorname{argmin}} \;\; \boldsymbol{\mu}_i^\top\left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right)\boldsymbol{\mu}_i - 2\mathbf{r}_i^\top\boldsymbol{\mu}_i - 2\boldsymbol{\mu}_i^\top\left(\sum_j \mathbf{W}_{ij}\boldsymbol{\mu}_j^*\right)\end{aligned} \tag{A.5}$$

Note that (A.5) is a convex problem. Differentiating the cost function and setting the gradient to zero, we get

$$\boldsymbol{\mu}_i^* = \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right)^{-1}\left(\mathbf{r}_i + \sum_j \mathbf{w}_{ij}\boldsymbol{\mu}_j^*\right). \tag{A.6}$$

Hence, for the Gaussian distribution in (A.1), the mean field update for computing the means $\{\boldsymbol{\mu}_i\}$ is given by

$$\boldsymbol{\mu}_i \leftarrow \left(\boldsymbol{I}_K + \sum_j \mathbf{W}_{ij}\right)^{-1}\left(\mathbf{r}_i + \sum_j \mathbf{W}_{ij}\boldsymbol{\mu}_j\right). \tag{A.7}$$

Appendix B: Deep Gaussian CRF Network for Image Denoising - Backpropagation

In this appendix, we derive the formulas used for backpropagating the loss derivatives through the layers of the deep Gaussian CRF network presented in Section 6.4. Let L be the final loss function.

B.1 Backpropagation - Combination Layer

Forward step:

$$\mathbf{\Sigma}_{ij} = \sum_k \gamma_{ij}^k \mathbf{\Psi}_k. \tag{B.1}$$

Backward step:

$$\begin{aligned}\frac{dL}{d\gamma_{ij}^k} &= \sum_{pq} \frac{dL}{d\mathbf{\Sigma}_{ij}(p,q)} \frac{d\mathbf{\Sigma}_{ij}(p,q)}{d\gamma_{ij}^k} \\ &= \sum_{pq} \frac{dL}{d\mathbf{\Sigma}_{ij}(p,q)} \mathbf{\Psi}_k(p,q) = \text{trace}\left(\mathbf{\Psi}_k^\top \frac{dL}{d\mathbf{\Sigma}_{ij}}\right).\end{aligned} \tag{B.2}$$

$$\begin{aligned}\frac{dL}{d\mathbf{\Psi}_k(p,q)} &= \sum_{ij} \frac{dL}{d\mathbf{\Sigma}_{ij}(p,q)} \frac{d\mathbf{\Sigma}_{ij}(p,q)}{d\mathbf{\Psi}_k(p,q)} \\ &= \sum_{ij} \frac{dL}{d\mathbf{\Sigma}_{ij}(p,q)} \gamma_{ij}^k \implies \frac{dL}{d\mathbf{\Psi}_k} = \sum_{ij} \gamma_{ij}^k \frac{dL}{d\mathbf{\Sigma}_{ij}}.\end{aligned} \tag{B.3}$$

B.2 Backpropagation - Quadratic Layer

Forward step:

$$s_{ij}^k = -\frac{1}{2}\bar{\mathbf{x}}_{ij}^\top \left(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2}\right)^{-1} \bar{\mathbf{x}}_{ij} + b_k. \tag{B.4}$$

Backward step:

$$\frac{dL}{db_k} = \sum_{ij} \frac{dL}{ds_{ij}^k} \frac{ds_{ij}^k}{db_k} = \sum_{ij} \frac{dL}{ds_{ij}^k}. \tag{B.5}$$

$$\frac{dL}{d\bar{\mathbf{x}}_{ij}} = \sum_k \frac{dL}{ds_{ij}^k} \frac{ds_{ij}^k}{d\bar{\mathbf{x}}_{ij}} = -\sum_k \frac{dL}{ds_{ij}^k} \left(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2}\right)^{-1} \bar{\mathbf{x}}_{ij}. \tag{B.6}$$

$$\begin{aligned}
\frac{dL}{d\mathbf{W}_k(p,q)} &= \sum_{ij} \frac{dL}{ds_{ij}^k} \; \frac{ds_{ij}^k}{d\mathbf{W}_k(p,q)} \\
&= -\frac{1}{2} \sum_{ij} \frac{dL}{ds_{ij}^k} \; \bar{\mathbf{x}}_{ij}^\top \; \frac{d(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})^{-1}}{d\mathbf{W}_k(p,q)} \; \bar{\mathbf{x}}_{ij} \\
&= \frac{1}{2} \sum_{ij} \frac{dL}{ds_{ij}^k} \; \bar{\mathbf{x}}_{ij}^\top \; (\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})^{-1} \; \frac{d(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})}{d\mathbf{W}_k(p,q)} \; (\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})^{-1} \; \bar{\mathbf{x}}_{ij} \\
&= \frac{1}{2} \sum_{ij} \frac{dL}{ds_{ij}^k} \left[(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})^{-1} \; \bar{\mathbf{x}}_{ij} \; \bar{\mathbf{x}}_{ij}^\top \; (\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})^{-1}\right](p,q).
\end{aligned} \tag{B.7}$$

From (B.7), we have

$$\begin{aligned}
\frac{dL}{d\mathbf{W}_k} &= \frac{1}{2} \sum_{ij} \frac{dL}{ds_{ij}^k} \left[(\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})^{-1} \; \bar{\mathbf{x}}_{ij} \; \bar{\mathbf{x}}_{ij}^\top \; (\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})^{-1}\right] \\
&= (\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})^{-1} \; \left(\sum_{ij} \frac{dL}{ds_{ij}^k} \frac{\bar{\mathbf{x}}_{ij} \; \bar{\mathbf{x}}_{ij}^\top}{2}\right) \; (\mathbf{W}_k + \sigma^2 \mathbf{I}_{d^2})^{-1}.
\end{aligned} \tag{B.8}$$

B.3 Backpropagation - Patch Inference Layer

Forward step:

$$\mathbf{z}_{ij} = \left(\mathbf{I}_{d^2} - \mathbf{G}^\top \left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\mathbf{G}^\top\right)^{-1} \mathbf{G}\right) \mathbf{y}_{ij}. \tag{B.9}$$

Backward step:

Let $\mathbf{A}_{ij} = \mathbf{I}_{d^2} - \mathbf{G}^\top \left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\mathbf{G}^\top\right)^{-1} \mathbf{G}$. Let $\mathbb{I}_{pq}$ be a matrix with (p, q) element as one and all other elements as zero. Note that the matrix $\mathbf{G} = \mathbf{I}_{d^2} - \frac{1}{d^2}\mathbf{1}$ satisfies $\mathbf{G}\mathbf{G}^\top = \mathbf{G}$.

$$\begin{aligned}
\mathbf{z}_{ij} = \mathbf{A}_{ij}\mathbf{y}_{ij} \implies \frac{dL}{d\mathbf{y}_{ij}} &= \mathbf{A}_{ij}^\top \frac{dL}{d\mathbf{z}_{ij}} \\
&= \left(\mathbf{I}_{d^2} - \mathbf{G}^\top \left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\mathbf{G}^\top\right)^{-1} \mathbf{G}\right) \frac{dL}{d\mathbf{z}_{ij}} \\
&= \left(\mathbf{I}_{d^2} - \mathbf{G}^\top \left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\right)^{-1} \mathbf{G}\right) \frac{dL}{d\mathbf{z}_{ij}}.
\end{aligned} \tag{B.10}$$

$$\mathbf{z}_{ij} = \mathbf{A}_{ij}\mathbf{y}_{ij} \implies \frac{dL}{d\mathbf{A}_{ij}} = \frac{dL}{d\mathbf{z}_{ij}}\mathbf{y}_{ij}^\top. \tag{B.11}$$

$$\frac{dL}{d\mathbf{\Sigma}_{ij}(p,q)} = \sum_{r,s} \frac{dL}{d\mathbf{A}_{ij}(r,s)} \frac{d\mathbf{A}_{ij}(r,s)}{d\mathbf{\Sigma}_{ij}(p,q)} = \text{trace}\left(\left(\frac{dL}{d\mathbf{A}_{ij}}\right)^\top \frac{d\mathbf{A}_{ij}}{d\mathbf{\Sigma}_{ij}(p,q)}\right). \tag{B.12}$$

$$\begin{aligned}
\frac{d\mathbf{A}_{ij}}{d\mathbf{\Sigma}_{ij}(p,q)} &= -\mathbf{G}^\top \ \frac{d\left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\mathbf{G}^\top\right)^{-1}}{d\mathbf{\Sigma}_{ij}(p,q)} \ \mathbf{G} \\
&= \mathbf{G}^\top \ \left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\mathbf{G}^\top\right)^{-1} \ \frac{d\left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\mathbf{G}^\top\right)}{d\mathbf{\Sigma}_{ij}(p,q)} \ \left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\mathbf{G}^\top\right)^{-1} \ \mathbf{G} \\
&= \mathbf{G}^\top \ \left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\mathbf{G}^\top\right)^{-1} \ \left(\beta \mathbb{I}_{pq}\right) \ \left(\beta \mathbf{\Sigma}_{ij} + \mathbf{G}\mathbf{G}^\top\right)^{-1} \ \mathbf{G}.
\end{aligned} \tag{B.13}$$

From (B.12) and (B.13), we have

$$\begin{aligned}\frac{dL}{d\boldsymbol{\Sigma}_{ij}(p,q)} &= \operatorname{trace}\left(\left(\frac{dL}{d\mathbf{A}_{ij}}\right)^{\top}\left(\mathbf{G}^{\top}\ \left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\mathbf{G}^{\top}\right)^{-1}\ (\beta\mathbb{I}_{pq})\ \left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\mathbf{G}^{\top}\right)^{-1}\ \mathbf{G}\right)\right)\\ &= \operatorname{trace}\left(\left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\mathbf{G}^{\top}\right)^{-1}\ \mathbf{G}\ \left(\frac{dL}{d\mathbf{A}_{ij}}\right)^{\top}\mathbf{G}^{\top}\ \left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\mathbf{G}^{\top}\right)^{-1}\ (\beta\mathbb{I}_{pq})\right).\end{aligned} \tag{B.14}$$

From (B.11) and (B.14), we have

$$\begin{aligned}\frac{dL}{d\boldsymbol{\Sigma}_{ij}} &= \beta\left(\left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\mathbf{G}^{\top}\right)^{-1}\ \mathbf{G}\ \left(\frac{dL}{d\mathbf{A}_{ij}}\right)^{\top}\mathbf{G}^{\top}\ \left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\mathbf{G}^{\top}\right)^{-1}\right)^{\top}\\ &= \beta\left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\mathbf{G}^{\top}\right)^{-1}\ \mathbf{G}\ \left(\frac{dL}{d\mathbf{A}_{ij}}\right)\ \mathbf{G}^{\top}\ \left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\mathbf{G}^{\top}\right)^{-1}\\ &= \beta\left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\right)^{-1}\ \mathbf{G}\ \frac{dL}{d\mathbf{z}_{ij}}\mathbf{y}_{ij}^{\top}\ \mathbf{G}^{\top}\ \left(\beta\boldsymbol{\Sigma}_{ij}+\mathbf{G}\right)^{-1}.\end{aligned} \tag{B.15}$$

Appendix C: Gaussian CRF Network for Semantic Segmentation - Backpropagation

In this appendix, we derive the formulas used for backpropagating the loss derivatives through the layers of the Gaussian CRF network presented in Section 7.4. Let L be the final loss function.

C.1 Backpropagation - Matrix Generation Layer

Forward step:

$$\mathbf{W}_{ij} = s_{ij}\mathbf{C}. \tag{C.1}$$

Backward step:

$$\begin{aligned} \frac{dL}{ds_{ij}} &= \sum_{pq} \frac{dL}{d\mathbf{W}_{ij}(p,q)} \frac{d\mathbf{W}_{ij}(p,q)}{ds_{ij}} \\ &= \sum_{pq} \frac{dL}{d\mathbf{W}_{ij}(p,q)} \mathbf{C}(p,q) = \operatorname{trace}\left(\left(\frac{dL}{d\mathbf{W}_{ij}}\right)^{\top} \mathbf{C}\right). \end{aligned} \tag{C.2}$$

$$\begin{aligned} \frac{dL}{d\mathbf{C}(p,q)} &= \sum_{ij} \frac{dL}{d\mathbf{W}_{ij}(p,q)} \frac{d\mathbf{W}_{ij}(p,q)}{d\mathbf{C}(p,q)} \\ &= \sum_{ij} \frac{dL}{d\mathbf{W}_{ij}(p,q)} s_{ij} \implies \frac{dL}{d\mathbf{C}} = \sum_{ij} s_{ij} \frac{dL}{d\mathbf{W}_{ij}}. \end{aligned} \tag{C.3}$$

C.2 Backpropagation - Similarity Layer

Forward step:

$$s_{ij} = e^{-\sum_{m=1}^{M}\left(\mathbf{f}_m^\top \mathbf{z}_i - \mathbf{f}_m^\top \mathbf{z}_j\right)^2}. \tag{C.4}$$

Backward step:

$$\begin{aligned}\frac{dL}{d\mathbf{z}_i} = \sum_j \frac{dL}{ds_{ij}}\frac{ds_{ij}}{d\mathbf{z}_i} &= \sum_j \frac{dL}{ds_{ij}}\left(-2s_{ij}\sum_{m=1}^{M}\frac{d\left(\mathbf{f}_m^\top \mathbf{z}_i\right)}{d\mathbf{z}_i}\left(\mathbf{f}_m^\top \mathbf{z}_i - \mathbf{f}_m^\top \mathbf{z}_j\right)\right)\\ &= \sum_j 2s_{ij}\frac{dL}{ds_{ij}}\sum_{m=1}^{M}\mathbf{f}_m\mathbf{f}_m^\top\left(\mathbf{z}_j - \mathbf{z}_i\right)\\ &= 2\left(\sum_{m=1}^{M}\mathbf{f}_m\mathbf{f}_m^\top\right)\left(\sum_j s_{ij}\frac{dL}{ds_{ij}}\left(\mathbf{z}_j - \mathbf{z}_i\right)\right).\end{aligned} \tag{C.5}$$

$$\begin{aligned}\frac{dL}{d\mathbf{f}_m} = \sum_{ij} \frac{dL}{ds_{ij}}\frac{ds_{ij}}{d\mathbf{f}_m} &= \sum_{ij} \frac{dL}{ds_{ij}}\left(-2s_{ij}\frac{d\left(\mathbf{f}_m^\top\left(\mathbf{z}_i - \mathbf{z}_j\right)\right)}{d\mathbf{f}_m}\left(\mathbf{z}_i - \mathbf{z}_j\right)^\top \mathbf{f}_m\right)\\ &= \sum_{ij} \frac{dL}{ds_{ij}}\left(-2s_{ij}\left(\mathbf{z}_i - \mathbf{z}_j\right)\left(\mathbf{z}_i - \mathbf{z}_j\right)^\top \mathbf{f}_m\right)\\ &= -2\left(\sum_{ij} s_{ij}\frac{dL}{ds_{ij}}\left(\mathbf{z}_i - \mathbf{z}_j\right)\left(\mathbf{z}_i - \mathbf{z}_j\right)^\top\right)\mathbf{f}_m.\end{aligned} \tag{C.6}$$

C.3 Backpropagation - Odd Update Layer

Forward step:

$$\boldsymbol{\mu}_i^{out} = \begin{cases}\mathbf{A}\left(\mathbf{r}_i + \sum_j \mathbf{W}_{ij}\boldsymbol{\mu}_j^{in}\right) & \text{if node } i \text{ is in an odd column}\\ \boldsymbol{\mu}_i^{in} & \text{elsewise,}\end{cases} \tag{C.7}$$

where $\mathbf{A} = \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1}$.

Backward step:

If node i is in an even column, then $\mathbf{r}_i$ does not play a role in the forward step and hence $dL/d\mathbf{r}_i = 0$. If node i is in an odd column, then

$$\begin{aligned}\frac{dL}{d\mathbf{r}_i} = \sum_p \frac{dL}{d\boldsymbol{\mu}_i^{out}(p)} \frac{d\boldsymbol{\mu}_i^{out}(p)}{d\mathbf{r}_i} &= \sum_p \frac{dL}{d\boldsymbol{\mu}_i^{out}(p)} \frac{d\left(\mathbf{A}(p,:)\mathbf{r}_i\right)}{d\mathbf{r}_i} \\ &= \sum_p \frac{dL}{d\boldsymbol{\mu}_i^{out}(p)} \mathbf{A}^\top(:,p) \\ &= \mathbf{A}^\top \frac{dL}{d\boldsymbol{\mu}_i^{out}} = \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1} \frac{dL}{d\boldsymbol{\mu}_i^{out}}.\end{aligned} \tag{C.8}$$

If node j is in an odd column, then $\boldsymbol{\mu}_j^{in}$ does not play a role in the forward step and hence $dL/d\boldsymbol{\mu}_j^{in} = 0$. If node j is in an even column, then

$$\begin{aligned}\frac{dL}{d\boldsymbol{\mu}_j^{in}} &= \frac{dL}{d\boldsymbol{\mu}_j^{out}} + \sum_i \sum_p \frac{dL}{d\boldsymbol{\mu}_i^{out}(p)} \frac{d\boldsymbol{\mu}_i^{out}(p)}{d\boldsymbol{\mu}_j^{in}} \\ &= \frac{dL}{d\boldsymbol{\mu}_j^{out}} + \sum_i \sum_p \frac{dL}{d\boldsymbol{\mu}_i^{out}(p)} \frac{d\left(\left(\mathbf{A}\mathbf{W}_{ij}\right)(p,:)\boldsymbol{\mu}_j^{in}\right)}{d\boldsymbol{\mu}_j^{in}} \\ &= \frac{dL}{d\boldsymbol{\mu}_j^{out}} + \sum_i \sum_p \frac{dL}{d\boldsymbol{\mu}_i^{out}(p)} \left(\mathbf{A}\mathbf{W}_{ij}\right)^\top(:,p) \\ &= \frac{dL}{d\boldsymbol{\mu}_j^{out}} + \sum_i \mathbf{W}_{ij}\mathbf{A} \frac{dL}{d\boldsymbol{\mu}_i^{out}} \\ &= \frac{dL}{d\boldsymbol{\mu}_j^{out}} + \sum_i \mathbf{W}_{ij} \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1} \frac{dL}{d\boldsymbol{\mu}_i^{out}}.\end{aligned} \tag{C.9}$$

Let $\mathbb{I}_{pq}$ be a matrix with (p, q) element as one and all other elements as zero. For i in odd columns,

$$\begin{aligned}
\frac{dL}{d\mathbf{W}_{ij}(p,q)} &= \left(\frac{dL}{d\boldsymbol{\mu}_i^{out}}\right)^{\top} \frac{d\boldsymbol{\mu}_i^{out}}{d\mathbf{W}_{ij}(p,q)} \\
&= \left(\frac{dL}{d\boldsymbol{\mu}_i^{out}}\right)^{\top} \frac{d\left(\mathbf{A}\left(\mathbf{r}_i + \sum_j \mathbf{W}_{ij}\boldsymbol{\mu}_j^{in}\right)\right)}{d\mathbf{W}_{ij}(p,q)} \\
&= \left(\frac{dL}{d\boldsymbol{\mu}_i^{out}}\right)^{\top} \left(-\mathbf{A}\frac{d\mathbf{A}^{-1}}{d\mathbf{W}_{ij}(p,q)}\mathbf{A}\left(\mathbf{r}_i + \sum_j \mathbf{W}_{ij}\boldsymbol{\mu}_j^{in}\right) + \mathbf{A}\frac{d\mathbf{W}_{ij}}{d\mathbf{W}_{ij}(p,q)}\boldsymbol{\mu}_j^{in}\right) \\
&= \left(\frac{dL}{d\boldsymbol{\mu}_i^{out}}\right)^{\top} \left(-\mathbf{A}\mathbb{I}_{pq}\boldsymbol{\mu}_i^{out} + \mathbf{A}\mathbb{I}_{pq}\boldsymbol{\mu}_j^{in}\right) \\
&= \text{trace}\left(\frac{dL}{d\boldsymbol{\mu}_i^{out}}\left(\mathbf{A}\mathbb{I}_{pq}\left(\boldsymbol{\mu}_j^{in} - \boldsymbol{\mu}_i^{out}\right)\right)^{\top}\right) \\
&= \text{trace}\left(\frac{dL}{d\boldsymbol{\mu}_i^{out}}\left(\boldsymbol{\mu}_j^{in} - \boldsymbol{\mu}_i^{out}\right)^{\top}\mathbb{I}_{qp}\mathbf{A}^{\top}\right) \\
&= \text{trace}\left(\mathbf{A}^{\top}\frac{dL}{d\boldsymbol{\mu}_i^{out}}\left(\boldsymbol{\mu}_j^{in} - \boldsymbol{\mu}_i^{out}\right)^{\top}\mathbb{I}_{qp}\right) \\
&= \left[\mathbf{A}^{\top}\frac{dL}{d\boldsymbol{\mu}_i^{out}}\left(\boldsymbol{\mu}_j^{in} - \boldsymbol{\mu}_i^{out}\right)^{\top}\right](p,q) \\
\implies \frac{dL}{d\mathbf{W}_{ij}} &= \mathbf{A}^{\top}\frac{dL}{d\boldsymbol{\mu}_i^{out}}\left(\boldsymbol{\mu}_j^{in} - \boldsymbol{\mu}_i^{out}\right)^{\top} \\
&= \left(\mathbf{I}_K + \sum_k \mathbf{W}_{ik}\right)^{-1}\frac{dL}{d\boldsymbol{\mu}_i^{out}}\left(\boldsymbol{\mu}_j^{in} - \boldsymbol{\mu}_i^{out}\right)^{\top}.
\end{aligned} \tag{C.10}$$

Bibliography

[1] Stefan Roth and Michael J. Black. Fields of experts. *International Journal of Computer Vision*, 82(2):205–229, 2009.

[2] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *International Conference on Learning Representations*, 2014.

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

[4] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[7] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[8] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques.* MIT Press, 2009.

[9] Havard Rue and Leonhard Held. *Gaussian Markov Random Fields: Theory and Applications.* Chapman & Hall/CRC, 2005.

[10] Mohammad Abdelaziz Gowayyed, Marwan Torki, Mohammed Elsayed Hussein, and Motaz El-Saban. Histogram of oriented displacements (HOD): Describing trajectories of human joints for action recognition. In *International Joint Conference on Artificial Intelligence*, 2013.

[11] Mohamed E. Hussein, Marwan Torki, Mohammad Abdelaziz Gowayyed, and Motaz El-Saban. Human action recognition using a temporal hierarchy of covariance descriptors on 3D joint locations. In *International Joint Conference on Artificial Intelligence*, 2013.

[12] Jiang Wang and Ying Wu. Learning maximum margin temporal warping for action recognition. In *IEEE International Conference on Computer Vision*, 2013.

[13] Jiang Wang, Zicheng Liu, Ying Wu, and Junsong Yuan. Mining actionlet ensemble for action recognition with depth cameras. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[14] Ferda Ofli, Rizwan Chaudhry, Gregorij Kurillo, René Vidal, and Ruzena Bajcsy. Sequence of the most informative joints (SMIJ): A new representation for human skeletal action recognition. *Journal of Visual Communication and Image Representation*, 25(1):24–38, 2014.

[15] Eshed Ohn-Bar and Mohan M. Trivedi. Joint angles similarities and HOG2 for action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2013.

[16] Pavan K. Turaga, Ashok Veeraraghavan, Anuj Srivastava, and Rama Chellappa. Statistical computations on Grassmann and Stiefel manifolds for image and video-based recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2273–2286, 2011.

[17] Oncel Tuzel, Fatih Porikli, and Peter Meer. Pedestrian detection via classification on Riemannian manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(10):1713–1727, 2008.

[18] Ruiping Wang, Huimin Guo, Larry S. Davis, and Qionghai Dai. Covariance discriminative learning: A natural and efficient approach to image set classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

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

[20] Jihun Ham and Daniel D. Lee. Grassmann discriminant analysis: A unifying view on subspace-based learning. In *International Conference on Machine Learning*, 2008.

[21] Tae-Kyun Kim, Ognjen Arandjelovic, and Roberto Cipolla. Boosted manifold principal angles for image set-based recognition. *Pattern Recognition*, 40(9):2475–2484, 2007.

[22] Mehrtash Tafazzoli Harandi, Conrad Sanderson, Sareh Abolahrari Shirazi, and Brian C. Lovell. Graph embedding discriminant analysis on Grassmannian manifolds for improved image set matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[23] Masoud Faraki, Mehrtash Tafazzoli Harandi, and Fatih Porikli. Image set classification by symmetric positive semi-definite matrices. In *IEEE Winter Conference on Applications of Computer Vision*, 2016.

[24] Sadeep Jayasumana, Richard I. Hartley, Mathieu Salzmann, Hongdong Li, and Mehrtash Tafazzoli Harandi. Kernel methods on the Riemannian manifold of symmetric positive definite matrices. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[25] Masoud Faraki, Mehrtash Tafazzoli Harandi, and Fatih Murat Porikli. Material classification on symmetric positive definite manifolds. In *IEEE Winter Conference on Applications of Computer Vision*, 2015.

[26] Oncel Tuzel, Fatih Porikli, and Peter Meer. Region covariance: A fast descriptor for detection and classification. In *European Conference on Computer Vision*, 2006.

[27] Forest Agostinelli, Michael R Anderson, and Honglak Lee. Adaptive multi-column deep neural networks with application to robust image denoising. In *Advances in Neural Information Processing Systems*, 2013.

[28] Harold Christopher Burger, Christian J. Schuler, and Stefan Harmeling. Image denoising: Can plain neural networks compete with BM3D? In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[29] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[30] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, 2013.

[31] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[32] Ashok Veeraraghavan, Anuj Srivastava, Amit K. Roy-Chowdhury, and Rama Chellappa. Rate-invariant recognition of humans and their activities. *IEEE Transactions on Image Processing*, 18(6):1326–1339, 2009.

[33] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[34] Brian Hall. *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Springer, 2003.

[35] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC press, 1994.

[36] J. M. McCarthy. *Introduction to Theoretical Kinematics*. MIT Press, 1990.

[37] Ben Kenwright. A beginners guide to dual-quaternions: What they are, how they work, and how to use them for 3D character hierarchies. In *International Conference on Computer Graphics, Visualization and Computer Vision*, 2012.

[38] Daniel Bump. *Lie Groups, Graduate Texts in Mathematics*, volume 225. Springer, New York, 2004.

[39] Frank C. Park and Bahram Ravani. Smooth invariant interpolation of rotations. *ACM Transactions on Graphics*, 16(3):277–295, 1997.

[40] Calin Belta and Vijay R. Kumar. An SVD-based projection method for interpolation on SE(3). *IEEE Transactions on Robotics and Automation*, 18(3):334–345, 2002.

[41] Milos Zefran and Vijay Kumar. Two methods for interpolating rigid body motions. In *IEEE International Conference on Robotics and Automation*, 1998.

[42] Milos Zefran, Vijay Kumar, and Christopher Croke. Choice of Riemannian metrics for rigid body kinematics. In *ASME Design Engineering Technical Conference and Computers in Engineering Conference*, 1996.

[43] Ted J. Broida, S. Chandrashekhar, and Rama Chellappa. Recursive 3-D motion estimation from a monocular image sequence. *IEEE Transactions on Aerospace and Electronic Systems*, 26(4):639–656, 1990.

[44] Ted J. Broida and Rama Chellappa. Estimating the kinematics and structure of a rigid object from a sequence of monocular images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):497–513, 1991.

[45] Gem-Sun J. Young and Rama Chellappa. 3-D motion estimation using a sequence of noisy stereo images: Models, motion estimation and uniqueness results. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):735–759, 1990.

[46] Pavan K. Turaga, Rama Chellappa, V. S. Subrahmanian, and Octavian Udrea. Machine recognition of human activities: A survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(11):1473–1488, 2008.

[47] J. K. Aggarwal and M. S. Ryoo. Human activity analysis: A review. *ACM Computing Surveys*, 43(3):16:1–16:43, 2011.

[48] Vladimir M. Zatsiorsky. *Kinematics of Human Motion*. Human Kinetics Publishers, 1997.

[49] Aravind Sundaresan and Rama Chellappa. Model driven segmentation of articulating humans in Laplacian Eigenspace. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(10):1771–1785, 2008.

[50] Aravind Sundaresan and Rama Chellappa. Multi-camera tracking of articulated human motion using shape and motion cues. *IEEE Transactions on Image Processing*, 18(9):2114–2126, 2009.

[51] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3):90–126, 2006.

[52] Jamie Shotton, Andrew W. Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[53] Gunnar Johansson. Visual perception of biological motion and a model for its analysis. *Perception & Psychophysics*, 14(2):201–211, 1973.

[54] Yaser Sheikh, Mumtaz Sheikh, and Mubarak Shah. Exploring the space of a human action. In *IEEE International Conference on Computer Vision*, 2005.

[55] Fengjun Lv and Ramakant Nevatia. Recognition and segmentation of 3D human action using HMM and multi-class adaboost. In *European Conference on Computer Vision*, 2006.

[56] Miguel Reyes, Gabriel Domínguez, and Sergio Escalera. Feature weighting in dynamic time warping for gesture recognition in depth data. In *IEEE International Conference on Computer Vision Workshops*, 2011.

[57] Lu Xia, Chia-Chih Chen, and J. K. Aggarwal. View invariant human action recognition using histograms of 3D joints. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2012.

[58] Zhanpeng Shao and Youfu Li. A new descriptor for multiple 3D motion trajectories recognition. In *IEEE International Conference on Robotics and Automation*, 2013.

[59] Ping Wei, Nanning Zheng, Yibiao Zhao, and Song-Chun Zhu. Concurrent action detection with structural prediction. In *IEEE International Conference on Computer Vision*, 2013.

[60] Xiaodong Yang and Yingli Tian. Effective 3D action recognition using Eigenjoints. *Journal of Visual Communication and Image Representation*, 25(1):2–11, 2014.

[61] David C. Marr and H. Keith Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London, Series B, Biological Sciences*, 200(1140):269 – 294, 1978.

[62] Yaser Yacoob and Michael J. Black. Parameterized modeling and recognition of activities. *Computer Vision and Image Understanding*, 73(2):232–247, 1999.

[63] Dariu M. Gavrila and Larry S. Davis. Towards 3-D model-based tracking and recognition of human movement: A multi-view approach. In *International Workshop on Automatic Face and Gesture Recognition*, 1995.

[64] Lulu Chen, Hong Wei, and James M. Ferryman. A survey of human motion analysis using depth imagery. *Pattern Recognition Letters*, 34(15):1995–2006, 2013.

[65] Mao Ye, Qing Zhang, Liang Wang, Jiejie Zhu, Ruigang Yang, and Juergen Gall. A survey on human motion analysis from depth data. In *Dagstuhl Seminar on Time-of-Flight Imaging and GCPR Workshop on Imaging New Modalities*, 2013.

[66] Junxia Gu, Xiaoqing Ding, Shengjin Wang, and Youshou Wu. Action and gait recognition from recovered 3-D human joints. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 40(4):1021–1033, 2010.

[67] Alexandros Andre Chaaraoui, José Ramón Padilla-Løpez, and Francisco Flórez-Revuelta. Fusion of skeletal and silhouette-based features for human action recognition with RGB-D devices. In *IEEE International Conference on Computer Vision Workshops*, 2013.

[68] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[69] Mihai Zanfir, Marius Leordeanu, and Cristian Sminchisescu. The moving pose: An efficient 3D kinematics descriptor for low-latency action recognition and detection. In *IEEE International Conference on Computer Vision*, 2013.

[70] Christopher Ellis, Syed Zain Masood, Marshall F. Tappen, Joseph J. LaViola Jr., and Rahul Sukthankar. Exploring the trade-off between accuracy and observational latency in action recognition. *International Journal of Computer Vision*, 101(3):420–436, 2013.

[71] Yu Zhu, Wenbin Chen, and Guodong Guo. Fusing spatiotemporal features and joints for 3D action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2013.

[72] Georgios Evangelidis, Gurkirt Singh, and Radu Horaud. Skeletal quads: Human action recognition using joint quadruples. In *International Conference on Pattern Recognition*, 2014.

[73] Tommi S. Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems*, 1998.

[74] Chunyu Wang, Yizhou Wang, and Alan L. Yuille. An approach to pose-based action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[75] Meinard Müller, Tido Röder, and Michael Clausen. Efficient content-based retrieval of motion capture data. *ACM Transactions on Graphics*, 24(3):677–685, 2005.

[76] Angela Yao, Juergen Gall, and Luc J. Van Gool. Coupled action recognition and pose estimation from multiple views. *International Journal of Computer Vision*, 100(1):16–37, 2012.

[77] Kiwon Yun, Jean Honorio, Debaleena Chattopadhyay, Tamara L. Berg, and Dimitris Samaras. Two-person interaction detection using body-pose features and multiple instance learning. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2012.

[78] Angela Yao, Juergen Gall, Gabriele Fanelli, and Luc J. Van Gool. Does human action recognition benefit from pose estimation? In *British Machine Vision Conference*, 2011.

[79] Rizwan Chaudhry, Ferda Ofli, Gregorij Kurillo, Ruzena Bajcsy, and René Vidal. Bio-inspired dynamic 3D discriminative skeletal features for human action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2013.

[80] Lee W. Campbell and Aaron F. Bobick. Recognition of human body motion using phase space constraints. In *IEEE International Conference on Computer Vision*, 1995.

[81] Jaeyong Sung, Colin Ponce, Bart Selman, and Ashutosh Saxena. Unstructured human activity detection from RGBD images. In *IEEE International Conference on Robotics and Automation*, 2012.

[82] Kaustubh Kulkarni, Georgios Evangelidis, Jan Cech, and Radu Horaud. Continuous action recognition based on sequence alignment. *International Journal of Computer Vision*, 112(1):90–114, 2015.

[83] Wanqing Li, Zhengyou Zhang, and Zicheng Liu. Action recognition based on a bag of 3D points. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2010.

[84] Lorenzo Seidenari, Vincenzo Varano, Stefano Berretti, Alberto Del Bimbo, and Pietro Pala. Recognizing actions from depth cameras as weakly aligned multi-part bag-of-poses. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2013.

[85] Omar Oreifej and Zicheng Liu. HON4D: Histogram of oriented 4D normals for activity recognition from depth sequences. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[86] Victoria Bloom, Dimitrios Makris, and Vasileios Argyriou. G3D: A gaming action dataset and real time action recognition evaluation framework. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2012.

[87] Maxime Devanne, Hazem Wannous, Stefano Berretti, Pietro Pala, Mohamed Daoudi, and Alberto Del Bimbo. 3D human action recognition by shape analysis of motion trajectories on Riemannian manifold. *IEEE Transactions on Cybernetics*, 45(7):1340–1352, 2015.

[88] Liliana Lo Presti, Marco La Cascia, Stan Sclaroff, and Octavia I. Camps. Gesture modeling by hanklet-based hidden Markov model. In *Asian Conference on Computer Vision*, 2014.

[89] Rushil Anirudh, Pavan K. Turaga, Jingyong Su, and Anuj Srivastava. Elastic functional coding of human actions: From vector-fields to latent variables. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[90] Siqi Nie and Qiang Ji. Capturing global and local dynamics for human action recognition. In *International Conference on Pattern Recognition*, 2014.

[91] Salem Said, Nicolas Courty, Nicolas Le Bihan, and Stephen J. Sangwine. Exact principal geodesic analysis for data on SO(3). In *European Signal Processing Conference*, 2007.

[92] Maxime Tournier, Xiaomao Wu, Nicolas Courty, Elise Arnaud, and Lionel Revéret. Motion compression using principal geodesics analysis. *Computer Graphics Forum*, 28(2):355–364, 2009.

[93] Knut Hüper, Martin Kleinsteuber, and F. Silva Leite. Rolling Stiefel manifolds. *International Journal of Systems Science*, 39(9):881–887, 2008.

[94] Knut Hüper and F. Silva Leite. On the geometry of rolling and interpolation curves on Sn, SOn, Grassmann manifolds. *Journal of Dynamical and Control Systems*, 13(4):467–502, 2007.

[95] R. W. Sharpe. *Differential Geometry.* Springer-Verlag, NewYork, 1996.

[96] Rui Caseiro, Pedro Martins, João F. Henriques, Fatima Silva Leite, and Jorge Batista. Rolling Riemannian manifolds to solve the multi-class classification problem. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[97] Knut Hüper and F. Silva Leite. Smoothing interpolation curves on manifolds with applications to path planning. In *Mediterranean Conference on Control and Automation*, 2002.

[98] Yueshi Shen, Knut Hüper, and F. Silva Leite. Smooth interpolation of orientation by rolling and wrapping for robot motion planning. In *IEEE International Conference on Robotics and Automation*, 2006.

[99] Rui Caseiro, João F. Henriques, Pedro Martins, and Jorge Batista. Beyond the shortest path: Unsupervised domain adaptation by sampling subspaces along the spline flow. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[100] Du Q. Huynh. Metrics for 3D rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

[101] Jingyong Su, Sebastian Kurtek, Eric Klassen, and Anuj Srivastava. Statistical analysis of trajectories on Riemannian manifolds: Bird migration, hurricane tracking, and video surveillance. *Annals of Applied Statistics*, 8(1):530–552, 2014.

[102] Anuj Srivastava, Eric Klassen, Shantanu H. Joshi, and Ian H. Jermyn. Shape analysis of elastic curves in Euclidean spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(7):1415–1428, 2011.

[103] David G. Kendall. Shape manifolds, Procrustean metrics, and complex projective spaces. *Bulletin of the London Mathematical Society*, 16(2):81–121, 1984.

[104] Ashok Veeraraghavan, Amit K. Roy-Chowdhury, and Rama Chellappa. Matching shape sequences in video with applications in human movement analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1896–1909, 2005.

[105] Rizwan Chaudhry, Avinash Ravichandran, Gregory D. Hager, and René Vidal. Histograms of oriented optical flow and Binet-Cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[106] Gert R. G. Lanckriet, Nello Cristianini, Peter L. Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[107] Alain Rakotomamonjy, Francis R. Bach, Stéphane Canu, and Yves Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521, 2008.

[108] Jos F. Sturm. Using sedumi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999.

[109] Alessandro Bissacco, Alessandro Chiuso, Yi Ma, and Stefano Soatto. Recognition of human gaits. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[110] Osamu Yamaguchi, Kazuhiro Fukui, and Ken-ichi Maeda. Face recognition using temporal image sequence. In *International Conference on Face and Gesture Recognition*, 1998.

[111] Pavan K. Turaga, Ashok Veeraraghavan, and Rama Chellappa. Statistical analysis on Stiefel and Grassmann manifolds with applications in computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[112] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

[113] Vikas Sindhwani, Partha Niyogi, and Mikhail Belkin. Beyond the point cloud: From transductive to semi-supervised learning. In *International Conference on Machine Learning*, 2005.

[114] Pierre-Antoine Absil, Robert Mahony, and Rodolphe Sepulchre. Riemannian geometry of Grassmann manifolds with a view on algorithmic computation. *Acta Applicandae Mathematicae*, 80(2):199–220, 2004.

[115] Alan Edelman, T. A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal of Matrix Analysis Applications*, 20(2):303–353, 1998.

[116] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A Riemannian framework for tensor computing. *International Journal of Computer Vision*, 66(1):41–66, 2006.

[117] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. Geometric means in a novel vector space structure on symmetric positive-definite matrices. *SIAM Journal of Matrix Analysis Applications*, 29(1):328–347, 2006.

[118] Kilian Q. Weinberger and Lawrence K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70(1):77–90, 2006.

[119] Minyoung Kim, Sanjiv Kumar, Vladimir Pavlovic, and Henry A. Rowley. Face tracking and recognition with visual constraints in real-world videos. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[120] Bastian Leibe and Bernt Schiele. Analyzing appearance and contour based methods for object categorization. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[121] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[122] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen O. Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007.

[123] Daniel Zoran and Yair Weiss. From learning models of natural image patches to whole image restoration. In *IEEE International Conference on Computer Vision*, 2011.

[124] Marshall F. Tappen, Ce Liu, Edward H. Adelson, and William T. Freeman. Learning Gaussian conditional random fields for low-level vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[125] Donald Geman and Chengda Yang. Nonlinear image recovery with half-quadratic regularization. *IEEE Transactions on Image Processing*, 5(7):932–946, 1995.

[126] Dilip Krishnan and Rob Fergus. Fast image deconvolution using hyper-Laplacian priors. In *Advances in Neural Information Processing Systems*, 2009.

[127] Yilun Wang, Junfeng Yang, Wotao Yin, and Yin Zhang. A new alternating minimization algorithm for total variation image reconstruction. *SIAM Journal on Imaging Sciences*, 1(3):248–272, 2008.

[128] Jeremy Jancsary, Sebastian Nowozin, and Carsten Rother. Loss-specific training of non-parametric image restoration models: A new state of the art. In *European Conference on Computer Vision*, 2012.

[129] Uwe Schmidt, Jeremy Jancsary, Sebastian Nowozin, Stefan Roth, and Carsten Rother. Cascades of regression tree fields for image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(4):677–689, 2016.

[130] Uwe Schmidt and Stefan Roth. Shrinkage fields for effective image restoration. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[131] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. 3-D depth reconstruction from a single still image. *International Journal of Computer Vision*, 76(1):53–69, 2008.

[132] Tadas Baltrusaitis, Peter Robinson, and Louis-Philippe Morency. Continuous conditional neural fields for structured regression. In *European Conference on Computer Vision*, 2014.

[133] Tao Qin, Tie yan Liu, Xu dong Zhang, De sheng Wang, and Hang Li. Global ranking using continuous conditional random fields. In *Advances in Neural Information Processing Systems*, 2008.

[134] Eero P. Simoncelli and Edward H. Adelson. Noise removal via Bayesian wavelet coring. In *International Conference on Image Processing*, 1996.

[135] Javier Portilla, Vasily Strela, Martin J. Wainwright, and Eero P. Simoncelli. Image denoising using scale mixtures of Gaussians in the wavelet domain. *IEEE Transactions on Image Processing*, 12(11):1338–1351, 2003.

[136] Yunjin Chen, Wei Yu, and Thomas Pock. On learning optimized reaction diffusion processes for effective image restoration. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[137] Mohammad Reza Hajiaboli. An anisotropic fourth-order diffusion filter for image noise removal. *International Journal of Computer Vision*, 92(2):177–191, 2011.

[138] Gerlind Plonka and Jianwei Ma. Nonlinear regularized reaction-diffusion filters for denoising of images with textures. *IEEE Transactions on Image Processing*, 17(8):1283–1294, 2008.

[139] Weisheng Dong, Xin Li, Lei Zhang, and Guangming Shi. Sparsity-based image denoising via dictionary learning and structural clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[140] Weisheng Dong, Lei Zhang, Guangming Shi, and Xin Li. Nonlocally centralized sparse representation for image restoration. *IEEE Transactions on Image Processing*, 22(4):1620–1630, 2013.

[141] Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, 2006.

[142] Julien Mairal, Francis R. Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. Non-local sparse models for image restoration. In *IEEE International Conference on Computer Vision*, 2009.

[143] Shuhang Gu, Lei Zhang, Wangmeng Zuo, and Xiangchu Feng. Weighted nuclear norm minimization with application to image denoising. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[144] Marc Lebrun, Antoni Buades, and Jean-Michel Morel. A nonlocal Bayesian image denoising algorithm. *SIAM Journal on Imaging Sciences*, 6(3):1665–1688, 2013.

[145] Viren Jain and H. Sebastian Seung. Natural image denoising with convolutional networks. In *Advances in Neural Information Processing Systems*, 2008.

[146] Shuangteng Zhang and Ezzatollah Salari. Image denoising using a neural network based non-linear filter in wavelet domain. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2005.

[147] Yi-Tong Zhou, Rama Chellappa, Aseem Vaid, and B. Keith Jenkins. Image restoration using a neural network. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1141–1151, 1988.

[148] Adrian Barbu. Training an active random field for real-time image denoising. *IEEE Transactions on Image Processing*, 18(11):2451–2462, 2009.

[149] Stéphane Ross, Daniel Munoz, Martial Hebert, and J. Andrew Bagnell. Learning message-passing inference machines for structured prediction. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[150] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*, 2010.

[151] Alexander G. Schwing and Raquel Urtasun. Fully connected deep structured networks. *CoRR*, abs/1503.02351, 2015.

[152] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip H. S. Torr. Conditional random fields as recurrent neural networks. In *IEEE International Conference on Computer Vision*, 2015.

[153] John R. Hershey, Jonathan Le Roux, and Felix Weninger. Deep unfolding: Model-based inspiration of novel deep architectures. *CoRR*, abs/1409.2574, 2014.

[154] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[155] David R. Martin, Charless C. Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *IEEE International Conference on Computer Vision*, 2001.

[156] Mark Everingham, S. M. Ali Eslami, Luc J. Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015.

[157] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.

[158] Yunjin Chen, Thomas Pock, René Ranftl, and Horst Bischof. Revisiting loss-specific training of filter-based MRFs for image restoration. In *German Conference on Pattern Recognition*, 2013.

[159] Pedro H. O. Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *International Conference on Machine Learning*, 2014.

[160] Bharath Hariharan, Pablo Andrés Arbeláez, Ross B. Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[161] Mohammadreza Mostajabi, Payman Yadollahpour, and Gregory Shakhnarovich. Feedforward semantic segmentation with zoom-out features. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[162] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, 2001.

[163] Sean Bell, Paul Upchurch, Noah Snavely, and Kavita Bala. Material recognition in the wild with the materials in context database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[164] George Papandreou, Liang-Chieh Chen, Kevin Murphy, and Alan L. Yuille. Weakly- and semi-supervised learning of a DCNN for semantic image segmentation. In *IEEE International Conference on Computer Vision*, 2015.

[165] Xiaojuan Qi, Jianping Shi, Shu Liu, Renjie Liao, and Jiaya Jia. Semantic segmentation with object clique potentials. In *IEEE International Conference on Computer Vision*, 2015.

[166] Fayao Liu, Chunhua Shen, and Guosheng Lin. Deep convolutional neural fields for depth estimation from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[167] Josiane Zerubia and Rama Chellappa. Mean field annealing using compound Gauss-Markov random fields for edge detection and image estimation. *IEEE Transactions on Neural Networks*, 4(4):703–709, 1993.

[168] Rama Chellappa and Shankar Chatterjee. Classification of textures using Gaussian Markov random fields. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 33(4):959–963, 1985.

[169] B. S. Manjunath and Rama Chellappa. Unsupervised texture segmentation using Markov random field models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):478–482, 1991.

[170] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Advances in Neural Information Processing Systems*, 2011.

[171] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[172] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[173] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision*, 2014.

[174] Jifeng Dai, Kaiming He, and Jian Sun. Convolutional feature masking for joint object and stuff segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[175] Abhishek Sharma, Oncel Tuzel, and David W. Jacobs. Deep hierarchical parsing for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[176] Abhishek Sharma, Oncel Tuzel, and Ming-Yu Liu. Recursive context propagation network for semantic scene labeling. In *Advances in Neural Information Processing Systems*, 2014.

[177] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *IEEE International Conference on Computer Vision*, 2015.

[178] Guosheng Lin, Chunhua Shen, Ian D. Reid, and Anton van den Hengel. Efficient piecewise training of deep structured models for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[179] Guosheng Lin, Chunhua Shen, Ian D. Reid, and Anton van den Hengel. Deeply learning the messages in message passing inference. In *Advances in Neural Information Processing Systems*, 2015.

[180] Trinh Minh Tri Do and Thierry Artières. Neural conditional random fields. In *International Conference on Artificial Intelligence and Statistics*, 2010.

[181] Jian Peng, Liefeng Bo, and Jinbo Xu. Conditional neural fields. In *Advances in Neural Information Processing Systems*, 2009.

[182] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep structured output learning for unconstrained text recognition. In *International Conference on Learning Representations*, 2014.

[183] Jonathan J. Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in Neural Information Processing Systems*, 2014.

[184] Liang-Chieh Chen, Alexander G. Schwing, Alan L. Yuille, and Raquel Urtasun. Learning deep structured models. In *International Conference on Machine Learning*, 2015.

[185] Yoshua Bengio, Yann LeCun, and Donnie Henderson. Globally trained handwritten word recognizer using spatial representation, convolutional neural networks, and hidden Markov models. In *Advances in Neural Information Processing Systems*, 1993.

[186] Jifeng Dai, Kaiming He, and Jian Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *IEEE International Conference on Computer Vision*, 2015.

[187] Seunghoon Hong, Hyeonwoo Noh, and Bohyung Han. Decoupled deep neural network for semi-supervised semantic segmentation. In *Advances in Neural Information Processing Systems*, 2015.

[188] Pedro H. O. Pinheiro and Ronan Collobert. From image-level to pixel-level labeling with convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[189] Justin Domke. Learning graphical model parameters with approximate marginal inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(10):2454–2467, 2013.

[190] Philipp Krähenbühl and Vladlen Koltun. Parameter learning and convergent inference for dense random fields. In *International Conference on Machine Learning*, 2013.

[191] Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[192] Marshall F. Tappen, Kegan G. G. Samuel, Craig V. Dean, and David M. Lyle. The logistic random field - A convenient graphical model for learning parameters for MRF-based labeling. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[193] Bharath Hariharan, Pablo Arbelaez, Lubomir D. Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *IEEE International Conference on Computer Vision*, 2011.

[194] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

[195] Jieping Ye, Shuiwang Ji, and Jianhui Chen. Multi-class discriminant kernel learning via convex programming. *Journal of Machine Learning Research*, 9:719–758, 2008.

[196] Raviteja Vemulapalli, Vinay Praneeth Boda, and Rama Chellappa. MKL-RT: Multiple kernel learning for ratio-trace problems via convex optimization. *CoRR*, abs/1410.4470, 2014.