Pose based human activity recognition

Wenbo Li

University at Albany, State University of New York

The University at Albany community has made this article openly available. Please share how this access benefits you.

Follow this and additional works at: https://scholarsarchive.library.albany.edu/legacy-etd

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation

https://scholarsarchive.library.albany.edu/legacy-etd/2324

This Dissertation is brought to you for free and open access by the The Graduate School at Scholars Archive. It has been accepted for inclusion in Legacy Theses & Dissertations (2009 - 2024) by an authorized administrator of Scholars Archive.
Please see Terms of Use. For more information, please contact scholarsarchive@albany.edu.
POSE BASED HUMAN ACTIVITY RECOGNITION

by

Wenbo Li

A Dissertation
Submitted to the University at Albany, State University of New York
in Partial Fulfillment of
the Requirements for the Degree of
Doctor of Philosophy

College of Engineering and Applied Sciences
Department of Computer Science
Summer 2019
ABSTRACT

Pose based human activity recognition is an important step towards video understanding. The last decade has witnessed the great progress in this field which is driven by multiple technical innovations, *i.e.*, kinect, pose estimation techniques, deep learning, *etc*.

In this dissertation, we revisit the area of human activity recognition by clarifying the definition of human activities, data modalities for activity recognition, the generation problem formulation of human activity recognition, and the general solution based on sequence recognition.

The introduction of the human activity recognition technologies follow the chronological order: first the conventional handcrafted activity feature designs, then activity temporal modeling with recurrent neural networks or convolutional neural networks, and finally the activity spatial modeling with graph convolutional neural networks. Before the introduction of the deep neural network based methods, we first review the literature of the deep neural networks as the fundamentals for understanding the related methods.

Following the literature review of others’ methods within each period, we present our own methods. Note that within each period, we did not just follow the trend but proposed some novel ideas that are orthogonal to the trend. Such ideas are aiming to push the practical applicability of the human activity recognition technologies.

Finally, we indicate the potential future directions of this field which hopefully can inspire researchers to land the technologies of human activity recognition into our daily life.
ACKNOWLEDGMENT

I owe my deepest gratitude to my adviser, Prof. Siwei Lyu, for his continuous support, guidance, and encouragement. Prof. Lyu guided me to the math thinking, and taught me how to formulate an elegant math model. Prof. Lyu gave me the right amount of freedom to explore, which really helped me find my interests and trained my ability to do independent research. He also provided insightful advice and enormous help when I got stuck or lost.

I want to thank my co-advisor Prof. Ming-Ching Chang. I worked with Prof. Chang on several projects, got inspired by his research, and got his instructions on research. I would like to thank the rest of thesis committee Prof. James Moulic and Dr. Xiao Bian for taking time to review my thesis. I also want to thank Prof. Mooi Choo Chuah at Lehigh. She drove me to the area of human activity recognition, and trained my various basic researcher skills.

I want to thank Dr. Pengchuan Zhang from MSR AI, who worked side by side with me for the past year, trained me and reshaped my deep learning knowledge and skills fundamentally with his solid math and optimization knowledge. I considered Dr. Pengchuan Zhang as a role model of young researcher in my career because he is a nice and modest person, qualified husband, trustful friend, responsible mentor, professional and devoted researcher. I also want to thank Dr. Lei Zhang from MS for his support and visionary guidance for my research projects in the past year. I regarded Dr. Lei Zhang as a role model of senior industrial researcher in my career because he is nice, disciplined, professional and visionary.

I want to thank my other collaborators who contribute to my thesis projects, including Prof. Yi Zhang, Linghan Xu, Xingyu Huang from TJU, Shuchen Weng from PKU, Dr. Yanbo Fan from Tencent AI, Dianqi Li from UW. I really learned a lot from the collaboration with them!

I would like to thank Dr. Junjie Yan for his supports when I was at SenseTime. I would also like to thank Dr. Sernam Lim, Dr. Xiao Bian and Dr. Longyin Wen for
their supports when I was at GE.

I thank my friends at Lehigh and UAlbany Dr. Dawei Li, Dr. Qinghan Xue, Shen Liu, Dr. Tinzhe Zhou, Dr. Han Zhang, Dr. Tao Xu, Dr. Dawei Du, Yi Wei, Yuezun Li, Lipeng Ke, Huidong Peng, and more - for the help, the fun, and the good time we had together.

Most importantly, I would like to thank my beloved parents, Mr. Jianping Li and Mrs. Ailan Zhang, for their endless love and support. No words can express how grateful I am for you and how much I love you. I would also like to thank my beloved parents in law and spirit, Prof. Wenhua Hou and Prof. Xuemei Li, for their love from God. You always guided us and never blamed us. Thank Lord for giving me you as my parents in law and spirit.

Finally, my thanks to my beloved wife, Mrs. Ruolin Hou. To accompany me, you have sacrificed your professional career in China, and taken the risk switching major from finance to computer science. Your love and devotion always encourage me and warm my heart. You are the most important person in my life, and also the bravest and smartest woman I have ever seen. Thank you for supporting and helping me all the way along. Thank Lord for having mercy on me!
CONTENTS

ABSTRACT ................................................................. ii

ACKNOWLEDGMENT ...................................................... iii

LIST OF FIGURES ....................................................... ix

1. Introduction .......................................................... 1

1.1 Structure of the Thesis ............................................ 3

2. Human Activity Recognition ....................................... 4

2.1 Human Activities .................................................. 4

2.1.1 Definition ....................................................... 4

2.1.2 Data Modalities ................................................ 4

2.1.3 Problem Formulation ......................................... 5

2.2 Sequence Recognition ............................................ 7

2.2.1 Training ......................................................... 9

2.2.2 Chronological Summary ...................................... 10

3. Handcrafted Features .............................................. 11

3.1 Background ........................................................ 11

3.2 Method ............................................................. 13

3.2.1 Modeling a Potential Activity Instance using the Category-
blind Visual Representation ........................................ 15

3.2.1.1 MU Model .................................................. 15

3.2.1.2 MU Combination Model .................................. 17

3.2.1.3 Construction of Reference Repositories ............... 21

3.2.2 Activity Classification using MCP .......................... 22

3.3 Experiments ........................................................ 24
# LIST OF FIGURES

2.1 Examples of different human activity categories. The image source for (a) is [50], for (b) and (c) is [73], and for (d) is [74]. ........................................ 5

2.2 Examples of different modalities: (a) samples frames of *RGB videos* with the *human poses* drawn; (b) *depth maps*. ........................................ 6

2.3 Examples of different formulations. A colored interval represents an activity instance. ........................................ 7

2.4 Unfolded RNN. ........................................ 8

2.5 Temporal CNN. ........................................ 9

3.1 Examples of human activities depicting the complexities in activity recognition: (a) a multi-person activity with each individual performing a single-person activity; (b) a multi-person activity involving a single-person activity and multi-person interaction activity; (c) a multi-person activity that cannot be reduced to the combination of two single-person activities. See texts for more details. ........................................ 11

3.2 Main steps of CHARM. This example involves 3 persons (*i.e.*, Person *A, B, and C*) and 6 potential activities. The recognition result indicates that *A* is performing a single-person activity of Class 1, and *B& C* are performing a two-person interaction of Class 10. ..................... 14

3.3 The representation of body part configurations. ........................................ 16

3.4 The upper row shows how an MP collection can be formed based on the extracted body motion matrix using the motion template codebook. The lower half is about the extraction of the MP co-occurrence patterns. Red arrows indicate the quantization procedure. The light grey blocks are reference repositories. In the block that shows the MP co-occurrence pattern, each activity class should have 22 disjoint groups (each group is illustrated as an ellipse) with each corresponding to a dimension. However, we merely show five disjoint groups to make the diagram clearer. The notation used herein and the rest of the paper are defined in table 3.1 18
3.5 Sample frames from two test sequences of the Hybrid Actions3D dataset:
The first row presents a hybrid-category test sequence involving two activity instances, \textit{i.e.}, Person A performing the single-person activity "throw", while Person B and C are performing the two-person "kick" interaction. The second row presents a mono-category test sequence consisting of three single-person activity instances, \textit{i.e.}, Person A performing the "throw", B performing "goggles", and C performing "duck" activity.

4.1 A three-layer fully-connected NN. ............................................. 32
4.2 A two-layer CNN. ................................................................. 32
4.3 Comparison of different normalization layers. ............................ 35

5.1 \textbf{Method overview}. (a) Visualization of action instances from three action classes. (b) A three-level RNN Tree (RNN-T) associated with learned Action Category Hierarchy (ACH) in (c). Each circle with a number represents an action class with action label. Grey circle represents ambiguous class (not yet confidently recognized by the current RNN and need further inference in the hierarchy). Black circle represents recognized action class. Blue rectangles depict action categories \(\{C_i\}\). Black arrows indicate the division of action categories and RNN nodes. Blue dotted arrows indicate the feeding of input skeletal data. ................................................................. 45

5.2 (a) A RNN architecture for activity recognition with 400 LSTM units, whose structure is shown in (b). 8 learnable parameters of a LSTM block is shown in (b), \textit{i.e.}, \(W_{xc}, W_{hc}, W_{xi}, W_{hi}, W_{xf}, W_{hf}, W_{xo}, \text{and } W_{ho}\) . . . 47

5.3 Incremental learning procedures: (a) inserting new classes to activity categories with similar classes as the new ones, and (b) updating ACH and RNN-T. Red circles represent the new classes. Better viewed in color and by zooming in. ................................................................. 54

5.4 Classification accuracy against \(\theta_o\) and \(\theta_l\) ...................................................... 57

5.5 Recognition results on standalone datasets. The number of activity classes of each dataset is shown behind its name. ................................................................. 59

5.6 Incremental learning comparison results. “I-” stands for incremental. . 60
6.1 (a) The activity representation is a tensor with three dimensions, which is formed in two steps. We represent each dimension at each time step as a feature vector, and concatenate them according to the dimension and time order.

(b) Module diagram. The red and green blocks represent the temporal and structural stream, respectively. L/M/HFE mean the low/medium/high-level feature extractor, and the prefixes of MFE, S/M/L, mean the short/medium/long range. ZI means the zoom-in module, and SE means the encoder shared by M-MFE and L-MFE. GT and CLS represent the gating and classification module, respectively. See texts for details.

6.2 The network structure of a single stream of our model.

7.1 An example of human skeleton input for individual activity recognition. Body joints are indexed in odd numbers, and bones are in even numbers. The body parts are categorized into five groups (highlighted in different colors) according to the human body structure.

7.2 (a) Independence assumption of heterogeneous nodes. (b) Interdependence modeling (indicated by the red edges) for heterogeneous nodes. The green and blue branches indicate the joint branch and bone branch, respectively.

7.3 (a) Exclusive weight function. (b) Intra weight function. (c) Inclusive weight function.

7.4 (a) Comparison of three weight functions for the joint branch. (b) Comparison of three weight functions for the bone branch. The white areas indicate the frozen zero entries, and the colored areas indicate the learnable entries.
CHAPTER 1

Introduction

In computer vision, the term “human activity recognition from video sequences” encompasses all tasks in which human motions with patterns are transcribed with discrete labels. Well-known examples include individual activity recognition for motion sensing games or smart home, group activity recognition for video analysis or surveillance. With the advent of mature pose estimation and depth estimation approaches, there are more choices regarding data modalities including videos, poses, depth sequences, and symbolic context annotations. Among these modalities, the human pose sequence is most relevant to human body motions, so it is very popular for body-motion-salient activities, such as individual human activities. In this dissertation, our focus is on pose based human activity recognition.

The goal of methods for human activity recognition from sequences is to extract expressive representations by mining the spatio-temporal patterns from the aforementioned modalities. Thus, the key components of an activity approach are usually two-fold: (i) a framework for modeling the sequential data, and (ii) mechanisms for mining the spatio-temporal patterns within the sequential framework.

The choices of sequential modeling approach for early-stage methods include sliding window [1], dynamic time warping [2], Hidden Markov Model (HMM) [3], etc. However, most of these sequential modeling approaches provide rigid processing (e.g., segmentation or hard categorization) over sequences, which restricts the model’s flexibility and capacity. The spatio-temporal patterns mined by the early-stage methods are mainly about the statistical handcrafted features regarding body parts for individual activities. These handcrafted features are intuitive and simple yet effective. However, since most early-stage methods feed the handcrafted features to shallow models (e.g., SVM and decision tree) for classification. The potential of these features cannot be fully exploited because the shallow model cannot handle the problem of “linearly nonseparable” well. The deep neural network (DNN) models
which stack layers of non-linear transformations are suitable for resolving this problem. However, DNN models require more computational resources due to its huge amount of parameters. With the maturity of the GPU platform, the computational bottleneck of DNN models is removed. So researchers started addressing human activity recognition problems with the resort to DNN models such as recurrent neural networks (RNN) and convolutional neural networks (CNN).

RNN is a type of DNN that is specially designed for modeling the sequential dependency of different steps of a sequence. The key difference between RNN and the feed-forward DNN is the recurrent connection. By unfolding the recurrent connection, a RNN model can be generalized (in theory) to handling sequences of different lengths thanks to the shared parameter of the recurrent connection. So RNN is a natural choice for modeling human activity sequences [4, 5]. One shortcoming of the vanilla RNN is the difficulty in modeling the long-term sequential dependency due to the constant “forgetting speed”. To this end, researchers developed Long Short-Term Memory (LSTM) [6] as a novel replacement of neurons in the vanilla RNN, in which the “forgetting speed” can be adjusted adaptively. However, the capability of such an adaptive modeling would be overstretched given the high complexity of the activity sequences. To this end, researchers started exploring to model the long-term sequential dependency in a more controllable fashion.

A majority of such attempts is to process the time steps via a hierarchical architecture in which nearby steps interact at lower layers while distant steps interact at higher layers. This provides a shorter path to capture long-term dependencies more effectively and efficiently than the chain structure modeled by RNN. The deep convolutional neural network (CNN) [7, 8] is a natural choice for such an approach.

Unlike usual low-dimensional sequential data, human activity sequences contain abundant spatial graphical dependencies[1] and thus require sophisticated design for modeling such dependencies. However, there are no such a sophisticated design for both RNN and CNN models. Therefore, researchers started exploring how to model the spatial graphical dependencies of human activities. A majority of the explorations focus on aggregating the contextual information to each node in the “activity graph”.

[1] For individual activity, the human body parts compose a graph; for group activity.
The major methodology being used herein is the graph CNN (GCN), which performs the convolution operations on the graph instead of on the regular grid as normal CNN does. GCN is powerful in modeling the spatial graphical dependencies of human activities, as this thesis will demonstrate with state-of-the-art results in individual activity recognition and group activity recognition.

1.1 Structure of the Thesis

The chapters are roughly grouped in to three parts: background material of human activity recognition is presented in Chapters 2 and 4. In Chapters 3, 5 and 7, we introduce the methodologies following the chronical order of their developments, respectively.

Chapter 2 briefly reviews the definition and data of human activities, and it also discusses the classical methodologies of sequence recognition.

The introduction of methodologies follows the chronological order which can be decomposed into three categories: (i) the handcrafted features for pose based activity recognition in Chapter 3 (ii) the temporal modeling with deep learning in Chapter 5 and (iii) the spatial modeling with deep learning in Chapter 7.

Before the introduction of the deep learning related methods in Chapters 5 and 7, we first review the background material for DNN in Chapter 4. In Chapter 8, we indicate the potential future directions.
CHAPTER 2
Human Activity Recognition

2.1 Human Activities

2.1.1 Definition

In daily life, the term “human activities” comprises of a variety of meanings, such as the body motions in the physical world, the online behaviors, or the financial behaviors, etc. In computer vision and pattern recognition, the definition of human activities is ambiguous and covers several categories (examples shown in fig. 2.1), e.g., (a) hand gestures, (b) body pose variations of a single person, (c) the interactions between a pair of persons, (d) collective body pose variations of a group, etc.

To summarize, the definition of “human activities” in computer vision is close to that the intentional motions performed by human body(s). By “intentional”, it means that there are patterns in the motions, so it is meaningful to recognize these motions via mining these patterns.

2.1.2 Data Modalities

To analyze human activities, researchers use a variety of sensors to acquire the data, such as cameras or depth sensors. The most common and cheapest modality is the RGB video. The depth maps captured by the depth sensors are either cheap and unreliable (collected by Kinect) or expensive and slow (collected by Lidar). As the maturity of the human pose estimation techniques, the pose key points or skeletons nowadays become an important modality. We visualize the three modalities in fig. 2.2. An observation aligned with other multi-modal recognition task is that different modalities are usually good at handling different scenarios. For instance, the human poses are sensitive to the human body motions but are agnostic to the contextual information and the background, so this modality is more suitable for motion-major activities, e.g., walking. The RGB videos are sensitive to the back-
Figure 2.1: Examples of different human activity categories. The image source for (a) is [50], for (b) and (c) is [73], and for (d) is [74].

ground or scene but are ambiguous to the foreground motions especially when the sensors are not fixed. So this modality is more suitable for scene-major activities, e.g., playing tennis.

2.1.3 Problem Formulation

Given an input sequence $X = (x_1, \ldots, x_t, \ldots, x_T)$, where $x_t$ is the activity data at the $t$-th frame and $T$ is the length of $X$. The goal of activity recognition is to infer the activity class labels $Y = (y_1, \ldots, y_t, \ldots, y_T)$ for $X$, where $y_t$ is the activity
Figure 2.2: Examples of different modalities: (a) samples frames of RGB videos with the human poses drawn; (b) depth maps.

Label(s) for the $t$-th frame. $Y$ may be different in form for different sub-topics (as shown in fig. 2.3). For example, (a) when assuming there is only one activity instance spanning over the entire sequence, $y_t$ is an activity index and uniform throughout $X$; (b) when assuming there are a certain number of non-overlapping activity instances throughout $X$, $y_t$ is still an activity index and may be different throughout $X$; (c) when assuming the activity instances are allowed to overlap, $y_t$ is a binary vector with each entry indicating the presence of an activity class. Therefore, the activity recognition can be formulated as a multi-class classification problem or the multi-class multi-label classification problem.
Figure 2.3: Examples of different formulations. A colored interval represents an activity instance.

2.2 Sequence Recognition

A sequence recognition model for human activities contains two components: (i) spatio-temporal dependency modeling module and (ii) classification module. Conventional sequence recognition models include the sliding window based approaches and the HMM based approaches. Both approaches require rigorous processing over sequences. Specifically, the sliding window based approaches limits the available temporal information by the window size. The HMM based approaches assumes that the hidden state for each time step can be categorized into one of the pre-defined state categories, and the activity label for each time step is determined by the pre-defined states and the emission probabilities describing the likelihood from the pre-defined states to the activity classes. Such an intermediate symbolic conversion limits the model flexibility of the HMM based approaches. Compared to these two approaches, the RNN and temporal CNN based approaches resolve the aforementioned restrictions effectively.

In RNN, the long-term dependency can be modeled by the recurrent spatio-temporal modeling module, which extracts the spatio-temporal features:

\[
h_t^{ST} = f^{ST}(x_t, h_{t-1}^{ST}),
\]

(2.1)

where \( f^{ST}(\cdot) \) denotes the recurrent module, and \( h_t^{ST} \) is the extracted spatio-temporal feature at the \( t \)-th step.
The extracted spatio-temporal feature $h_{t}^{ST}$ is then fed into the classification module $g(\cdot)$ to get the recognition result at the $t$-th step:

$$y_t = g(h_{t}^{ST}).$$  \hspace{1cm} (2.2)

The whole recognition pipeline of RNN is visualized in fig. 2.4. In RNN, the historical information can be passed to the current time step via the unfolded RNN.

In temporal CNN, the spatio-temporal features are extracted via a spatial module and a temporal convolutional module subsequently:

$$h_{t}^{S} = f^{S}(x_t)$$  \hspace{1cm} (2.3)

where $f^{S}(\cdot)$ denotes the spatial module, and $h_{t}^{S}$ is the extracted spatial feature.
where $f^T(\cdot)$ denotes the temporal module with the kernel size $2\tau + 1$. Fig. 2.5 shows the case where $\tau = 1$. By stacking more layers of $f^T(\cdot)$, longer-term dependencies can be modeled via the hierarchical framework. The extracted spatio-temporal feature $h^\text{ST}_t$ is fed into the classification module $g(\cdot)$ as in eq. (2.2) to get the recognition result.

2.2.1 Training

To train a sequence recognition model, we use a standard negative log likelihood loss function to measure the recognition performance, and minimize the loss using the back-propagation algorithm and stochastic gradient descent. The recognition loss
function is usually defined as follows:

\[
L = - \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log \left( \frac{\exp(y_{n,t}(n_c^*))}{\sum_{n_c=1}^{N_c} \exp(y_{n,t}(n_c))} \right),
\]  

(2.5)

where \( N \) and \( N_c \) represent the number of samples and classes of interest, respectively. \( T_n \) represents the length of the \( n \)-th sample. \( y_{n,t} \) denotes the logits for the \( n \)-th sample at the \( t \)-th time step. \( n_c^* \) represents the ground-truth class label.

### 2.2.2 Chronological Summary

In the early stage, the focus of the community is to design expressive handcrafted features for human activities, and classify the extracted features using shallow discriminative models, \textit{e.g.}, SVM. There are two drawbacks for the early methods: \( (i) \) the capability of the temporal model is limited due to some rigorous requirements; \( (ii) \) the expressiveness of handcrafted features are restricted by the limited non-linear transformations, which are insufficient for resolving the “linearly nonseparable” problem.

To this end, researchers started using RNNs, a major branch of DNNs, to overcome these two drawbacks. As briefly discussed in section 2.2, RNNs are naturally suitable for modeling the sequential dependencies. The performance gain is obvious after the exploitation of RNNs. However, there are still two accompanying drawbacks: \( (i) \) the capacity of the latent representation is very likely to be overstretched as the length of a sequence increases, so it is hard to model the long-range dependencies; \( (ii) \) the activity feature is usually represented as a latent vector which losses much spatial information, and such spatial information is critical to the discrimination.

Therefore, researchers resorted to temporal CNN for temporal modeling, in which the sequential dependencies of different ranges are captured by different levels in CNN. Furthermore, researchers extend the regular CNN to GCN for better spatial modeling, which is able to adaptively model the spatial dependencies of human body parts.

The remaining part of this dissertation presents the background material and our full spectrum of methods following the aforementioned chronological order.
CHAPTER 3
Handcrafted Features

3.1 Background

Conventional pose based activity recognition approaches attempted to follow the “anthropometry inspirations” to extract the handcrafted features. For example, Vemulapalli et al [9, 10] modeled the 3D geometric interactions between body parts using transformations, which are represented as elements in a Lie group. Gowayyed et al [11] represented 3D trajectories of body joints using the histogram of oriented displacements descriptor. Zanfir et al [12] proposed a moving pose descriptor that considers both pose information and the differential quantities of body joints. Wang et al [13] learned a subset of skeleton joints for each activity class. Wu et al [3] modeled transition dynamics of an activity using the Hidden Markov Model and developed a hierarchal dynamic framework to learn high-level features to estimate emission probability. Zhao et al [2] extracted structured streaming skeleton features to represent activities and constructed classifiers based on sparse coding features. Performance of these methods is bounded by their inability to model long-term evolution of discrete human poses, which are crucial for activity recognition.

Figure 3.1: Examples of human activities depicting the complexities in activity recognition: (a) a multi-person activity with each individual performing a single-person activity; (b) a multi-person activity involving a single-person activity and multi-person interaction activity; (c) a multi-person activity that cannot be reduced to the combination of two single-person activities. See texts for more details.
The majority of existing methods based on handcrafted features are designed for two general categories: single-person activity \cite{2,3,14} and multi-person interaction activity \cite{15,16,17}. However, human activities in real-world scenarios are much more complex because multiple activities belonging to both categories usually co-exist in a sequence. For example\footnote{We show frames from videos to illustrate the idea, but the same scenario holds for sequences of 3D joint positions.}, fig. 3.1 (a) describes three activities all belonging to the single-person activity category: three children are skating without interacting with each other; fig. 3.1 (b) depicts two activities belonging to two categories respectively: “two children are fighting for a toy” (a two-person interaction activity) while “a woman lifts two hands to hold her forehead” (a single-person activity). Thus, it is more desirable to have a method that can recognize human activities without involving any prior categorization.

However, existing handcrafted features based algorithms designed to recognize single-person activities from 3D pose sequences \cite{2,3,18} cannot be used to recognize multi-person interaction activities, and vice versa \cite{17,19}. This is because the two categories of activities are exclusive by definition. A simple approach fusing methods for different categories to recognize an activity in a competitive manner is unlikely to work as shown by the example in fig. 3.1 (c). From the perspective of single-person activity recognition, the image depicts one person pushing his hand, but considering the other person, the same activity can be recognized as a two-person interaction activity of “patting on the shoulder”.

In this chapter, we present a unified recognition model for single-person activities and multi-person interaction activities. As such, we term our method as Category-blind Human Action Recognition Method (CHARM). Given a sequence of 3D poses of each person in a video, we first generate possible combinations of mutually inclusive potential activities\footnote{We define that two potential activities without any common person involved as mutually inclusive.}. We then model potential activities with a category-blind visual representation, which models an activity as a set of weighted graphs (one for each activity class) with the same topology. In each weighted graph, nodes represent \textit{motion primitives}, and edges represent the co-occurrence of two motion primitives in a particular activity class. The weight on an edge represents the co-
occurring probability of two motion primitives. The likelihood of a potential activity being classified into a particular class is computed by identifying a maximum clique of motion primitives from the corresponding weighted graph. Then CHARM can classify a potential activity by identifying the class with the maximum likelihood score. After all potential activity are classified into potential activity classes with their associated likelihood scores, CHARM computes the reliability score for each possible combination by averaging the likelihood scores of all involved potential activities. Finally, CHARM outputs the most reliable activity combination and identifies the person(s) performing each activity. The overall procedure that is more systematic is presented in section 3.2

CHARM includes the following major contributions: First, we design a category-blind visual representation which allows an activity instance to be modeled as a set of weighted graphs which encode the co-occurring probabilities of motion primitives (section 3.2.1). Second, such a category-blind visual representation allows the recognition of co-existing activities of different categories (section 3.2). Third, we design a novel activity classification algorithm based on finding maximum cliques of motion primitives on the weighted graphs of the motion primitive co-occurrence patterns (section 3.2.2). Finally, we collect a dataset to evaluate the performance of CHARM in scenarios where activities of different categories co-exist (section 6.3).

3.2 Method

In this section, we describe the procedures of CHARM. As shown in fig. 3.2, the input to CHARM is a 3D pose sequence. Given the input, the goal of CHARM is two-fold, that is, recognizing all activities occurring in this video and identifying the person performing each activity. CHARM entails the following steps: (a) CHARM enumerates potential activities for the current sequence, e.g., determining the number of persons and the number of pairs of persons. (b) CHARM generates possible combinations of mutually inclusive potential activities, e.g., with three persons, there are four possible combinations as shown in fig. 3.2. (c) It extracts relevant body motion

3The human poses are estimated and tracked by the Microsoft Kinect SDK, so the 3D joint positions for different persons can be distinguished.
Figure 3.2: Main steps of CHARM. This example involves 3 persons (i.e., Person A, B, and C) and 6 potential activities. The recognition result indicates that A is performing a single-person activity of Class 1, and B&C are performing a two-person interaction of Class 10.

data from these potential activities. (d) CHARM extracts category-blind patterns for the current potential activity based on the information available in the reference repositories. Such reference repositories are constructed in the training stage. (e) The extracted category-blind patterns are fed to the classification module. (f) The classification module classifies the current potential activity and outputs an activity label with an associated likelihood score. (g) After all potential activities are classified, CHARM computes a reliability score for each possible combination by averaging the likelihood scores of all involved potential activities, and chooses the most reliable combination as the recognition result.

Steps (c), (d), (e), and (f) are four core steps of the overall procedure of CHARM. These four steps are divided into two phases: steps (c) and (d) model a potential activity using the category-blind visual representation, and steps (e) and (f) classify a modeled potential activity into an activity class by solving several maximum clique problems (MCP). These two phases are presented in section 3.2.1 and section 3.2.2.
respectively.

3.2.1 Modeling a Potential Activity Instance using the Category-blind Visual Representation

In CHARM, we model a potential activity instance using a category-blind visual representation so that we can directly compare the likelihood scores of any two potential activities which belong to different categories.

We assume that a human activity can be represented as a combination of motion units (MUs). For a single-person activity, an MU corresponds to the motion of a single body part (e.g., the right upper arm), while for a two-person interaction activity, an MU corresponds to the motions of a pair of body parts from two interacting persons (e.g., a pair of right upper arms). Thus, an activity instance of any activity category can be modeled using the category-blind visual representation in two steps: (i) First, we model all MUs of an activity instance (section 3.2.1.1) and then (ii) model the combinations of MUs that can match potential activity instances (section 3.2.1.2).

3.2.1.1 MU Model

Let us first consider how to model MUs of a single-person activity. Given an input 3D pose sequence, there are many ways to represent body part configurations. In CHARM, we adopt the approach used in [21], which uses eight bilateral symmetrical human body parts, i.e., upper arms, lower arms, thighs, and legs. The reason for choosing this body part configuration representation is that it is invariant to the body position, orientation and size, because the person-centric coordinate system is used, and the limbs are normalized to the same length. As shown in fig. 3.3 all the 3D joint coordinates are transformed from the world coordinate system to a person-centric coordinate system. Upper arms and thighs are attached to the torso at the ball-and-socket joints and move freely in 3D. These four body parts are modeled as four 3D unit vectors \( v_1, v_2, v_3, \) and \( v_4 \) as shown in fig. 3.3 and are computed from the coordinates of their endpoints. Lower arms and legs can only bend \( 0^\circ - 180^\circ \) at the elbow and knee joints. Thus, we model their relative positions with respect to the upper body parts, e.g., the upper arms and thighs using four angles \( \alpha_1, \alpha_2, \alpha_3, \) and
Figure 3.3: The representation of body part configurations.

\[ \alpha_4 \] as shown in fig. 3.3. These four angles are planar angles because a upper body part and its corresponding lower body part are represented as two intersecting line segments in CHARM. Besides these angles, we also keep track of the planes containing the upper and lower arms which are represented by the unit normals \( n_1 \) and \( n_2 \) to the planes. We assume the normal direction of the plane formed by legs and thighs remains unchanged with regards to the human centric coordinate system, since the lower leg does not move flexibly with regards to its upper thigh. The four 3D unit vectors \( \{v_i\} \), four planar angles \( \{\alpha_i\} \), and two 3D normals \( \{n_i\} \) form a 22-element body pose vector. Thus, the MUs of a single-person activity can be collectively represented as a body motion matrix, with each row corresponding to a dimension of the body pose vector, and each column corresponding to a particular video frame. Note that each row of the body motion matrix is referred to as a dimension.

A natural choice of modeling MUs of a two-person interaction activity is to directly use two body motion matrices, one for each person. However, doing so does not capture inter-person temporal correlations, which are very important cues in recognizing two-person interaction activities. Hence, we augment the MU model
with additional inter-person temporal correlations for two-person interactions. We use four types of inter-person temporal correlations, which we will illustrate using the following example. For Persons $A$ and $B$, we can represent the world coordinates of the origin and coordinate axes of their person-centric coordinates as \{$o^A_w, x^A_w, y^A_w, z^A_w$\} and \{$o^B_w, x^B_w, y^B_w, z^B_w$\}. The Euclidean distance between $A$ and $B$ can be represented as $d_{AB} = \|o^A_w - o^B_w\|^2$, and the angles between corresponding coordinate axes can be represented as $\alpha_x$, $\alpha_y$, and $\alpha_z$. $d_{AB}$, $\alpha_x$, $\alpha_y$, and $\alpha_z$ can form a 4-element inter-person correlation vector. Thus, the inter-person temporal correlation of two persons can be represented as an inter-person temporal correlation matrix, with each row corresponding to a dimension of the inter-person correlation vector, and each column corresponding to a video frame.

### 3.2.1.2 MU Combination Model

A natural choice for modeling the combination of MUs is to concatenate the representations of all individual MUs. However, due to the complexity of MUs, similar MUs need not be numerically similar. The variations between similar MUs will complicate the training of classifiers with potential overfitting problems. Inspired by [22], we assume an MU can be quantized into several motion primitives (MPs) which are common patterns for each dimension shared by a variety of human activities.

As such, the task of modeling the combination of MUs for a potential activity can be formulated as modeling the combination of MPs of this potential activity. We first discuss how to quantize the MUs of a potential activity to form a collection of MPs. Using the MU model described in section 3.2.1.1, the MUs of a single-
Figure 3.4: The upper row shows how an MP collection can be formed based on the extracted body motion matrix using the motion template codebook. The lower half is about the extraction of the MP co-occurrence patterns. Red arrows indicate the quantization procedure. The light grey blocks are reference repositories. In the block that shows the MP co-occurrence pattern, each activity class should have 22 disjoint groups (each group is illustrated as an ellipse) with each corresponding to a dimension. However, we merely show five disjoint groups to make the diagram clearer. The notation used herein and the rest of the paper are defined in table 3.1.

Person activity is represented as a body motion matrix, and the MUs of a two-person interaction is represented as two body motion matrices. The formation of MP collection relies on a reference repository, namely motion template codebook (see section 3.2.1.3) which stores a number of motion templates identified from the training activity instances. An MP for a single-person activity is obtained by finding its best match motion template in the codebook while an MP for a two-person interaction is obtained by selecting the best pair of motion templates in the codebook. The formation of MP collection for both activity categories is described as follows:

(i) As shown in the upper half of fig. 3.4, given the body motion matrix of a potential single-person activity, each dimension of the body motion matrix is matched to $K$ nearest motion templates from the same dimension in the codebook. Each matched motion template becomes an MP for the corresponding dimension. The intuition of
generating multiple MPs for each dimension is that the skeletal data collected by
the depth camera might be noisy due to some degrading factors, e.g., illumination
changes, and the noisy data will impact the quantization from the MUs to MPs; thus,
we generate multiple MPs per dimension to increase the tolerance to the quantization
error.

(ii) The formation of an MP collection for a two-person interaction is similar to
the single-person activity case. Given two body motion matrices of two interacting
persons, we first respectively match each dimension of their body motion matrices
to $K$ nearest motion templates on the same dimension. Then, a pair of matched
motion templates (one for each person) from the same dimension compose an MP of
a two-person interaction.

**MP Co-occurrence Pattern Extraction.** Based on the MP collection, we
model a potential activity by extracting its MP co-occurrence pattern (with each pat-
tern representing a combination of MPs) for each class. To extract such co-occurrence
patterns, we rely on a reference repository, namely the *MP co-occurrence matrices*
(see section 3.2.1.3), with each matrix representing the conditional probabilities for
any two MPs co-occurring in an activity instance of a class.

We define the MP co-occurrence pattern of a potential activity for class $c$ as
a weighted graph $G_c = \{V_c, E_c, \Omega_c\}$, where $V_c$, $E_c$, $\Omega_c$ represent the set of nodes,
edges and edge weights respectively. Each node represents an MP. The nodes in $V_c$
are divided into 22 disjoint groups (see fig. 3.4) with each group $R$ corresponding to
a dimension. Thus, $R_j = \{P_j^1, P_j^2, \ldots, P_j^K\}$, where $P_j^i$ denotes the $i$th MP of the
$j$-th group. $K$ is defined in table 3.1, and $H$ is the number of persons involved in this
action. $E_c$ includes the edges of $G_c$ that connect nodes from different groups, but
not within the same group. The edge weights correspond to the pairwise co-occurring
probabilities of MPs, and such co-occurring probability is determined by two factors:
(i) the conditional co-occurring probabilities of the two MPs and (ii) the confidence
level of using a particular MP to represent a dimension of the potential action. $G_c$
is an undirected graph with the symmetrized edge weights calculated by averaging the
two weights for different directions:

\[
\Omega_c(P_i^j, P_l^m) = \frac{1}{2} \cdot (\Phi'_c(P_i^j, P_l^m) \cdot \varphi(P_i^m)
+ \Phi'_c(P_l^m, P_i^j) \cdot \varphi(P_i^j)),
\]

(3.1)

where \(\Phi'_c(P_i^j, P_l^m) = \Phi_c(P_i^j, P_l^m) \cdot (O_c)^{\frac{1}{2}}\), and \(\Phi_c(P_i^j, P_l^m)\) is an entry of the MP co-occurrence matrix of class \(c\), indicating the conditional probability of \(P_i^j\) occurring in an activity instance of class \(c\) given that \(P_l^m\) occurs. \((O_c)^{\frac{1}{2}}\) is a parameter used to normalize the effects of different co-occurrence matrix sizes on the value of \(\Phi_c(P_i^j, P_l^m)\), and \(O_c\) is the order of the MP co-occurrence matrix of class \(c\). \(\varphi(P_l^m)\) reflects the confidence level of using \(P_l^m\) to represent a dimension of the potential activity:

\[
\varphi(P_l^m) = \exp(-\beta \cdot \frac{1}{H} \cdot \sum_{h=1}^{H} \Delta(T_{l,h}^m, B_{m,h})).
\]

(3.2)

where \(\beta\) is a sensitivity controlling parameter, and \(H\) is the number of persons involved in this potential activity. We have \(P_l^m = \{T_{l,h}^m\}_{h=1}^H\) where \(T_{l,h}^m\) is a matched motion template of the \(h\)th person used to generate \(P_l^m\). \(B_{m,h}\) is the \(m\)th dimension of the body motion matrix of the \(h\)th person. \(\Delta(\cdot, \cdot)\) is the dynamic time warping distance [23].

**Inter-person Temporal Correlation Pattern Extraction.** To augment the MP combination modeling of a potential two-person interaction, we extract its inter-person temporal correlation pattern for each interaction class. Such a temporal correlation pattern is represented as a confidence score, which describes how well the inter-person temporal correlations of this potential interaction instance is aligned with the correlation distribution of a specific interaction class. The confidence score of class \(c\), \(\ell_c\), is computed as:

\[
\ell_c = \prod_{i=1}^{4} \delta(f_{i,c}(I_i) > \tau_i).
\]

(3.3)

\(I_i\) is the \(i\)th row of the inter-person temporal correlation matrix (see section [3.2.1.1]).

---

\(^4\)It can be easily seen that if class \(c\) is a single-person activity class, \(O_c = N \cdot C\); if class \(c\) is a two-person interaction class, \(O_c = N \cdot C^2\)
$f_{i,c}$ is a Gaussian distribution which models the distribution of a temporal correlation type for class $c$. $\delta$ is a delta function that takes 1 when the condition is true, and 0 otherwise. $\tau_i$ is a threshold. Relevant Gaussian models associated with all interaction classes are stored in a reference repository (see section 3.2.1.3). Since a single-person activity does not have inter-person temporal correlations, we set the values of these confidence scores to 1 for a potential single-person activity to create a uniform representation.

3.2.1.3 Construction of Reference Repositories

**Motion template codebook.** We start with a training dataset with sequences covering activity classes of interest. Each training sequence contains only one activity instance, and is associated with an activity class label. Each person in the training sequence is represented by twenty 3D joint positions. To construct the motion template codebook, two steps are performed: First, we use the MU model described in section 3.2.1.1 to represent the MUs of each person in the training sequences as a body motion matrix. Second, we pull all the body motion data from the same dimension together and apply k-means clustering to obtain $C$ clusters per dimension. Then, we store the centroid of each cluster as a motion template in the codebook. We adopt dynamic time warping distance [23] in the clustering process so that we can cope with training sequences of different lengths.

**MP co-occurrence matrices.** Given the MUs of each person in the training sequences modeled as a body motion matrix with $N$ dimensions, we can match each dimension of the body motion matrix to the nearest motion template on the same dimension of the codebook and hence represent an activity instance as a collection of $N$ MPs with each MP corresponding to a dimension. Next, we construct an MP co-occurrence matrix for each activity class. We construct the MP co-occurrence matrix for an activity class $c$, by computing the conditional probabilities of any two MPs using the following equation:

$$
\Phi(c, P_i^j, P_l^m) = p(P_i^j | P_l^m, c) = \frac{\Theta(c, P_i^j, P_l^m)}{\Theta(c, P_l^m)} ,
$$

(3.4)
where $\Theta_c(\mathcal{P}_j^i)$ is the number of training activity instances of class $c$ containing MP $\mathcal{P}_j^i$ and $\Theta_c(\mathcal{P}_j^i, \mathcal{P}_l^m)$ is the number of activity instances of class $c$ where $\mathcal{P}_j^i$ and $\mathcal{P}_l^m$ co-occur. If the denominator $\Theta_c(\mathcal{P}_l^m)$ equals zero, then $\Phi_c(\mathcal{P}_j^i, \mathcal{P}_l^m)$ is directly set as zero. However, if $\mathcal{P}_j^i$ and $\mathcal{P}_l^m$ are MPs corresponding to the same dimension, i.e., $j = m$, $\Phi_c(\mathcal{P}_j^i, \mathcal{P}_l^m)$ is set to zero, i.e., we do not allow them to co-occur since we only allow one MP for each dimension for any activity instance. As shown in (3.4), the element $\Phi_c(\mathcal{P}_j^i, \mathcal{P}_l^m)$ is equivalent to the conditional probability $p(\mathcal{P}_j^i | \mathcal{P}_l^m, c)$ which is the probability of $\mathcal{P}_j^i$ occurring in an activity instance of class $c$ given that $\mathcal{P}_l^m$ occurs. The advantages of defining a co-occurrence matrix using the conditional probability are: (i) It enhances the robustness of CHARM such that it can tolerate activity variations caused by personal-styles more because the conditional probability does not penalize those co-occurrences of MPs that happen less frequently. (ii) The resulting co-occurrence matrix is asymmetric; such asymmetry property is helpful for coping with intra-class variations because it ensures that the probability of $\mathcal{P}_j^i$ occurring in an activity instance given that $\mathcal{P}_l^m$ occurs is not necessarily equivalent to that of the reverse situation.

Gaussian models with respect to each interaction class. There are four Gaussian models for each interaction class, with each modeling the distribution of an inter-person temporal correlation type for this interaction class. The mean and standard deviation of each Gaussian model are computed using the relevant data for that correlation type from all training instances for that interaction class.

3.2.2 Activity Classification using MCP

The category-blind representation of a potential activity includes a set of class-specific patterns. We compute a likelihood score for each class-specific pattern and then choose the class label associated with the highest score as the label for that potential activity. Recall that each class-specific pattern can be represented as a weighted graph, e.g., $\mathbf{G}_c = \{\mathbf{V}_c, \mathbf{E}_c, \Omega_c\}$ for class $c$. Since in the MP co-occurrence pattern extraction process, we include multiple MPs per dimension, for each class $c$, we thus first need to identify a subgraph, $\mathbf{G}_s = \{\mathbf{V}_s, \mathbf{E}_s, \Omega_s\}$, from $\mathbf{G}_c$, which includes only one MP per dimension. A feasible $\mathbf{G}_s$ has to satisfy the following three
constraints: (i) Given $\mathcal{N}$ disjoint groups (dimensions), one and only one node from each group should be selected. (ii) If one node is selected to be in $G_s$, then exactly $(\mathcal{N} - 1)$ of its edges should be included in $G_s$ (this is because each selected node should be connected to one node at each of the rest $(\mathcal{N} - 1)$ groups). (iii) If an edge is included in $G_s$, the nodes incident to it should be also included and vice versa.

The metric that we use to identify a $G_s$ from $G_c$ is the co-occurring likelihood of the clique of MPs in $G_s$. Such a co-occurring likelihood is defined as follows:

$$\lambda(G_s) = \sum_{p=1}^{\mathcal{N}} \sum_{q=1, q\neq p}^{\mathcal{N}} \Omega_c(V_s(p), V_s(q)).$$

(3.5)

where $V_s(p)$ denotes the node within the $p$th group of $G_s$. Thus, we formulate the identification process of a subgraph $G_s^*$ which contains the MP clique that is most likely to occur, as the following optimization problem:

$$G_s^* = \arg \max_{V_s} \lambda(G_s).$$

(3.6)

Once we have such $G_s^*$ for each activity class, we compute a likelihood score, $\Upsilon_c$, which measures how likely we should classify this potential activity to a particular activity class $c$:

$$\Upsilon_c = \ell_c \cdot \lambda(G_s^*, c).$$

(3.7)

where $\ell_c$ is defined in (5.1). Eventually, we classify this potential activity to the class which yields the highest likelihood score. Such a classification task is formulated as follows:

$$c^* = \arg \max_c (\Upsilon_c).$$

(3.8)

The optimization problem in (5.4) is a maximum clique problem (MCP) that is NP-hard. Several approximation algorithms exist for solving MCP, e.g., [24, 25, 26]. In this work, we adopt a mixed binary integer programming (MBIP) based solver [25]. We set the objective function of the MBIP as the optimization problem of (5.4), and set the constraints of the MBIP as the formulation of three aforementioned feasibility constraints. The MBIP is solved by the Cplex [27].
3.3 Experiments

For evaluation, we use three test scenarios: (i) videos with three persons and a mixture of single-person activities and two-person interactions, (ii) videos with two interacting persons, and (iii) videos with a single person. The test scenario (i) is closer to real-world scenarios where the category of a video sequence is not given, and multiple activity instances belonging to different categories co-exist in a video sequence. This scenario is used to highlight the advantage of our CHARM approach where no prior category information is given. In contrast, each video sequence in test scenario (ii) and (iii) contains only one activity instance at one time and its activity category (either single-person activity or two-person interaction) is predefined.

Since no prior existing datasets include test scenario (i), we collect our own “Hybrid Actions3D” dataset\(^5\). In addition, we adopt SBU Interaction dataset [17] for test scenario (ii), and MSRC12-Gesture dataset [28] for (iii).

**Baselines.** Since no prior work is designed to handle test scenario (i), we create a baseline scheme called “SimpleComb” which combines two dedicated approaches: one is used to recognize single-person activities and the other one is used to recognize two-person interactions. For the single-person activity recognition, we use the approach described in [3]. Since there is no publicly available code for previous two-person interaction recognition approaches, we use our CHARM by limiting it to only deal with two-person interactions. Each dedicated approach will label a potential activity with a particular activity class associated with a likelihood score. Before we decide which mutually inclusive activity combination (see section 3.2) to be the recognition result, we first scale the likelihood scores of both approaches so that their values are comparable. Then, we choose the combination which yields the highest reliability score computed as described in step (g) of CHARM (see section 3.2).

For test scenario (ii), we compare CHARM with two previous approaches (based on Linear-SVM, and MILBoost respectively) described in [17]. For test scenario (iii), we compare CHARM with four state-of-the-art approaches [2, 3, 28, 29].

**Parameter Settings.** We use the same motion template codebook for both single-person activity recognition and two-person interaction recognition. We set the

\(^5\)The dataset is available at: https://www.albany.edu/wil523363/main.html.
default values for the parameters of CHARM as follows: The number of motion dimension of this codebook is $N = 22$, and the number of motion templates per dimension is $C = 22$. In the recognition phase, when we quantize the body motion data into MPs, we match each body motion data with $K = 2$ nearest motion templates. The sensitivity controlling parameter in (3.2) is $\beta = 0.1$. The thresholds for four types of temporal correlation in (5.1) are: $\tau_d = 2 \times 10^{-5}$, $\tau_x = 10^{-12}$, $\tau_y = 0.1$, and $\tau_z = 10^{-12}$.

3.3.1 Action Recognition using Hybrid Actions3D

Datasets. Hybrid Actions3D is captured using the Kinect camera [30], which tracks human skeletons and estimates the 3D joint positions for each person at each frame. This dataset includes 10 activity classes, 5 of which are single-person activity classes (i.e., duck, goggles, shoot, beat both, and throw), and the remaining ones are two-person interaction classes (i.e., kick, push, punch, hug, and exchange). The single-person activity classes are from the MSRC12-Gesture [28], and the two-person interaction classes are from the SBU Interaction [17]. 10 volunteers were recruited to create this dataset. We first ask each volunteer to perform single-person activi-
ties. Next, we formed 14 pairs out of these 10 volunteers and ask them to perform two-person interactions. In total, this dataset contains 910 pre-segmented video sequences including 580 for training and 330 for testing. Specifically, there are 280 two-person interaction training sequences (56 sequences per activity class) and 300 single-person activity training sequences (60 sequences per activity class). The 330 test sequences consist of (i) 280 hybrid-category sequences (constructed by fusing relevant sequences), each contains a two-person interaction instance and a single-person activity instance, and (ii) 50 mono-category sequences containing three single-person activity instances. The activity instances in a test sequence are not required to have the same starting and ending time points. Some sample frames of the test sequences in the Hybrid Actions3D dataset are shown in fig. 3.5.

Evaluation metrics. Two metrics are used to evaluate the recognition capability of CHARM, namely (i) the sequence-level accuracy, and (ii) the activity-level accuracy. Since each test sequence in our Hybrid Action3D dataset contain more than one activity instance, for sequence-level accuracy, we consider a recognition result as accurate only if all activity instances are classified correctly. On the other hand, every activity instance that is correctly classified is counted towards the activity-level accuracy.

Table 3.2 shows the results for the test scenario (i). It shows that CHARM yields better performance than SimpleComb in terms of both sequence-level accuracy and activity-level accuracy. We also observe that although SimpleComb uses the CHARM as a dedicated approach to recognize two-person interactions, its two-person interaction accuracy is lower than that of our CHARM. We believe that such a performance gap is caused by the difficulties in merging seamlessly the results from the two dedicated approaches such that the benefits of each dedicated approach cannot be fully utilized. We notice that the overall performance of CHARM for single-person activi-

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleComb</td>
<td>0.909</td>
<td>0.746</td>
<td>0.739</td>
</tr>
<tr>
<td>CHARM</td>
<td>0.921</td>
<td>0.811</td>
<td>0.800</td>
</tr>
</tbody>
</table>

Table 3.2: Comparison on Hybrid Actions3D (test scenario (i)).
ties is better than the performance for two-person interactions. In general, CHARM works well for most activities but tends to have problem classifying similar two-person interactions, e.g., “push” and “hug”. In CHARM, we did some tradeoffs between the expressiveness capability of our visual representation model towards any specific activity category and its capability for a uniform visual representation, e.g., compared to existing approaches (e.g., [17]), we only use a very simple model in CHARM to capture the inter-person temporal correlations.

3.3.2 Activity Recognition using SBU Interaction

Datasets. SBU Interaction dataset consists of approximately 300 pre-segmented two-person interaction sequences in the form of 3D joint positions. This dataset includes eight classes: approach, depart, push, kick, punch, exchange objects, hug, and shake hands. As in [17], we use the sequence-level accuracy as our evaluation metric.

We compare our CHARM with the two interaction recognition approaches in [17] using five-fold cross validation. As shown in table 3.3, the only approach that slightly outperforms CHARM is the MILBoost based approach [17], which uses spatio-temporal distances between all pairs of joints of two persons as feature. However, the MILBoost based approach [17] focuses on recognizing two-person interactions (test scenario (ii)), and cannot be used to cope with the test scenario (i) and (iii).

3.3.3 Activity Recognition using MSRC12-Gesture

Datasets. The MSRC12-Gesture dataset was collected by having volunteers perform certain activities either based on “Video + Text” based instructions or based on “Image + Text” instructions. We refer to these different procedures as different modality. This dataset is chosen to validate the effectiveness of CHARM to handle the streaming data sequences. It contains 594 sequences collected from 30 people.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint Features + Linear SVM [17]</td>
<td>0.687</td>
</tr>
<tr>
<td>Joint Features + MILBoost [17]</td>
<td>0.873</td>
</tr>
<tr>
<td>CHARM</td>
<td>0.839</td>
</tr>
</tbody>
</table>

Table 3.3: Comparison on SBU Interaction (test scenario (ii)).
performing 12 different single-person activities, including lift outstretched arms, duck, push right, goggles, wind it up, shoot, bow, throw, had enough, change weapon, beat both, and kick. Each sequence may contain multiple activity instances, thus there is a total of 6244 activity instances in this dataset. All sequences in this dataset are non-segmented, i.e., there do not exist information about where the starting and ending times of an activity instance are within a sequence. We only know the ending points of all activity instances since they are manually labeled by the authors who release this dataset. The authors indicate that any recognition system which can correctly identify the ending time of an activity instance within $\pm \xi = 10$ video frames should be considered as accurately identify this activity instance.

To use CHARM on a non-segmented streaming data sequence, as in [28], we use a 35-frame sliding window to continuously segment potential activity instances from the streaming data. In addition, inspired by [14], we introduce a background class and use a threshold for each class such that we can balance the precision and recall rates of our CHARM-based recognition system. Specifically, we redefine (3.8) as $\epsilon^* = \arg \max_\epsilon (\Upsilon_\epsilon)$, s.t. $\Upsilon_\epsilon > \theta_\epsilon$. As in [2, 14, 28], the optimal $\theta_\epsilon$ is chosen such that it minimizes the recognition error, e.g., we set the threshold $\theta_b$ for the background class to be zero. We conduct our experiments using the same test procedure described in [3] where training and test sequences can potentially come from different modality. We did two groups of experiments, namely “intra-modality” and “inter-modality”. “Intra-modality” indicates that training and test sequences are from the same modality, e.g., both collected using “Video+Text” instructions. “Inter-modality” indicates that training and test sequences are collected using different modality. It is clear from our results that sequences using the “Image+Text” instructions tend to have more variations and hence lower recognition accuracy.

As in [3], we use the criteria, F-score which combines the precision and recall to evaluate the performance of different activity recognition methods. Table 3.4 shows that CHARM performs better than all baseline methods for single-person activity recognition for the intra-modality scenario. For the inter-modality scenario, CHARM performs better than [28] and yields comparable performance to [3]. Methods in [2] and [29] are not compared because the authors neither publish their performance for
<table>
<thead>
<tr>
<th>Method</th>
<th>intra-modality</th>
<th>inter-modality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomized Forest [28]</td>
<td>0.621</td>
<td>0.576</td>
</tr>
<tr>
<td>Structured Streaming Skeletons [2]</td>
<td>0.718</td>
<td>N\A</td>
</tr>
<tr>
<td>Multi-scale Activity Detection [29]</td>
<td>0.685</td>
<td>N\A</td>
</tr>
<tr>
<td>DBN-ES-HMM [3]</td>
<td>0.724</td>
<td>0.710</td>
</tr>
<tr>
<td>CHARM</td>
<td>0.725</td>
<td>0.700</td>
</tr>
</tbody>
</table>

Table 3.4: Comparison on MSRC12-Gesture (test scenario (iii)). "N\A" indicates that experimental results of the corresponding approaches are not available mainly because the authors neither provide the results in their paper nor publish their code.

3.4 Remarks

There are two major drawbacks of the conventional approaches. First, from the perspective of spatial modeling, they use the shallow model to learn the representations or just use the handcrafted representations, both of which limit the expressiveness. Second, from the perspective of temporal modeling, they use the temporal model with strong restrictions, e.g., sliding window, which restricts the flexibility of the temporal modeling. Therefore, researchers kept exploring models with higher flexibility and adaptability. It turns out that DNNs are helpful for overcoming these drawbacks, which are introduced in the following chapters.
CHAPTER 4
Deep Neural Networks

The recent human activity recognition methods have been mainly developed based on DNNs. Therefore, we revisit three types of DNNs which are major components of recent methods, including RNN (section 4.2), CNN (section 4.1) and GCN (section 4.3).

4.1 Recurrent Neural Networks

While the difference between a feed-forward DNN and a RNN may seem trivial, the implications for sequence learning are far-reaching. A feed-forward can only map from input to output vectors, whereas a RNN can in principle map from the entire history of previous inputs to each output. The key point is that the recurrent connections allow a ‘memory’ of previous inputs to persist in the network’s internal state, and thereby in influence the network output.

Forward Pass. The forward pass of a RNN is the same as that of a feed-forward DNN with a single hidden layer, except that activations arrive at the hidden layer from both the current external input and the hidden layer activations from the previous time step, which is defined in eq. (2.1).

Backward Pass. The backward pass of a RNN is slightly different from that of a regular feed-forward DNN because of the introduced recurrent connection. The most popular backpropagation algorithm for RNN is the “Backpropagation Through Time” (BPTT) [6], which repeatedly applies the chain rule of a standard backpropagation.

4.2 Convolutional Neural Networks

Two typical components of a layer in DNN are the learnable linear transformation and a non-linear transformation. The advantages of DNN over shallow machine learning model are two-fold: (i) the sufficiency of degree of non-linear transforma-
tions can be satisfied by stacking many layers, so that the the problem of “linearly nonseparable” can be resolved; (ii) the coupling of the non-linear transformation and the learnable linear transformation makes DNN able to approximate any functions in theory.

A regular DNN receives an input (a single vector), and transforms it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the output layer and in classification settings it represents the class scores. In fig. 4.1, we show an example of three-layer fully-connected NN for binary classification.

The regular DNNs are powerful yet due to the fully-connected fashion, the amount of learnable parameters is increased significantly as the dimension of input scales up. This comes up with two problems: (i) the required computational cost is also increased significantly; (ii) the search space for optimization is expanded remarkably. These two limitations weaken the practicability, effectiveness and efficiency of the regular DNNs. To this end, researchers [31] proposed CNN with the property of sharing parameters via the convolution operations. By sharing parameters, the two aforementioned challenges are resolved effectively.

4.2.1 Architecture Overview

The convolution operations of CNN be applied onto input of different shapes, such as 1D vectors, 2D grids, 3D tensors, etc. The most common application of CNN is to extract features for images. Therefore, we use CNN over images as an example to briefly go through the basic architecture of CNN.

Unlike a regular DNN, the layers of a CNN have neurons arranged in 3 dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full NN, which can refer to the total number of layers in a network.)

The input images are an input volume of activations, and the volume has di-
dimensions $W \times H \times C$ (width, height, depth respectively). The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for classification task have dimensions $1 \times 1 \times N$ where $N$ is the number of classes, because by the end of the CNN architecture, we will reduce the full image into a single vector of class scores, arranged along the depth dimension. In fig. 4.2 we show an example of two-layer CNN for binary classification. Note that there are four convolutional kernels.
in “hidden layer 1”, which determines the channel number of its output activations. There is one convolutional kernel in the output layer, and the kernel size is align with the spatial shape of activations. Therefore, the shape of the final output is $1 \times 1 \times 1$.

4.2.2 Layers

As described above, a simple CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function. In this section, we discuss four main types of layers to build CNN architectures: Convolutional Layer, Pooling Layer, Normalization Layer and Nonlinear-transformation Layer.

4.2.2.1 Convolutional Layer

The conv layer is the core building block of CNN. The conv layers parameters consist of a set of learnable kernels. Every kernel is small spatially (along width and height), but extends through the full depth of the input volume. During the forward pass, we slide (convolve) each kernel across the width and height of the input volume and compute dot products between the entries of the kernel and the input at any position. As we slide the kernel over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that kernel at every spatial position. Intuitively, the network will learn kernels that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of kernels in each conv layer, and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

Local Connectivity. When dealing with high-dimensional inputs such as images, as we argued above it is impractical to connect neurons to all neurons in the previous volume. Instead, we will connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the kernel size). The extent of the
connectivity along the depth axis is always equal to the depth of the input volume. The connections are local in space (along width and height), but always full along the entire depth of the input volume.

We now formally define the computation within the conv layer. Given a convolution operator with the kernel size of $K \times K$, and an input feature map $f_{\text{in}}$ with the number of channels $c$. The output value for a single channel at the spatial location $x$ can be written as

$$f_{\text{out}}(x) = \sum_{h=1}^{K} \sum_{w=1}^{K} f_{\text{in}}(p(x, h, w)) \cdot w(h, w), \quad (4.1)$$

where the sampling function $p : Z^2 \times Z^2 \rightarrow Z^2$ enumerates the neighbors of location $x$. The weight function $w : Z^2 \rightarrow \mathbb{R}^c$ provides a weight vector in $c$-dimension real space for computing the inner product with the sampled input feature vectors of dimension $c$. Note that the weight function is irrelevant to the input location $x$. Thus the kernel weights are shared everywhere on the input image. Standard convolution on the image domain is therefore achieved by encoding a rectangular.

### 4.2.2.2 Pooling Layer

The function of Pooling layer is to reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX or AVG or L2-norm operation. The most common form is a pooling layer with kernels of size $2 \times 2$ applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every operation would in this case be taking over 4 numbers (little $2 \times 2$ region in some depth slice). The depth dimension remains unchanged.

### 4.2.2.3 Normalization Layer

To increase the stability of a CNN, researchers developed various normalization layers, e.g., Batch Normalization (BN) [32], Instance Normalization (IN) [33], Layer
4.2.2.4 Nonlinear-transformation Layer

Every Nonlinear-transformation layer (or activation function) takes a single number and performs a certain fixed mathematical operation on it. There are several classic Nonlinear-transformation layers, including Sigmoid, Tanh, ReLU and Leaky ReLU.

**Sigmoid** has the mathematical form $\sigma(x) = \frac{1}{1 + e^{-x}}$ which takes a real-valued number and “squashes” it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1). In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks: (i) sigmoids saturate and kill gradients; (ii) sigmoid outputs are not zero-centered which leads to all-positive or all-negative gradients, and further introduces undesirable zig-zagging dynamics in the gradient updates.

**Tanh** has the mathematical form $\tanh(x) = 2\sigma(2x) - 1$ which squashes a real-
valued number to the range $[-1, 1]$. Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity. Also note that the tanh neuron is simply a scaled sigmoid neuron.

**ReLU** computes the function $f(x) = \max(0, x)$. The activation is simply thresholded at zero. The major advantages of ReLU are two-fold: (i) due to the linear and non-saturating form, the convergence of stochastic gradient descent is significantly accelerated compared to sigmoid and tanh functions; (ii) The operations of ReLU are much cheaper compared to tanh or sigmoid functions involving expensive exponential operations. The drawback of ReLU is also obvious: ReLU units can “die” during training. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on.

**Leaky ReLU** is an attempt to fix the “dying ReLU” problem. Instead of the function being zero when $x < 0$, a leaky ReLU will have a small negative slope $\alpha$: $f(x) = \ell(x < 0)(\alpha x) + \ell(x \geq 0)(x)$.

### 4.3 Graph Convolutional Neural Networks

CNNs have been successfully applied onto regular grids. However, much data in computer vision are of irregular shapes, researchers proposed GCN for this scenario. Similar to CNN, the major function of GCN is also to aggregate the contextual features onto the feature of each position. The early GCN approaches fall into two categories, *i.e.*, the spectral-based and the spatial-based. The spectral-based approaches define graph convolutions by introducing filters from the perspective of graph signal processing [36]. Spatial-based approaches formulate graph convolutions as aggregating feature information from spatial neighbors. Recently, more proposals regarding GCNs are about how to generalize the graph connectivity, and how to extend GCN to the spatio-temporal domain.
4.3.1 Spectral Perspective vs. Spatial Perspective

4.3.1.1 Spectral-based GCN

The design of spectral-based GCN is based on the graph Laplacian matrix and the graph Fourier transform. So before introducing the spectral-based GCN, we first define these two key bases.

**Graph Laplacian Matrix.** The normalized graph Laplacian matrix is defined as

$$L = I_n - D^{-1/2} A D^{-1/2},$$

where $D$ is a diagonal matrix of node degrees, $D_{ii} = \sum_j A_{ij}$. The normalized graph Laplacian matrix possesses the property of being real symmetric positive semidefinite. With this property, the normalized Laplacian matrix can be factored as $L = U \Lambda U^T$, where $U = [u_0, u_1, \ldots, u_{N_v-1}] \in \mathbb{R}^{N_v \times N_v}$ is the matrix of eigenvectors ordered by eigenvalues, and $\Lambda$ is the diagonal matrix of eigenvalues with $\Lambda_{ii} = \lambda_i$. The eigenvectors of the normalized Laplacian matrix form an orthonormal space, i.e., $U^T U = I$.

**Graph Fourier Transform.** Denote the feature vector of a node as $x \in \mathbb{R}^{N_v}$. The graph Fourier transform to $x$ is defined as

$$\mathcal{F}(x) = U^T x.$$  \hspace{1cm} (4.3)

The inverse graph Fourier transform is defined as

$$\mathcal{F}^{-1}(\tilde{x}) = U^T \tilde{x},$$ \hspace{1cm} (4.4)

where $\tilde{x}$ represents the resulting feature from graph Fourier transform.

By the definition of eq. (4.3), the graph Fourier transform projects the input node feature to the orthonormal space where the basis is formed by eigenvectors of the normalized graph Laplacian. Elements of the transformed feature $\tilde{x}$ are the coordinates in the new space so that the input feature can be represented as $x = \sum_i \tilde{x}_i u_i$, which is exactly the inverse graph Fourier transform defined in eq. (4.4).
The graph convolution of a kernel $g \in \mathbb{R}^{N_o}$ over a node feature $x$ is defined as

$$x \ast_G g = \mathcal{F}^{-1}(\mathcal{F}(x) \odot \mathcal{F}(g)) = U(U^T x \odot U^T g) \quad (4.5)$$

where $\odot$ denotes the Hadamard product, and $\ast_G$ denotes the convolutional operator on graph $G$. We reformulate the graph convolution by representing the kernel with a more elegant representation $g_\theta = diag(U^T g)$:

$$x \ast_G g_\theta = Ug_\theta U^Tx. \quad (4.6)$$

This is a typical formulation of spectral-based GCN, and the major difference among different variants lies in the design of the kernel $g_\theta$. Some researchers [37] follow the fashion of CNN which treats $g_\theta$ as a set of learnable parameters.

There are two limitations associated with the formulation in eq. (4.6): (i) the eigen decomposition of the Laplacian matrix is required for each forward pass, which is poor in scalability for large graphs; (ii) the parameters of kernel $g_\theta$ depend on the eigen basis indicated by $U$, so the generalization of the learned parameters is poor when the structure of an input graph is unseen at the training stage.

Researchers strive for addressing the above two limitations along two directions: (i) follow the spectral stream, and avoid the computation of eigen decomposition via the reformulation of eq. (4.6) based on Chebyshev polynomials; (ii) abandon the spectral stream but follow the spatial stream (introduced in section 4.3.2).

**Chebyshev Spectral GCN** was proposed by Defferrard et al [38] which defines a kernel as Chebyshev polynomials of the diagonal matrix of eigenvalues:

$$g_\theta = \sum_{n=0}^{N_o-1} \theta_n T_n(\tilde{\Lambda}), \quad (4.7)$$

where $\tilde{\Lambda} = 2\Lambda/\lambda_{\text{max}} - I_{N_o}$. The Chebyshev polynomials are defined recursively by $T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$ with $T_0(x) = 0$ and $T_1(x) = x$. The graph convolution
in eq. (4.6) can be thus reformulated by replacing $g_\theta$ with the Chebyshev polynomials:

$$
\mathbf{x} \ast_G g_\theta = U \left[ \sum_{n} \theta_n T_n(\tilde{\Lambda}) \right] U^T \mathbf{x} \approx \sum_{n} \theta_n T_n(\tilde{L}) \mathbf{x},
$$

(4.8)

where $\tilde{L} = 2L/\lambda_{\max} - I_{N_o}$. By the definition of eq. (4.8), there is no need to calculate the eigen decomposition, which reduces the computation complexity significantly. Since $T_n(\tilde{\Lambda})$ is a polynomial of $\tilde{\Lambda}$ of the $n$-th order, the convolution $\theta_n T_n(\tilde{L}) \mathbf{x}$ is performed locally on each node, and the order $n$ determines the inter-node hops. Due to such a local property, the Chebyshev Spectral GCN in essence bridges the gap between the spectral-based GCN and the spatial-based one.

### 4.3.2 Spatial-based GCN

Similar to the Chebyshev spectral GCN, the spatial-based GCN also aggregates the local contextual information. The only difference is that the local connectivity in Chebyshev spectral GCN is captured by the polynomials of the Laplacian matrix, while the local connectivity in the spatial-based GCN is captured by the adjacency matrix directly.

In order to draw an analogy between GCN and CNN, we follow the notations of the regular convolution presented in eq. (4.1) to introduce the spatial-based graph convolution. Therefore, we introduce two major components of graph convolution, i.e., the sampling function and weight function.

#### 4.3.2.1 Sampling Function

On the regular grids (e.g., images), the sampling function $\mathbf{p}(h, w)$ is defined on the neighboring pixels with respect to the center location $\mathbf{x}$. On irregular graphs, we can similarly define the sampling function on the neighbor set $\mathcal{N}(v_i) = \{ v_j | d(v_j, v_i) \leq \eta \}$ of a node $v_i$. $d(v_j, v_i)$ denotes the minimum length of any path from $v_j$ to $v_i$. Thus, the sampling function $\mathbf{p} : \mathcal{N}(v_i) \rightarrow V$ is written as

$$
\mathbf{p}(v_i, v_j) = v_j.
$$

(4.9)
4.3.2.2 Weight Function

In the regular 2D convolution, a regular grid exists around the center location. So pixels within the receptive field follow a fixed spatial order. However, there is no such an implicit spatial arrangement. A solution to such a problem has been addressed in [39, 40], i.e., assigning a label to each node within the neighbor set of a root node according to a partitioning strategy. The partition is implemented as a mapping function \( \Psi_i : \mathcal{N}(v_i) \to \{0, \ldots, N_p - 1\} \) which maps a neighboring node to a subset label. The weight function \( w(v_i, v_j) : \mathcal{N}(v_i) \to \mathbb{R}^c \) can be implemented by indexing a tensor of \((c, N_p)\) dimension:

\[
w(v_i, v_j) = w'(\Psi_i(v_j)).
\]  

(4.10)

4.3.2.3 Overall Formulation

With the sampling function and weight function defined, we write the formulation of spatial-based GCN as

\[
f_{\text{out}}(v_i) = \sum_{v_j \in \mathcal{N}(v_i)} \frac{1}{Z_i(v_j)} f_{\text{in}}(p(v_i, v_j)) \cdot w(v_i, v_j),
\]

(4.11)

where the normalizing term \( Z_i \) is used to balance the contributions of different partition subsets, which is defined as the cardinality of the corresponding partition subset, i.e., \( Z_i(v_j) = |\{v_k | \Psi_i(v_k) = \Psi_i(v_j)\}| \).

By substituting eq. (4.10) and eq. (4.10) into eq. (4.11), the overall formulation can be rewritten as

\[
f_{\text{out}}(v_i) = \sum_{v_j \in \mathcal{N}(v_i)} \frac{1}{Z_i(v_j)} f_{\text{in}}(v_j) \cdot w'(\Psi_i(v_j)).
\]

(4.12)

Note that the formulation defined in eq. (4.12) is so general that can cover the formulation of the regular CNN defined in eq. (4.1) given the suitable sampling and weight function.
Conventionally, the spatial-based GCN is implemented as follows

\[ f_{out} = \sum_{n}^{N_p} W_n f_{in}(\tilde{D}_n^{-\frac{1}{2}} A_n \tilde{D}_n^{-\frac{1}{2}}). \]  \hspace{1cm} (4.13)

\( f_{in} \) and \( f_{out} \) denote the feature map with the shape of \( C_{in} \times N_o \) and \( C_{out} \times N_o \), respectively, where \( C_{in} \) is the number of input channels, \( C_{out} \) is the number of output channels, and \( N_o \) is the number of nodes. \( A_n \) represents an \( N_o \times N_o \) adjacency matrix, whose entry \( A_{j,i}^n \) indicates whether a node \( v_j \) is in the subset of node \( v_i \). \( \tilde{D}_n \) is a normalized diagonal matrix with \( \tilde{D}_{i,i}^n = \sum_j (A_{j,k}^n) + \epsilon \), where \( \epsilon \) is a value added for numerical stability to avoid the empty rows in \( A_n \). \( W_n \) is a \( 1 \times 1 \) convolutional kernel of the shape \( C_{out} \times C_{in} \times 1 \times 1 \).

For notion simplicity, we further rewrite the formulation in eq. (4.13) as

\[ f_{out} = \sum_{n}^{N_p} W_n f_{in} A_n. \]  \hspace{1cm} (4.14)

### 4.3.3 Non-local Connectivity

The topology of the graph is set by hand and fixed for all layers in spatial-based GCN, which may not be optimal. Therefore, Shi et al.[41] proposed to adapt the topology through the attention estimation and learning. Specifically, they add two more adaptive topology terms to eq. (4.14):

\[ f_{out} = \sum_{n}^{N_p} W_n f_{in}(A_n + B_n + C_n), \]  \hspace{1cm} (4.15)

where \( B_n \) represents the learned connection which is shared by all input graphs. \( C_n \) is an attentively estimated connection which is adaptive to different input graphs, which is defined as

\[ C_n = \text{softmax}(f_{in}^T W_{\theta_n}^T W_{\varphi_n} f_{in}). \]  \hspace{1cm} (4.16)

where \( W_{\theta_n} \) and \( W_{\varphi_n} \) represent the linear layers for mapping the representations of different nodes into the same space.
4.3.4 Temporal Modeling

The temporal modeling is a critical component for sequence recognition. The most common approach for temporal modeling of GCN is to aggregate the corresponding nodes across frames. This can be implemented as a typical 1D convolution. The spatial and temporal conv layers are stacked alternately in GCN for sequence recognition. Considering the temporal modeling, the shapes of $f_{in}$ and $f_{out}$ in eq. (4.14) are $C_{in} \times T \times N_o$ and $C_{out} \times T \times N_o$, respectively.
CHAPTER 5
Temporal Modeling with RNN

5.1 Background

Five years ago, researchers gradually started exploring to use RNN for temporal modeling. Within the recurrent framework, researchers design the network architecture following four directions:

**Feature Discriminability.** Several works [42, 43, 44, 45] designed a network structure following the style of conventional works that took the inspiration of the activity spatial structural properties. Another set of works [46, 47] directly sought statistically representative features from the activity sequences using attention mechanisms, gating, *etc.*

**Integration of Spatial and Temporal Information.** Song *et al* [47] incorporated the LSTM with both the spatial and temporal attention mechanisms to adaptively select the discriminative dimensions and time steps for classification. Wang *et al* [48] formulated the activity sequences as two sequences along the spatial and temporal axes, respectively, and used a two-stream RNN to model these two sequences. A common drawback of these work is that they assume independence between the structural and temporal information in the activity sequences.

**Sequential Orders.** Activity instances satisfy the temporal causality, *i.e.*, the observation at the current time step depends on those of the previous time steps. However, statistical dependency also exists in reverse of the time arrow. Thus, Du *et al* [42] used a bidirectional RNN structure to model activity sequences. When the components of each observation at a time step can be represented by a tree structure, by traversing the tree structure bidirectionally using the depth-first search, an activity sequence can also be represented as a sequence spatially. Liu *et al* [44, 46] developed the spatiotemporal RNN for modeling such a spatial sequential order as an addition to the temporal sequential order.
Sequential Dependency Ranges. The range of temporal dependency is an important factor in modeling activity sequences. This is commonly modeled using RNN with long-short term memory (LSTM) [6], in which the range is determined by the cooperation of the memory cell and several modulative gates. However, the capability of such an adaptive modeling would be overstretched given the high complexity of the activity sequences. To this end, several works explored to model the dependency ranges in a more controllable fashion. For example, Lee et al [49] incorporated the LSTM with multi-scale temporal sliding windows, so as to explicitly control the dependency ranges.

The RNN-based approaches achieved significant performance improvement compared to the conventional approaches, thanks to better long-term temporal modeling. However, there is still a lot of improvement space for both spatial and temporal perspectives.

Although much progress has been achieved, these algorithms still have two challenges. The first one is the discriminative challenge. The space of human body movements where activities are performed can be categorized by locating the limbs with possibly high dynamics: upper limbs (e.g., punching), lower limbs (e.g., kicking), or both (e.g., jogging). Subtle inter-class variations require a finer level of discrimination of the recognition methods. For example, the sign language recognition [50] is considered to be more challenging than general-purpose activity recognition. The discriminative challenge becomes severe as more classes are packed into the limited degree of freedom of the 3D joint coordinates, in which decision boundaries between classes are hard to determine. Another challenge in skeleton-based activity recognition is the adaptability to handle incrementally introduced activity classes that are unknown in training. Most existing methods require re-training of entire model or sophisticated hyper-parameter adjustment, which hinders real-world applications. Some existing works [21, 51] handle new classes using non-parametric models. However, they are not scalable against large amount of incremental training data.

With the growing need of large-scale and fine-grained activity recognition, we hypothesize that the above challenges are not resolvable via a single network, and sophistication must be expanded at the structure of representation. A single RNN with
a flat N-way classifiers sharing fully connected layers is not sufficient for large-scale (> 100 activity classes) recognition. The following barriers prevent the naive combination of multiple RNNs such as a RNN-based decision tree, as the structure is much larger to maintain and learn: (i) How to learn the structure of RNNs automatically (instead of manual specification)? (ii) How to leverage the commonalities among the RNNs to achieve effective co-training? (iii) How to reduce overall training time?

In this chapter, we present an adaptive learning framework for large-scale skeleton-
based human activity recognition that aggregates multiple discriminative RNN models hierarchically. Specifically, we organize activity classes into activity categories (clusters of activity classes), which can be divided into fine-grained sub-categories. All activity categories form a tree structure that we termed as activity category hierarchy (ACH). Each activity category is associated with a RNN model. The ensemble of them form the RNN Tree (RNN-T), as shown in fig. 5.1. RNN-T and ACH are jointly learned from training data using a level-by-level approach. Individual RNNs are co-trained, such that each RNN down the tree can be more effectively optimized in discriminating finer-grained activities in that category. At run time, RNN-T recognizes activities via a hierarchical inference process, during which individual RNNs differentiate activity classes with increasing confidence, and ambiguous decisions are deferred to sub-trees of RNNs where fine-grained activities can be effectively differentiated. Inference is completed when the activity is recognized with high confidence, or a leaf node of RNN-T is reached. We further develop an incremental learning algorithm such that RNN-T is adaptive to new activities classes. Specifically, new activity classes are inserted into existing activity categories in ACH, where the respective RNN sub-trees can be either incrementally updated (for categories with minor changes) or re-built when necessary (for categories with drastic changes).

To assist the research community of skeleton based human activity recognition, we create a large-scale skeleton-based activity recognition dataset with 140 activity classes by aggregating and processing 10 existing smaller-scale datasets [13, 14, 21, 28, 50, 52, 53, 54, 55, 56], which we term as 3D-Skeleton-Activity-140 dataset. Our dataset is significantly larger than most of the existing dataset, for instance the HDM05 dataset [54] including 65 activity classes is currently the largest dataset for the same purpose. We perform experiments on 3D-Skeleton-Activity-140 to e-

valuation the RNN-T based methods and compare it with state-of-the-art methods. Our experimental results demonstrate that RNN-T performs favorable against the state-of-the-art methods.

The main contributions of this work are four-folds. (i) We propose a novel, adaptive and hierarchical framework for fine-grained, large-scale skeleton-based activity recognition method. Multiple RNNs are incorporated effectively in a tree-like
Figure 5.2: (a) A RNN architecture for activity recognition with 400 LSTM units, whose structure is shown in (b). 8 learnable parameters of a LSTM block is shown in (b), i.e., $W_{xc}$, $W_{hc}$, $W_{xi}$, $W_{hi}$, $W_{xf}$, $W_{hf}$, $W_{xo}$, and $W_{ho}$.

hierarchy to mitigate the discriminative challenge by ambiguity-aware deferral and divide-and-conquer. (ii) We develop an effective learning method to build ACH and RNN-T jointly, so as to achieve high recognition accuracy and running efficiency. (iii) We design an incremental learning algorithm for RNN-T to be adapted to new classes, which significantly reduces the re-training time. (iv) We create 3D-Skeleton-Activity-140, a large-scale skeleton-based activity recognition dataset with the largest number of classes to-date, and produce a benchmark to evaluate RNN-T against state-of-the-art methods.

5.2 Overview

RNN [6] is a class of neural network whose neurons send feedback signals to each other. fig. 5.2 (a) shows a segment of unrolled RNN from time step 1 to $T$. Along the time axis at each step $t$, RNN accepts the current input $x_t$ together with the previous hidden state, and updates the current state via a set of non-linear activation functions. RNN is suitable for handling sequences of data because it can represent complex relations in the sequence, and is robust with local distortions.

One limitation of the original RNN model is that it cannot model long-term dependencies for sequential data because of vanishing and exploding gradients [6]. This problem is alleviated with the long short-term memory (LSTM) cells [57] as
a replacement of the traditional nonlinear units. A LSTM block contains a self-connected memory cell to store information across long duration, and also contains 3 modulative gates (i.e., input, forget, and output gates) to adaptively adjust the forgetting rates of the old state, see fig. 5.2 (b). When RNN model is applied to pose-based human activity recognition problem, it usually serves as multi-nominal classification and predicts a single activity class label at the end of the sequence.

Algorithm 1 Learning of ACH and RNN-T

Input: \( C_1 \) including all \( N \) activity classes
Output: ACH, RNN-T
1: Initialize ACH by embedding \( C_1 \), and marking \( C_1 \) as unvisited
2: Train \( \mathbf{R}_1 \) for \( C_1 \)
3: Initialize \( B_1 = (1, \ldots, 1) \) with list size = \( N \)
4: while \( \exists \) \( C_i \) is unvisited and \( |C_i| > \theta_l \) do
5:   • Generate candidate divisions for \( C_i \) (section 5.3.1.1)
6:   • Pre-train RNNs for each candidate division (section 5.3.1.2)
7:   • Evaluate candidate divisions (section 5.3.1.3)
8:   • Expand ACH and RNN-T based on the optimal division
9:   • Fine tune the newly-added RNNs in RNN-T jointly (section 5.3.1.4)
10: Mark \( C_i \) as visited
11: end while

5.3 ACH and RNN-T for Activity Recognition

We start with notations that will be used throughout the chapter. A human activity dataset consists of labeled activity instances \( \{x_i, y_i\} \), where \( x_i \) is a sequence of 3D skeletal poses collected from a video sequence, and \( y_i \) is the corresponding label out of a total of \( N \) activity classes. An activity category \( C \) is defined as a set of activity classes that share similar characteristics. Activity categories are organized in the activity category hierarchy (ACH), which is constructed for large-scale activity recognition following a hierarchical divide-and-conquer strategy. Specifically, the root category (level \( h = 1 \)) initially contains all activity classes. It is further divided into activity categories corresponding to finer granularity to form a tree structure. Let \( C_{i,h} \) be the \( i \)-th activity category at the \( h \)-th level of ACH \( (h \geq 1) \). \( \forall \) node \( C_{j,h+1} \), \( \exists \) parent node \( C_{i,h} \) s.t. \( C_{j,h+1} \subseteq C_{i,h} \). To simplify the notation, we omit the level index \( h \) of an activity category, i.e., \( C_{i,h} \rightarrow C_i \), and define the children of \( C_i \) in ACH as \( \{C^3\} \). In
ACH, the ambiguity of classes in category $C_i$ is encoded in a bit set $B_i = \{b_s | c_s \in C_i\}$, where $b_s = 0$ means class $c_s$ is ambiguous (whose recognition decision will be deferred to subtrees), and $b_s = 1$ means $c_s$ is a recognized class.

RNN-T is an ensemble of RNN models that mirror the structure of ACH (see fig. 5.1 (b)), where each RNN is associated with one activity category in ACH. We denote the RNN associated with category $C_i$ as $\mathcal{R}_i$. RNN-T/ACH resembles a Decision Tree (DT) with RNN as base classifiers. The base classifiers in a DT are usually learned separately, but the RNN modules in RNN-T/ACH are co-trained following the ACH tree structure. In addition, activity classes contained in different activity categories can overlap (e.g., both $C_2$ and $C_3$ in fig. 5.1 (c) contain activity class 06). Therefore, unlike in a DT, where a classification error is irrecoverable once a branch is reached, RNN-T/ACH allows multiple sub-trees to output the same class label (with small overhead), which improves the robustness of the final classification.

5.3.1 Learning of ACH and RNN-T

The ACH and RNN-T are learned from the labeled data jointly, as they share the same tree structure. Taking advantage of the tree structure, learning of ACH and RNN-T are orchestrated in a level-by-level scheme, which is presented in algorithm 1. Specifically, starting with the root activity category at the first level of ACH, which includes all activity classes to be considered, activity categories at the current level are successively divided into finer activity categories for the next level, with a RNN trained for each newly generated activity category.

The division of an activity category is performed as follows. First, multiple candidate divisions of the ambiguous activity classes (whose label is not yet confidently determined and needs further inference in the hierarchy) in the current activity category are obtained by repeatedly running a spectral clustering algorithm [58] (section 5.3.1.1). For each candidate division, a set of RNNs are pre-trained independently (section 5.3.1.2). Then, one optimal candidate division based on a performance evaluation metric is chosen as the next-level activity categories to be included in the ACH (section 5.3.1.3). As the last step, RNNs corresponding to the newly generated activity categories are fine tuned jointly (section 5.3.1.4). After one level is complete,
the same process is repeated for the next level. The process completes when all activity classes can be classified by RNNs in RNN-T with high confidence, or the resulting activity category is too small to divide. As a result, leaf nodes of ACH should contain only recognizable fine-grained activity classes. We provide details of these four steps in the following.

5.3.1.1 Generate Candidate Divisions

For each activity category, if it contains any ambiguous activity class, it needs to be further segmented to activity categories in the next level. Specifically, if we are to process activity category $C_i$, we first identify ambiguous classes within $C_i$ using its corresponding RNN model $\mathfrak{R}e_i$. For each class $c_s \in C_i$, we compute the F-scores for training ($F^t_s$) and validation ($F^v_s$) respectively, based on the recognition results of $\mathfrak{R}e_i$ on the training and validation datasets. If $\min(F^t_s, F^v_s) < \theta_c$ ($\theta_c$ is a pre-determined threshold), $c_s$ is regarded as an ambiguous class. The calculated ambiguity terms in $C_i$ are stored in the corresponding bit set $B_i$.

We denote the set of all ambiguous classes in $C_i$ as $C_i^a$. If $|C_i^a| \leq \theta_t$, $C_i$ is not be further divided, where $\theta_t$ is the pre-determined minimum activity category size. Otherwise, we generate $n = \lceil N/\theta_t \rceil$ different divisions of $C_i^a$ in two steps. First, we calculate the confusion matrix based on recognition results of RNN $\mathfrak{R}e_i$ on the validation dataset. Secondly, we perform spectral clustering \cite{05} for $n$ rounds on the confusion matrix. At the $k$-th round, we set the desired number of clusters as $k$. As a result, a candidate division containing $k$ disjoint clusters (sub-categories) $\{C_d^j\}_{j=1}^k$ is generated with $C_d^j \subset C_i^a$, where the subscript $d$ denotes disjoint sub-categories. However in practice, spectral clustering occasionally generates fewer clusters than the desired amount. In this case we discard these candidate divisions. From our experience, the number of remaining candidate divisions $n'$ is usually less than 8.

The above disjoint division scheme forbids error recovery when misclassification occurs during divide-and-conquer. As we mentioned previously, ACH allows activity categories to overlap to improve classification robustness. To this end, we purposely allow highly ambiguous class(es) to be treated in more than one categories according to a misclassification likelihood $p^j(s)$ as defined below, so that these classes can be
resolved across more subtrees accordingly.

\[ p^j(s) = \frac{\sum_{x \in X_s, c_s \notin C^j_d} 1(y' \in C^j_d)}{|X_s|}, \]  

(5.1)

where the numerator is the number of activity instances in \( c_s \) that are misclassified into \( C^j_d \), and \( 1(\cdot) \) is an indicator function. We denote \( X_s \) the set of activity instances in class \( c_s \). For each candidate division \( \{C^j_d\}_{k=1}^k \) and \( c_s (\notin C^j_d) \), \( p^j(s) > \theta_o \) (\( \theta_o \) is a pre-determined threshold) suggests that \( c_s \) is very likely to be misclassified by RNNs \( \Re_c \) with other classes in \( C^j_d \). Specifically, given a predicated label \( y' \in C^j_d \) and the ground truth class \( c_s \notin C^j_d \), deferring \( y' \) to category \( C^j_d \) for finer classification is known to be wrong. To avoid hard division and improve fault tolerance, for each candidate division \( C^j_d \) with \( p^j(s) > \theta_o \), we add such highly ambiguous class to \( C^j_d \) to form \( C^j \), i.e., \( C^j = \{C^j_d, c_s\} \) for all possible \( c_s \). As a result, each candidate division \( \{C^j_d\}_{k=1}^k \) is augmented as \( \{C^j\}_{j=1}^k \) with \( C^j \subseteq C^j \).

5.3.1.2 Pre-training of RNNs

To maximize the fitting capability of RNN-T, modular RNNs should be trained jointly. Whereas the joint training is time-consuming, and for each candidate division, we need to train a set of RNNs in order to evaluate their respective effect on the recognition performance, so the total training time increases quadratically in the number of candidate divisions. Thus, we decompose the training process into two steps to ensure the training efficiency. First, in this subsection, we present the pre-training of new RNNs independently for the evaluation process in the next step (section 5.3.1.3). After identifying the optimal division, we fine-tune the newly-added RNNs jointly (section 5.3.1.4) to enhance their classification performance.

For each candidate division \( \{C^j\}_{j=1}^k \), we train a set of RNNs \( \{\Re^j\}_{j=1}^k \). To train \( \Re^j \), first, we initialize it by copying weights from its parent node \( \Re_c \) except those on the output layer, and initialize the output layer using random values. Then, we train \( \Re^j \) independently by minimizing the maximum-likelihood loss function using...
the stochastic gradient descent (SGD):

\[ L_{\Phi_j}(W) = -\sum_{r=1}^{\mid \Phi_j \mid} \ln p(y_r|x_r), \quad (5.2) \]

where \( W \) represent the learnable weights of \( \mathcal{R}e^j \). The types of weights vary for different RNN models, we refer readers to fig. 5.2 or [6] to see the learnable weights of a typical RNN with LSTM as an example. \( \Phi_j \) denotes the training set for \( \mathcal{R}e^j \), and \( x_r \) is the \( r \)th activity instance in \( \Phi_j \). \( y_r \) is the ground truth label of \( x_r \). \( p(y_r|x_r) \) is output by the softmax function of \( \mathcal{R}e^j \), representing the probability of \( x_r \) being labeled as \( y_r \). The back-propagation through time (BPTT) [6] is used to compute the derivatives of (5.2) w.r.t. \( W \).

### 5.3.1.3 Evaluate Candidate Divisions

We then use the individually trained RNNs to choose an optimal division for category \( C_i \). To this end, we first build a two-level category sub-hierarchy \( \text{CH}_k \), with \( C_i \) as the root at the first level and the \( k \)-th candidate division \( \{C^j\}_{j=1}^k \) at the second level as the children (sub-categories) of \( C_i \). RNN \( \mathcal{R}e_i \) corresponding to \( C_i \), and RNNs \( \{\mathcal{R}e^j\}_{j=1}^k \) corresponding to the \( j \)-th candidate division are organized in a RNN sub-tree (RNN-ST\(_k\)) with the same structure as \( \text{CH}_k \). Within \( \text{CH}_k \), an ambiguous class \( c_s \) (\( \in C_i \)) may belong to multiple sub-categories \( \{C^j\} \) at the second level, so it causes confusion during run time, when we need to defer \( c_s \) to a specific sub-category \( C^j \). Thus, we refer to a lookup table to resolve such a confusion. For candidate division \( \{C^j\}_{j=1}^k \), the lookup table \( f_{i,k} \) of \( C_i \) is built upon \( \{C^j\}_{j=1}^k \) and its corresponding disjoint division \( \{C^j_d\}_{j=1}^k \) (see section 5.3.1.1), with \( C^j_d \subseteq C^j \). Specifically, given a predicted label \( y' \), if \( y' \in C^j_d \), \( f_{i,k}(y') = C^j \).

Next, we introduce a metric \( \mathcal{R} \) to evaluate the optimality of each candidate division, which is inspired by the splitting of nodes in a decision tree [59] and defined as:

\[
\mathcal{R} = \mathcal{A}^t + \mathcal{A}^v + \min \left( \frac{\mathcal{A}^t}{\mathcal{A}^v}, \frac{\mathcal{A}^u}{\mathcal{A}^t} \right) - \lambda \exp \left( \frac{H}{N} \log N_l \right), \quad (5.3)\]

where \( \mathcal{A}^t \) and \( \mathcal{A}^v \) represent the training and validation classification accuracy, respec-
tively, and \( N_l \) is the number of leaf nodes, \( H \) is tree depth, and \( N \) is the number of all classes. For each candidate division, \( A^t \) and \( A^v \) is computed by feeding the training and validation data back to the RNN sub-tree RNN-ST. \( \min(A^t, A^v) \) represents the stability between \( A^t \) and \( A^v \), which ensures that RNN-T will not yield recognition accuracy with large variations when applied to training dataset and validation dataset. The inefficiency term penalizes trees with complex structures, i.e., deep with few leaves. Parameter \( \lambda \) balances accuracy with inefficiency. Note that the usage of the stability and inefficiency term potentially reduces the risk for us to select an over-fitted tree structure of ACH/RNN-T.

We compute a score \( R_k \) for each candidate division, and also compute \( R_0 \) assuming no further division. We select division based on \( m = \arg \max \_k R_k \). If \( m = 0 \), we do not divide the current category \( C_i \); otherwise, we divide \( C_i \) into finer categories given by the \( m \)-th division at the next level. Correspondingly, RNNs \( \{\mathcal{R}^j\}_{j=1}^m \) are organized into RNN-T and the cross level lookup table \( f_i \) is set accordingly.

### 5.3.1.4 Joint Fine-tuning of RNNs

This step fine tunes the newly-added modular RNNs \( \{\mathcal{R}^j\}_{j=1}^m \) and their parent RNN \( \mathcal{R}_i \) jointly, so as to achieve higher classification accuracy. Such fine-tuning is achieved by adding a new term in the RNN objective to reduce the risk of deferring the ambiguous predicted label to a wrong activity category via the lookup table \( f_i \). Thus, we have:

\[
L_\Phi(W) = -\sum_{r=1}^{\Phi} \ln \{1(f_i(y_r') = \emptyset) p(y_r|x_r) \\
+ \sum_{j=1}^{m} [1(f_i(y_r') = C^j) \sum_{s=1}^{\left| C^j \right|} 1(c_s = y_r)p_j(c_s|x_r)] \}
+ \frac{\alpha}{\left| \Phi \right|} \sum_{r=1}^{\left| \Phi \right|} 1(f_i(y_r') \neq f_i(y_r)), \tag{5.4}
\]

where \( W \) represent the learnable weights of \( \{\mathcal{R}^j\}_{j=1}^m \) and \( \mathcal{R}_i \). \( \Phi \) is the training set that corresponds to \( C_i \). \( y_r \) is the groundtruth class label of \( x_r \). \( y_r' \) is the label of \( x_r \) that is predicted by \( \mathcal{R}_i \). The children of \( C_i \) are denoted as \( \{C^j\}_{j=1}^m \) that corresponds
Figure 5.3: Incremental learning procedures: (a) inserting new classes to activity categories with similar classes as the new ones, and (b) updating ACH and RNN-T. Red circles represent the new classes. Better viewed in color and by zooming in.

5.3.2 Recognizing Human Activities

RNN-T/ACH recognizes a skeleton based human activity in a hierarchical manner. For an input skeleton sequence \( \mathbf{x} \), the process starts at the root level, where the root RNN \( \mathcal{R}_1 \) (corresponding to \( C_1 \)) generates its classification result \( y_1' \). Given that \( y_1' \) refers to class \( c_s \) (\( \in C_1 \)), if \( b_s = 1 \) where \( b_s \in B_1 \), \( y_1' \) is output directly as and the recognition process is completed. Otherwise, \( y_1' \) is deferred via the lookup table \( f_1 \) to a specific child \( C_j \) of \( C_1 \) for finer classification using \( \mathcal{R}_j \). This process continues until \( \mathbf{x} \) is recognized, or a leaf node of RNN-T produces a classification result.

5.4 Incremental Learning

When ACH/RNN-T encounters newly-defined activity classes that are not presented in previous training data, we augment it to include these new classes with an incremental learning algorithm. We exploit the transferable information of ACH and RNN-T to improve the effectiveness of the incremental learning even in the presence of limited training data. The transferable information of ACH is the inter-category relations encoded by the lookup tables (see section 5.3.1.3). The transferrable infor-
mation of RNN-T is the similar network structure (see section 5.3.1.2) shared by the modular RNNs. ACH and RNN-T are updated level by level from the bottom up with the following two procedures: (i) inserting new classes to the activity categories with similar activities, and (ii) updating ACH and RNN-T to reflect the change in structure. fig. 5.3 shows the changes of ACH after each procedure.

**Inserting new classes.** New classes are manually labeled. All new classes are first put into the root level activity category. We then traverse the tree structure of ACH and RNN-T to find appropriate activity categories for the new classes. Specifically, when the process reaches an activity category $C_i$, we estimate the likelihood that an activity instance of new class $c_s$ is classified as the member classes of each child $C^j$ of $C_i$, using their corresponding modular RNN in the RNN-T. Such a likelihood is defined as:

$$p^j(s) = \frac{\sum_{x \in X_s, c_s \in C_i} 1(y' \in C^j)}{|X_s|},$$

(5.5)

where the numerator counts how many activity instances of $c_s$ are classified as the member classes of $C^j$. $X_s$ represents the activity instance set of $c_s$. If $p^j(s) > \theta_o$, which is a threshold defined in section 5.3.1.1, $c_s$ is forwarded to and inserted into that activity category. As such, possibly multiple children of $C_i$ will process $c_s$ in parallel subsequently. The process continues until a leaf node is reached.

**Updating ACH and RNN-T.** After the new activity classes are inserted into the ACH, activity categories $\{C_i\}$, bit sets $\{B_i\}$, lookup tables $\{f_i\}$ and the modular RNNs $\{\mathcal{R}e_i\}$ in RNN-T are then updated in a similar level by level fashion. We traverse the tree structure of ACH and RNN-T. When the process reaches an activity category $C_i$, we update $\mathcal{R}e_i$ to recognize new classes in $C_i$. Then, we use the updated $\mathcal{R}e_i$ to identify the ambiguity of these new classes as in section 5.3.1.1 and update $\{B_i\}$ accordingly. If there exist any new classes in $C_i$ is identified as recognized, we remove such recognized classes from the offsprings of $C_i$ in ACH. Finally, we determine how to update the lookup table $f_i$, following the principle "incrementally updating for minor changes, and re-building for drastic changes". We measure the degree of change occurring in $C_i$ using the increment ratio of ambiguous classes in it, which is defined as $\tau = \frac{n_{new}}{n_{old}}$. $n_{new}$ and $n_{old}$ represent the number of new ambiguous classes.

\footnote{If $n_{new} > 0$ and $n_{old} = 0$, it means a drastic change. If $n_{new} = 0$ and $n_{old} = 0$, we set $\tau$ as 0.}
and old ambiguous classes, respectively. We use $\theta_r = h \exp(1 - h)$ as a threshold for $\tau$, with $h$ indicating the depth of the level that $C_i$ resides. $\tau < \theta_r$ means that the change in $C_i$ is minor, so we just update $f_i$ by enabling the deferral for each new ambiguous class $c_s$ to a specific child $C^j$ with the highest likelihood score $p^j(s)$ defined in (5.5). $\tau \geq \theta_r$ means the occurrence of drastic change in $C_i$. Then, we will rebuild the sub-tree structure of both ACH and RNN-T starting from $C_i$ as in section 5.3.1.

5.5 Experiments

We evaluate RNN-T in two test scenarios: (i) with a fixed number of activity classes (section 5.5.1), and (ii) with increasing number of classes over time (section 5.5.2). For scenario (i), we use the classification accuracy as the metric. Test scenario (ii) is used to evaluate our incremental learning algorithm, so we use both accuracy and the retraining time as metrics. All reported results are based on the implementation using a single CPU core (3.4GHz) on an Intel Xeon E5-2687W v2 machine with 128GB RAM. The four parameters $\lambda$, $\theta_c$, $\theta_o$, $\theta_l$ described in section 5.3.1 are chosen as the following. We use $\lambda$ to balance between the recognition accuracy and inefficiency on the criterion function Eq.(5.3). Since inefficiency grows exponentially with ACH levels, we set $\lambda$ to be a small value, i.e., $\lambda = 0.03$. We use $\theta_c$ as a threshold to determine if a class with low F-scores is ambiguous, thus we prefer $\theta_c$ to be a large value within $[0, 1]$; $\theta_c$ is empirically set to be 0.85. $\theta_o$ and $\theta_l$ affect the class categorization quality and the values are case dependent for each dataset. We choose the best values empirically based on the resulting accuracy.

Dataset. Because existing skeletal activity datasets do not contain enough activity classes (i.e., more than 100), we create a new dataset with 140 classes by aggregating all distinct classes from the following 10 existing datasets, which we name 3D-Skeleton-Activity-140: CMU Mocap [52] (23), ChaLearn Italian [50] (20), MSRC-12 Gesture [28] (12), MSR Activity3D [53] (20), HDM05 [54] (65), Kintense [21] (10), Berkeley MHAD [55] (12), MSR Daily Activity 3D [13] (13), UTKinect-Activity [56] (10), and ORGBD [14] (7), where the number of classes are shown next to each dataset. See supplemental material for a complete class list.

As these datasets have different attributes, we standardize them in 3D-Skeleton-
Activity-140, such that the number of sequences per class is 28 on average, and the frame rate is normalized to 20 FPS, and the human skeleton is represented by 20 skeletal joints (see supplemental material for details). We partition 60% of the 3D-Skeleton-Activity-140 as the training set, 20% as the validation set, and the remaining 20% as the test set. 3D-Skeleton-Activity-140 is a challenging benchmark due to two factors: (i) A large variety of movements and dynamics in various contexts are included, where fine-grained recognition is required. (ii) Video length for individual activities are in wide varieties (ranging from 5 to 800 frames) within or across classes.

RNN Modules and Feature Extraction. We derive 5 variants of RNN-T’s which differ in the RNN modules they are based on: uni/bi-directional vanilla RNN (URN-N/BRNN), uni/bi-directional RNN with LSTM (URNN-L/BRNN-L), and hierarchically bidirectional RNN (HBRNN-L) [42]. The code of HBRNN-L [42] is available to the public, while the other RNNs are re-implemented using RNNLIB [60]. Concerning the input feed to the RNNs, similar to [42], skeletal joints are divided into five parts (i.e., four limbs and one trunk) as the input to HBRNN-L. For the rest RNN models, we follow [61] to extract four features (position, angle, offset, pairwise joint distances) from the skeletal joints, and concatenate them to create a 310 dimensional feature vector per frame. The network architecture and other configurations of RNN are set according to [42] and [61], respectively. See supplemental material for more details.
5.5.1 Fixed Activity Classes

To evaluate RNN-T/ACH in multiple aspects, we enable/disable key features (EJR, IP, FT below) and compare results, which not only demonstrates the effectiveness of individual components of RNN-T, but also differentiates RNN-T with Decision Tree with RNN base classifiers. We further compare several variants of RNN-T with four baselines (URNN, URNN-L, BRNN, and BRNN-L) and 8 state-of-the-art methods (HBRNN-L [42], RR [10], Lie-group [9], HOD [11], CHARM [1], DBN-HMM [3], MP [12] and SSS [2]) in table 5.1. All methods based on RNN-T are denoted with suffix “-T”. All methods are adapted to 3D-Skeleton-Activity-140 by adjusting parameters using the validation set. We show that better results of RNN-T over state-of-the-art are achieved on 3D-Skeleton-Activity-140 and 10 existing datasets.

Activity category division vs. RNN decision tree. $\theta_l$ and $\theta_o$ are two parameters driving category division (see section 5.3.1). We vary $\theta_l$ within the range $\{5, 10, 15\}$ and $\theta_o$ within the range $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$, to investigate their impacts on the accuracy. By setting $\theta_o = 1$, the activity categories are potentially no longer allowed to overlap, which reduces RNN-T to a RNN based naive decision tree. The best results are obtained with $\theta_l = 5$ and $\theta_o = 0.2$ as shown in fig. 5.4. For subsequent experiments, we fixed $\theta_l = 5$ and $\theta_o = 0.2$.

Early jump-out of recognized activity classes (EJR). To study the effect of EJR in ordinary RNN-T, we build a standalone HBRNN-L-T with $\theta_c = \infty$ which essentially implements EJR by treating all classes to be ambiguous — and hence all classes need to be deferred to subtrees as discussed in section 5.3.1.1. The resulting accuracy decrease from 0.756 to 0.700, which shows that the early output of confidently recognized classes is advantageous in both the efficiency and recognition performance.

Inefficiency penalization (IP). To verify the inefficiency penalization term in Eq. (5.3) designed to prevent over-fitting, we build a standalone HBRNN-L-T without this penalization. Consequently, a 6-level, 26-category ACH is generated, which is more complex than the ordinary 4-level, 22-category one. As shown in table 5.1, accuracy drops from 0.756 to 0.697 by taking out the IP, which we interpret as over-fitting.
Table 5.1: Recognition results on 3D-Skeleton-Activity-140. See text that “-L” stands for LSTM, “-T” stands for RNN-T.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Accur.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URNN</td>
<td>0.296</td>
</tr>
<tr>
<td>URNN-L</td>
<td>0.665</td>
</tr>
<tr>
<td>BRNN</td>
<td>0.643</td>
</tr>
<tr>
<td>BRNN-L</td>
<td>0.672</td>
</tr>
<tr>
<td>RR [10]</td>
<td>0.723</td>
</tr>
<tr>
<td>HBRNN-L [42]</td>
<td>0.604</td>
</tr>
<tr>
<td>CHARM [1]</td>
<td>0.618</td>
</tr>
<tr>
<td>DBN-HMM [3]</td>
<td>0.601</td>
</tr>
<tr>
<td>Lie-group [2]</td>
<td>0.745</td>
</tr>
<tr>
<td>HOD [11]</td>
<td>0.657</td>
</tr>
<tr>
<td>MP [12]</td>
<td>0.203</td>
</tr>
<tr>
<td>SSS [2]</td>
<td>0.253</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Our Methods</th>
<th>Accur.</th>
</tr>
</thead>
<tbody>
<tr>
<td>URNN-T</td>
<td>0.539</td>
</tr>
<tr>
<td>URNN-L-T</td>
<td>0.743</td>
</tr>
<tr>
<td>BRNN-T</td>
<td>0.705</td>
</tr>
<tr>
<td>BRNN-L-T</td>
<td>0.751</td>
</tr>
<tr>
<td>HBRNN-L-T (4 levels)</td>
<td>0.756</td>
</tr>
<tr>
<td>HBRNN-L-T (3 levels)</td>
<td>0.750</td>
</tr>
<tr>
<td>HBRNN-L-T (2 levels)</td>
<td>0.735</td>
</tr>
<tr>
<td>HBRNN-L-T (1 level)</td>
<td>0.604</td>
</tr>
<tr>
<td>HBRNN-L-T w/o EJR</td>
<td>0.700</td>
</tr>
<tr>
<td>HBRNN-L-T w/o IP</td>
<td>0.697</td>
</tr>
<tr>
<td>HBRNN-L-T w/o FT</td>
<td>0.733</td>
</tr>
</tbody>
</table>

Figure 5.5: Recognition results on standalone datasets. The number of activity classes of each dataset is shown behind its name.

caused by the complex structure.

Fine-tuning (FT). The joint fine-tuning in section 5.3.1.4 is another factor that RNN-T is superior than the RNN-based decision tree (where RNNs are trained separately). As shown in table 5.1, such fine-tuning increases performance from 0.733 to 0.756, demonstrating that fine-tuning in RNN co-training is indeed beneficial.

Increasing levels of RNN-T. table 5.1 shows that as the level of RNN-T increases, accuracy increases monotonically until saturation close to 4 levels. This shows that a single RNN is not sufficient for large-scale activity recognition, and for 140 classes, a 4-level tree is a good trade-off between model sophistication and efficiency.

Comparison with baselines and state-of-arts. As shown in table 5.1, our method achieves significant performance gain over 4 baselines and 8 state-of-the-art methods. In the case of LSTM RNN as module in RNN-T, hierarchical BRNN outperforms ordinary BRNN, where the main cause is the difference in network capacity — 60
units per hidden layer for HBRNN-L versus 400 units per hidden layer for BRNN-L. Specifically, since individual RNNs in RNN-T share the same topology except for the output layer, the network capacity of BRNN-L is well enough for a large number of activity classes in a category. However it can be too large for categories with a small number of classes. As a result, the excess network capacity can incur over-fitting.

**Comparison on existing datasets.** To provide broader contexts of RNN-T, we compare it with the state-of-the-art methods on 10 existing datasets which are used to create 3D-Skeleton-Activity-140. We follow existing methods to setup experiments (see supplementary material). Fig. 5.6 summarizes the results. As the number of classes decreases, activity recognition becomes easier, where the advantage of RNN-T is less obvious. Moreover, performance of RNN-T is restricted by insufficient training data in the cases of MSR Action3D, MSR Daily, Kintense, and ORGBD. Finally, the performance of RNN-T is inevitably bounded by the RNN it is based on. For example in sign languages in ChaLearn, HBRNN-L is not suited for modeling delicate activities, thus the improvement made by HBRNN-L-T is limited.

**5.5.2 Increasing the Activity Classes**

To create an incremental learning process, we start from a dataset with 70 classes and incrementally add 14 random new classes at a step, until 140 classes are
reached. To create a realistic simulation, we assume that the first 70 classes were trained from scratch using 100% of the training data. Then, we vary the percentage of the training data used for new classes as 100% and 80%, to see its impact on the retraining time and accuracy. We compare two incremental learning methods I-HBRNN-L-T and I-HBRNN-L, and two from-scratch learning methods HBRNN-L-T and HBRNN-L, where I-HBRNN-L is initialized with weights from HBRNN-L.

As shown in fig. 5.6, I-HBRNN-L achieves higher accuracy and less retraining time than I-HBRNN-L and HBRNN-L. I-HBRNN-L-T achieves similar accuracy compared to HBRNN-L-T, but with significantly less retraining time. We highlight two observations as the number of class increases: (i) Both I-HBRNN-L-T and HBRNN-L-T yield stable accuracy, while the performances of I-HBRNN-L and HBRNN-L fluctuate, this shows the advantage of RNN-T. (ii) The retraining time of I-HBRNN-L-T is relatively short, which demonstrates the effectiveness of our incremental learning algorithm. With the decrement of training data, I-HBRNN-L-T still outperforms three baselines in both metrics. This shows that our incremental learning algorithm is applicable when the training data is limited.
CHAPTER 6
Temporal Modeling with CNN

6.1 Background

Compared to RNN which overstretch the historical representation with a fixed-size latent vector, CNN is better at the long-term dependency modeling thanks to its hierarchical structure which maintains the dependency modeling of different term lengths at different levels. Specifically, as going to higher level, longer-term dependencies can be modeled.

Ke et al [7] represented the activity sequences as 2D clips. Specifically, each clip is generated from one channel of the cylindrical coordinates of the skeleton sequence. Each frame of the generated clips represents the temporal information of the entire skeleton sequence, and incorporates one particular spatial relationship between the joints. The entire clips include multiple frames with different spatial relationships, which provide useful spatial structural information of the human skeleton. Then, they use CNN to learn long-term temporal information of the skeleton sequence from the frames of the generated clips, and then use a Multi-Task Learning Network to jointly process all frames of the generated clips in parallel to incorporate spatial structural information for activity recognition.

Though the CNN-based approaches are effective in modeling long-term temporal dependencies, their way of modeling spatial dependencies is ad-hoc and involves hard-coding. Therefore, there still exist much improvement space in spatial dependency modeling.

One important aspect of human activity recognition is that the statistical dependencies in the spatial and temporal domains are usually intertwined. However, most previous methods model the spatial and temporal dependencies independently. Furthermore, these methods typically focus on improving individual steps of the pipeline, e.g., explicitly modeling spatiotemporal information [46, 47, 48, 62], increas-
ing sequential orders [42, 44, 46], and adjusting sequential dependency ranges [7, 49]. Due to the complexity of the activity data, we argue that a desirable method should be holistic, adaptive and flexible. This motivates us to design a deep neural network (DNN) with such merits.

Inspired by the dual-stream neural processing hypothesis of human visual neural system [63], we propose a dual-stream CNN (as shown in fig. 6.1). Instead of modeling the spatial and temporal dependency independently, the dual-stream CNN explicitly models the interweaved spatiotemporal dependency in activity sequences jointly, which is not the case for existing CNN methods of the same name, e.g., [64]. The two streams have different emphasis, either the structural or temporal dependencies, and model such dependencies at different scales. This endows our model higher adaptability and flexibility in modeling the spatiotemporal relationships of input elements, compared to the single-stream [46, 47, 48, 62] based methods. Specifically, an activity instance is first transformed into a 3-rank tensor, with the three dimensions correspond to the time steps, spatial structure and descriptive features of each sample, respectively. The two streams are implemented as CNN so that it is not limited to forward or backward directions as typical sequential models, e.g., RNN [6]. The dual-stream CNN model is based on a set of dual-stream convolution kernels, each formed as a tensor product of two 2D convolution kernels, one on the time and feature axes (red side of activity representation in fig. 6.1(a)), and one on the structure and feature axes (green side in fig. 6.1(a)), and we refer to the former as the temporal kernel and the latter as the structural kernel. The convolutional kernels of the dual-stream CNN model are organized into a hierarchical structures, namely, the structural (temporal) kernels are organized into different levels to represent features of different scales (ranges). We apply a gating module for kernels of different sequential dependency ranges so the contributions of different feature ranges can be determined adaptively in a data-driven fashion.

6.2 Dual-stream Formulation with CNN

The temporal and structural dependency are important patterns for activity recognition. The temporal dependency is commonly modeled with a chain structure
Figure 6.1: (a) The activity representation is a tensor with three dimensions, which is formed in two steps. We represent each dimension at each time step as a feature vector, and concatenate them according to the dimension and time order. (b) Module diagram. The red and green blocks represent the temporal and structural stream, respectively. L/M/HFE mean the low/medium/high-level feature extractor, and the prefixes of MFE, S/M/L, mean the short/medium/long range. ZI means the zoom-in module, and SE means the encoder shared by M-MFE and L-MFE. GT and CLS represent the gating and classification module, respectively. See texts for details.

that goes in forward or backward direction, such as in RNN. This is based on the sequential causality assumption that the intrinsic temporal dependency of a sequence along the time axis.

However, the sequential causality assumption does not always hold, as suggested by the indefinite causal order theory in quantum mechanics [65], i.e., the causality order does not always obey a specific element permutation, but a mixture of multiple permutations. We refer to such a problem as the indefinite order problem, which contradicts the sequential causality assumption made by RNN. In particular, for a RNN, long-range dependencies between two distant elements in a sequence might be affected by many other irrelevant elements on the long path through the chain. It is also hard for RNNs to accommodate such “indefinite” permutations on the fly due to the indifferentiability of permuting operations.
Mitigating the long-range dependency modeling problem and indefinite order problem requires us to avoid using the chain structure but to create an activity recognition model that is more flexible and adaptive, and to reduce the impacts caused by different element positions in a sequence. In this work, we model the sequential order of activity data using the multi-layer CNN, which creates hierarchical representations over the input activity in which the dependencies of nearby elements are modeled by lower layers while those of distant elements are modeled by higher layers. The replacement of RNN with CNN alleviates the two aforementioned problems, as the sequential dependency modeling is no longer strictly limited by a sequential order or a chain structure, but directly handled by the multi-scale receptive fields of CNN. In the following, we describe the overall processing steps, starting with an augmented activity representation and then on the structure of the model itself.

6.2.1 Activity Representation

We first augment the original activity data with empirical hand-crafted features before the feature learning process to form a rank-3 tensor $R_{d,t,f}$ and $R_{t,d,f}$ as follows. Given an activity $\{(t_i, < d_1, d_2 \rightarrow d_1, d_3 \rightarrow d_1, d_4 \rightarrow d_2, \ldots >)\}_{i=1}^{n}$ with each dimension being a point $d_j = (x_{j1}, x_{j2}, \ldots, x_{jl})$ in the $l$ dimensional space, we follow [5, 61] to extract four types of features for each dimension $d_j$ at each time step $t_i$, and concatenate them to form a feature vector $h_{d_j}^{t_i}$: (1) **Position.** $x_{j1}, x_{j2}, \ldots, x_{jl}$ are concatenated to form a $l$ dimensional feature vector; (2) **Angles.** Given multiple edges $\{e_{kj}\}_{k \in 8_j}$ connecting $d_j$ and its neighboring dimensions $8_j$ in the tree, we compute the normalized pairwise angles between these edges; (3) **Offset.** Offsets of elements in $d_j$ between $t_i$ and $t_{i-1}$ are computed and concatenated to form a $l$ dimensional feature vector. (4) **Distance.** We calculate the pairwise distance between $d_j$ and the mean position of all dimensions at $t_i$.

Then, we pad the extracted features to be equal in length, and concatenate the extracted features to form the activity representation, i.e., $R_{t,d,f} = (h_{d_1}^{t_i}, h_{d_2}^{t_i}, \ldots, h_{d_m}^{t_i})_{j=1}^{m}$; similarly, we have $R_{d,t,f} = (h_{d_1}^{t_i}, h_{d_2}^{t_i}, \ldots, h_{d_m}^{t_i})_{i=1}^{n}$ which is a transpose of $R_{t,d,f}$. The order of the activity dimensions in $R_{t,d,f}$ is determined by the traversing algorithm [44] starting from $d_1$. 
6.2.2 Model Structure

The temporal and structural streams in the dual-stream CNN model share similar structures. Each stream focuses on a distinct aspect of the spatiotemporal structure in the activity data, which is reflected by different parameterization strategy. Given an activity in the tensor representation described previously, we process the 2D slices constituted by the time and feature elements of the activity tensor with the temporal stream CNN, and process the 2D slices constituted by the structure and feature elements of the activity tensor with the structural stream CNN. For the temporal stream, the convolutional computation (parameterized) on the time and feature axes captures the temporal dependencies, while the spatial dependencies are...
captured by the addition computation (non-parameterized). The case is reversed for the structural stream. This different characteristics of the two streams suggest that the dual-stream design cannot be simply replaced by a single-stream 3D-CNN. For the choice of the convolutional kernels, we use the 2D kernel rather than using the 3D one. This is because 2D kernel performs the fully connected (shared weight for 3D kernel) computation along the time or structure axis, which makes it beneficial for modeling the long-range dependencies.

The preprocessed activity features are fed for feature learning. The adaptive selection of sequential dependency range is important to activity recognition, so we fuse the adaptive learning of the sequential dependency range into the feature learning process. There are ten building blocks within our model as shown in fig. 6.2 including low/medium/high-level feature extractor (L/M/HFE), two zoom-in modules (ZI), a shared encoder (SE), a gating module (GT) and a classification module (CLS). The MFE is composed of three sub-blocks, i.e., short/medium/long-range MFE.

As shown in fig. 6.2, we decompose the feature learning process into three stages, i.e., low/medium/high-level feature extractors (L/M/HFE). The low-level features keep more details of the original input activity data, while the high-level features are more conceptual which are used for classification directly. The medium-level features bridge the low-level and the high-level features, so it determines the reliability and the meaningfulness of the high-level features.

**LFE** consists of a conv layer with kernel size 7 and the batch normalization (BN) followed by a leaky relu (LReLu). We chose a large kernel size because the consecutive elements in a sequence may contain much redundancy. We use LReLu as the nonlinearity because it does not gate the negative values in the activity representation.

**MFE** plays the key role in feature learning, so its architecture is the most complicated. MFE is decomposed into three sub-stages with each focusing on a specific sequential dependency range corresponding to the short/medium/long-range MFE (denoted as S/M/L-MFE). The MFEs for different dependency ranges are connected by zoom-in (ZI) modules and shared encoders (SE). ZI is implemented as a max pooling layer, and SE is a block shared by the M-MFE and L-MFE. Since the space covered by L-MFE is larger than that of short/medium-range MFEs, we further split L-MFE.
into four finer scales similarly to the inception module in [66].

**GT** is posed as the backend of S/M/L-MFE which adaptively determines the contribution of each sequential dependency range to the high-level features. We implement the gating module as the gated linear unit (GatedLu) [67] over the output of the convolution \( Y = [A B] \in \mathbb{R}^{w,h,2c}, v([A B]) = A \otimes \sigma(B) \), where \( A, B \in \mathbb{R}^{w,h,c} \) are the inputs to the non-linearity, \( \otimes \) is the point-wise multiplication and the output \( v([A B]) \in \mathbb{R}^{w,h,c} \) is half size the size of \( Y \). The gates \( \sigma(B) \) (implemented as the sigmoid) determine the importance of inputs \( A \) for learning the high-level features.

**HFE** is formed with a fully connected layer (FC) which takes the flattened and concatenated medium-level features output by S/M/L-MFE, and outputs a 500 dimensional vector.

**Classification.** The extracted high-level features are fed into a softmax layer to obtain the probability distributions, which are used to compute the negative log-likelihood loss, \( L = -\sum_i y'_i \log(y_i) \), where \( y_i \) is the probability distribution output of an input activity by the softmax layer, and \( y'_i \) is its corresponding ground truth one-hot vector representation.

### 6.3 Experiments

**Experimental Settings.** We evaluate the dual-stream CNN model on three pose based activity recognition benchmark datasets: MSR Action3D [53], CharLearn Italian [50] and 3D-SAR-140 [5]. Our method is implemented using PyTorch, and all experiments are conducted on four machines on each of which an NVIDIA TITAN X GPU with 12GB onboard memory is installed. The overall objective function is minimized using back-propagation implemented with the ADAM algorithm [68]. We train the network using mini-batch gradient descent, and set learning rate, momentum and decay rate as \( 1 \times 10^{-3}, 0.9, 0.999 \). As usual, we scale the input to be equal in temporal length, so as to enable the mini-batch processing.

**Compared Methods.** We compared our method against 10 existing methods, i.e., RR [10], HBRNN-L [42], CHARM [1], DBN-HMM [3], Lie-group [9], HOD [11], MP [12], SSS [2], HBRNN-L-T [5] and URNN-2L-T [5].
Table 6.1: Classification accuracy.

<table>
<thead>
<tr>
<th>Methods</th>
<th>MSR Action3D</th>
<th>ChaLearn</th>
<th>3D-SAR-140</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>0.891</td>
<td>0.438</td>
<td>0.723</td>
</tr>
<tr>
<td>HBRNN-L</td>
<td>0.897</td>
<td>0.559</td>
<td>0.604</td>
</tr>
<tr>
<td>CHARM</td>
<td>0.747</td>
<td>0.476</td>
<td>0.618</td>
</tr>
<tr>
<td>DBN-HMM</td>
<td>0.735</td>
<td>0.628</td>
<td>0.601</td>
</tr>
<tr>
<td>Lie-group</td>
<td>0.866</td>
<td>0.401</td>
<td>0.745</td>
</tr>
<tr>
<td>HOD</td>
<td>0.844</td>
<td>0.539</td>
<td>0.657</td>
</tr>
<tr>
<td>MP</td>
<td>0.909</td>
<td>0.452</td>
<td>0.203</td>
</tr>
<tr>
<td>SSS</td>
<td>0.560</td>
<td>0.413</td>
<td>0.253</td>
</tr>
<tr>
<td>HBRNN-L-T</td>
<td>0.915</td>
<td>0.673</td>
<td>0.756</td>
</tr>
<tr>
<td>URNN-2L-T</td>
<td>0.931</td>
<td>0.753</td>
<td>0.892</td>
</tr>
<tr>
<td>Ours ⊕ gating module</td>
<td>0.947</td>
<td>0.766</td>
<td>0.864</td>
</tr>
<tr>
<td>Ours ⊕ structural stream</td>
<td>0.848</td>
<td>0.677</td>
<td>0.814</td>
</tr>
<tr>
<td>Ours ⊕ temporal stream</td>
<td>0.934</td>
<td>0.729</td>
<td>0.889</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>0.963</strong></td>
<td><strong>0.772</strong></td>
<td><strong>0.896</strong></td>
</tr>
</tbody>
</table>

**MSR Action3D.** This dataset consists of 20 activities performed by 10 subjects for two or three times, 557 valid samples with 22,077 frames. We follow the experimental protocol presented in [13] on this dataset, which is the most challenging protocol for this dataset. Half of actor subjects are used for training and the rest are used for test. Note that the average number of training samples per class is nearly 14, which is quite limited for training deep neural networks, and poses great potential risks on the overfitting issue. The comparison on MSR Action3D in table 6.1 demonstrates the good generalization capability of our method.

**CharLearn Italian.** This dataset captures 20 Italian cultural signs, and contains 393 labeled sequences with a total of 7,754 gesture instances. We follow the experimental protocol in [3]: 350 sequences for training and the rest 43 sequences for testing. The recognition of sign languages require the fine-grained recognition ability of the evaluated methods, and always desire the careful feature representation design. As shown by the comparison on CharLearn Italian in table 6.1, our method constantly outperforms the evaluated methods without designing any special features for this dataset.

**3D-SAR-140.** This dataset [5] contains 140 diverse activity classes. It is challenging due to two factors: (i) a large variety of movements in various contexts are included,
where fine-grained recognition is required; (ii) sequence length for individual activities varies significantly (ranging from 5 to 800 frames), which poses the challenges on the adaptive configuration of the sequential dependency range. Notably, URNN-2L-T is designed to recognize fine-grained activities and large-scale dataset. However, our method still performs slightly better than URNN-2L-T. The good performance also demonstrates our method’s effectiveness in adaptively configuring the sequential dependency ranges.

**Ablation Study.** We evaluate three components in our method, i.e., the gating module, structural stream and temporal stream. We disable these components one by one and conduct the evaluation. Table 6.1 shows the comparison results, and it clearly shows that each of these three components is beneficial for the generalization. Generally speaking, the descending order of their influences is structural stream, gating module and temporal stream. An interesting phenomenon is that the structural stream seems to have more important effect than the temporal stream in the final classification performance, which accords with the theory of dual-stream hypothesis in neural science. This strengthens the reasonability of our inspiration drawn from the dual-stream hypothesis, and further shows the necessity of interweaving spatiotemporal modeling for activity recognition.
CHAPTER 7
Spatial Modeling with GCN

Individual activity recognition is an important research problem with many applications, *e.g.*, security surveillance, smart home, and video retrieval based on events *etc.* Compared to group activity recognition, the individual one focuses more on the human body motion patterns. As mentioned in section 2.1.2, the modality of human skeletons are sensitive to the human body motions, so most recent promising individual activity recognition methods are based on the input of 3D human skeletons. In fig. 7.1 we visualize an example of a 3D human skeleton.

### 7.1 Background

GCN is naturally suitable for modeling spatial dependencies because of its flexibility and capability in capturing the inter-node dependencies. Therefore, the focus of the community shifted to using GCN for individual activity recognition.

Yan *et al* [40] and Li *et al* [69] introduced the GCN into individual activity recognition which treats the skeleton joints as the nodes in a graph, and uses the predefined adjacency matrix (*i.e.*, determined by the connections of skeleton joints). The graph node-wise representations are extracted through a hierarchical structure with the topology remaining unchanged. Finally, the graph representation are extracted by pooling all node-wise representations. The pooled graph representation is classified to get the recognition result. The directions of subsequent works follow three major directions:

**Graph Node.** Both [40] and [69] use skeleton joints as the graph nodes. Zhang *et al* [70] for the first time propose to use bones as the additional graph nodes, and use two-branch CNN to extract features from the two types of nodes. Similarly, Hu *et al* [71] propose to use the “velocity” (temporal difference between joints) as the additional graph nodes, and use two separate branches to extract features. The two branches of [70] and [71] are separated, so the representations of different types of
Figure 7.1: An example of human skeleton input for individual activity recognition. Body joints are indexed in odd numbers, and bones are in even numbers. The body parts are categorized into five groups (highlighted in different colors) according to the human body structure.

nodes are extracted independently.

**Connectivity.** Most existing works adopt simple temporal modeling strategy as defined in section 4.3.4 in which for the temporal domain, only the dependencies between the corresponding nodes across frames are considered. In [72], Gao et al. proposed a more generalized temporal connectivity by enabling the potential connections between relevant nodes (regardless of the correspondence) across frames. In
addition to the temporal connectivity study in [72], Shi et al [41] investigated the adaptability of spatial connectivity of GCN. As introduced in section 4.3.3 [41] do not only use the predefined adjacency matrix but also the learnable adjacency matrix and the attentively estimated one.

**Heterogeneous Graph.** As suggested by [70][71], a sequence of skeletal poses contain so rich information that it requires multiple heterogeneous graph representations for encoding, joint vs. bone [70], and position vs. velocity [71]. However, as
shown in fig. 7.2(a), these methods handle the heterogeneous graph representations using multiple independent branches on the basis of assumed feature orthogonality. This contradicts to the fact that the heterogeneous nodes are usually interdependent, e.g., the states of joint and bone. Therefore, we argue that it is beneficial to bond such heterogeneous nodes more tightly, as shown in fig. 7.2(b), for the sake of a more expressive graph representation.

7.2 Summary

Therefore, we dissect the typical graph convolution operation, and study how to bond the heterogeneous graph representations regarding two key aspects, i.e., weight function and graph connectivity. For the weight function, we evaluate three flexibility-varied options for linear fusion of heterogeneous graph representations, and study their impact on GCN’s performance. As for the graph connectivity, we investigate the capability of the predefined connectivity, attentively estimated connectivity and the learnable connectivity. We select the most common joint-bone heterogeneous graphs [41, 70] for analysis, and create multiple GCN alternatives in correspondence with different bonding strategies by alternating the options of the two graph convolution aspects. Since these two aspects are of domain-agnostic, so our empirical conjectures about the bonding strategies should be generic even if they are made on the basis of a specific combination of heterogeneous graphs.

7.3 Graph Convolution Aspects

In this section, we introduce the two graph convolution aspects that impact the bonding strategies for heterogeneous graph representations. Before diving into the introduction of these aspects, we first revisit the formulation of graph convolution to show a big picture. The formulation of GCN is defined in eq. (4.14). We first reformulate the graph convolution for the joint branch and bone branch in fig. 7.2(a):

\[ f_{\text{out}}^{\text{joint}} = \sum_{n} W_{n}^{\text{joint}} f_{\text{in}}^{\text{joint}} A_{n}^{\text{joint}}, \quad (7.1) \]
7.3 Weight Function

We introduce three weight functions for the linear fusion of heterogeneous graph representations, namely (i) exclusive weight, (ii) intra weight and (iii) inclusive weight. We illustrate the usage of three weight functions in fig. 7.3. For the exclusive weight function, the representation of a certain gene is not affected by the other. For the intra weight function, the representations of different genes contribute to each other. However, the initial features are weighted independently. On the contrary, for the inclusive weight function, the initial features are weighted by considering features of the other gene. We first provide the general definition of the three weight
functions as follows:

\[
f_{\text{joint\ out}} = \sum_{n}^{N_p} (f_{\text{joint\ mid}} A_n^{\text{joint}} + f_{\text{bone\ to\ joint\ mid}} A_n^{\text{bone\ to\ joint}}),
\]
(7.5)

\[
f_{\text{bone\ out}} = \sum_{n}^{N_p} (f_{\text{bone\ mid}} A_n^{\text{bone}} + f_{\text{joint\ to\ bone\ mid}} A_n^{\text{joint\ to\ bone}}),
\]
(7.6)

where

\[
f_{\text{joint\ mid}} \oplus f_{\text{bone\ to\ joint\ mid}} = W^{\text{joint\ to\ bone}} (f_{\text{in\ joint\ mid}} \oplus f_{\text{in\ bone}}),
\]
(7.7)

\[
f_{\text{bone\ mid}} \oplus f_{\text{joint\ to\ bone\ mid}} = W^{\text{bone\ to\ joint}} (f_{\text{in\ bone\ mid}} \oplus f_{\text{in\ joint}}).
\]
(7.8)
In fig. 7.4, we visualize $W_{\text{joint-bone}}$ and $W_{\text{bone-joint}}$ for three weight functions. The larger the frozen zero area is, the higher flexibility of a weight function is. It is obvious that the flexibility of the weight functions increases as it goes from exclusive to intra and further to inclusive.

7.3.2 Graph Connectivity

We discuss the graph connectivity in section 4.3.3. In addition to the predefined connectivity, we introduce the attentively estimated connectivity and the learnable connectivity. Among these three types of connectivity, the predefined and learnable ones are input-independent, while the attentively estimated one is input-dependent. We compare three variants of graph convolution, i.e. (i) the predefined, (ii) the attentive, and (iii) the complete (all three types of connectivity).

7.4 Experiments

7.4.1 Implementation Details

We follow the separate-branch architecture in [70], and created nine baselines by permutating the options of weight functions and types of graph connectivity, i.e., (i) exclusive-predefined, (ii) exclusive-attentive, (iii) exclusive-complete, (iv) intra-predefined, (v) intra-attentive, (vi) intra-complete, (vii) inclusive-predefined, (viii) inclusive-attentive, (ix) inclusive-complete. Our methods are implemented with PyTorch. All experiments are conducted on a machine with 4 NVIDIA Tesla M40 GPUs with 96GB on-board memory.

7.4.2 Datasets and Experiment Setup

MSR Action3D. This dataset consists of 20 actions performed by 10 subjects in an unconstrained way for two or three times, 557 valid samples with 22077 frames. All sequences are captured in 15 FPS, and each frame in a sequence contains 20 skeleton joints. We follow the experimental protocol presented in [53] on this dataset, which is the most challenging protocol for this dataset. Half of actor subjects are used for training and the rest are used for test. Note that the average number of
training samples per class is nearly 14, which is quite limited for training deep neural networks, and poses great potential risks on the overfitting issue.

**3D-SAR-140.** This dataset is proposed in [5], which contains 140 diverse action classes by aggregating all distinct classes from 10 existing datasets. 3D-SAR-140 is a challenging benchmark due to two factors: (i) a large variety of movements and dynamics in various contexts are included, where fine-grained recognition is required; (ii) sequence length for individual actions varies significantly (ranging from 5 to 800 frames) within or across classes, which poses the challenges on the adaptive configuration of the sequential dependency range.

**NTU RGB+D.** This dataset [73] contains more than 56000 samples, and includes 60 classes. To our knowledge, this is the largest skeleton based activity recognition dataset. There are two standard evaluation protocols: (i) cross-subject: 20 subjects are used for training, and the remaining 20 subjects are for testing; (ii) cross-view: two view-points are used for training, and one is for test. The large amount of variations in subjects and views make this dataset challenging.

**Compared Methods.** We compare our 9 model variants with ST-GCN [47]. The original ST-GAN takes the graph of joints as input. We also build a variant of ST-GAN that takes the graph of bones as input. Note that the baselines named with the prefix “exclusive” are in fact the variants of [70], which is consists of two independent branches of GCN.

**Evaluation Metric.** We adopt the recognition accuracy as the evaluation metric for individual activity recognition.

### 7.4.3 Results and Analysis

We show the performance comparison of the evaluated methods in table 7.1 and table 7.2.

**Necessity of Exploration.** The best performances on three datasets are always achieved by models in which the heterogeneous graph representations are effectively bonded. Therefore, it is necessary to explore the bonding strategies.

**Graph Connectivity.** First of all, based on the comparison of different connec-
Table 7.1: Comparison on MSR Action3D and 3D-SAR-140. The best, 2nd best and 3rd best performances are indicated in red, magenta and blue, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>MSR Action3D</th>
<th>3D-SAR-140</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) ST-GCN-joint</td>
<td>85.35</td>
<td>82.70</td>
</tr>
<tr>
<td>(b) ST-GCN-bone</td>
<td>86.81</td>
<td>85.62</td>
</tr>
<tr>
<td>(c) exclusive-predefined</td>
<td>85.35</td>
<td>86.36</td>
</tr>
<tr>
<td>(d) exclusive-attentive</td>
<td>89.38</td>
<td>82.57</td>
</tr>
<tr>
<td>(e) exclusive-complete</td>
<td>87.91</td>
<td>85.37</td>
</tr>
<tr>
<td>(f) intra-predefined</td>
<td>85.71</td>
<td>86.13</td>
</tr>
<tr>
<td>(g) intra-attentive</td>
<td>90.11</td>
<td>88.04</td>
</tr>
<tr>
<td>(h) intra-complete</td>
<td>92.31</td>
<td>88.17</td>
</tr>
<tr>
<td>(i) inclusive-predefined</td>
<td>84.62</td>
<td>88.93</td>
</tr>
<tr>
<td>(j) inclusive-attentive</td>
<td>90.84</td>
<td>89.31</td>
</tr>
<tr>
<td>(k) inclusive-complete</td>
<td>89.38</td>
<td>89.31</td>
</tr>
</tbody>
</table>

Table 7.2: Comparison on NTU RGB+D. The best, 2nd best and 3rd best performances are indicated in red, magenta and blue, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>Cross-sub</th>
<th>Cross-view</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) ST-GCN-joint</td>
<td>80.54</td>
<td>87.75</td>
</tr>
<tr>
<td>(b) ST-GCN-bone</td>
<td>84.04</td>
<td>91.34</td>
</tr>
<tr>
<td>(c) exclusive-predefined</td>
<td>85.18</td>
<td>91.74</td>
</tr>
<tr>
<td>(d) exclusive-attentive</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(e) exclusive-complete</td>
<td>87.26</td>
<td>93.28</td>
</tr>
<tr>
<td>(f) intra-predefined</td>
<td>86.65</td>
<td>92.35</td>
</tr>
<tr>
<td>(g) intra-attentive</td>
<td>80.62</td>
<td>86.91</td>
</tr>
<tr>
<td>(h) intra-complete</td>
<td>89.06</td>
<td>94.12</td>
</tr>
<tr>
<td>(i) inclusive-predefined</td>
<td>85.76</td>
<td>89.53</td>
</tr>
<tr>
<td>(j) inclusive-attentive</td>
<td>85.59</td>
<td>89.48</td>
</tr>
<tr>
<td>(k) inclusive-complete</td>
<td>86.60</td>
<td>89.76</td>
</tr>
</tbody>
</table>

...tivity settings, we can conclude that the “complete connectivity” always lead to better performance than the others. Second, as shown by table 7.1 (d) and table 7.2 (d,g), the reliability of the attentively estimated graph connectivity cannot be guaranteed. We argue that the reliability may be arisen by the intra-class variations. The degree of intra-class variations increase follow the order of MSR Action3D, 3D-SAR-140, and NTU RGB+D, which complies with the observed performance impacts caused by the unreliable attentive connectivity.

**Weight Function.** As claimed in section:aspects:weightfunc, the flexibility
of the weight functions increases as it goes from exclusive to intra and further to inclusive. As the most “flexible” weight function, the performances of “inclusive weight function” on MSR Action3D and 3D-SAR-140 are better than those on NTU RGB+D. We argue that the flexibility of the “inclusive” help the model overfit the datasets where the intra-class variations are not that severe, e.g., MSR Action3D and 3D-SAR-140, but make the model suffer when the intra-class variations are significant. We also observe that the linear fusion of heterogeneous graphs can make up for the unreliability of the graph connectivity estimation, which is demonstrated by table 7.1 (g,j) and table 7.2 (g,j).

Group activity recognition is an important research problem with many applications, e.g., security surveillance, sports analysis, and video understanding etc. In individual activity recognition, the partial graph topology can be predefined, e.g., the connectivity of skeleton joints. On the contrary, in most cases, the graph topology for the group activity recognition cannot be predefined at all. In addition, the graph topology is an important cue to infer the group activity recognition. Another difference between the individual-level and the group-level tasks is that the intra-class variations for the individual motions are much larger at the group-level compared to the individual-level. Therefore, for group activity recognition, seldom are used the human poses or skeletons, but most works use the modality of RGB videos as input.

7.5 Background

Two-stage LSTM. Ibrahim et al [74, 75] first introduce the two-stage LSTM in this area, in which the first stage models the person-level representation, and the second stage models the group-level representation based on the pooled person-level representation. There are several subsequent works that follow the two-stage formulation. For example, Kong et al [76] introduced the attention mechanism based on which to estimate the key body parts at the person-level and key persons at the group-level. Then, they use the estimated attention weight to guide the feature aggregation so that the aggregated features are more expressive. Yan et al [77] modified the two-stage formulation to exploit the dynamics-degree information, so that the dynamical persons can be identified as the key persons in the group. The dynamics-
aware person-level representations are fed into the LSTM at the second stage for the recognition purpose. A major draw-back of these methods is that the inter-person dependencies, which should contain much information, are either ignored or treated sloppily with an ad-hoc sequential modeling.

**Distracted Inter-person Dependency Modeling.** Then, several works spend more efforts in modeling the inter-person dependency modeling within the two-stage LSTM framework. Shu et al. [78] models the inter-person dependency within each LSTM block at the group level by a co-memory cell, which simply multiplies the memory state of each person. Qi et al. [79] modeled the inter-person dependency via a message passing layer which is built upon a complete graph. Biswas et al. [80] modeled the local inter-person dependency via the grid pooling, which treats all n-nearby persons equally. These methods model the inter-person dependency based on a strong assumption that the contextual persons are equally important to augmenting the representation of the current person. We argue that such a distracted assumption restricts the capability of their model of inter-person dependency.

As demonstrated by the previous methods, we observe that the inter-person dependency is an important cue to the group activity recognition. However, to our best knowledge, there are no existing works that models the inter-person dependency adaptively. Therefore, we explore to use GCN to model the adaptive inter-person dependency so as to assist the group activity recognition.

### 7.6 Summary

We propose an intuitive GCN-based approach to address the group activity recognition problem. Our model takes the person trajectories as input, and use a pretrained CNN model to extract the person-level representations across the temporal domain. Then, we use the proposed GCN model to augment the representation of each person with the contextual persons’ representations according to the attentively estimated graph connectivity. The augmented person-level representations are the pooled to form the raw group-level representation, which is then fed into the group-level RNN for the recognition purpose.
7.7 Formulation

Given the raw person-level representations $f_{\text{in}} \in \mathbb{R}^{C_{\text{in}} \times T \times N_0}$, we augment $f_{\text{in}}$ with the graph convolution:

$$f_{\text{out}} = \sigma(Wf_{\text{in}}A) + f_{\text{in}},$$

(7.9)

where $W \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times 1 \times 1}$ is the convolutional kernel. $A \in \mathbb{R}^{T \cdot N_0 \times T \cdot N_0}$ is the attentively estimated adjacency matrix encoding the connections of persons throughout the spatio-temporal domain, which is computed as

$$A = \text{softmax}(\mathcal{R}(f_{\text{in}})^T W_\theta W_\varphi \mathcal{R}(f_{\text{in}})),$$

(7.10)

where $\mathcal{R}(\cdot)$ is a reshape function that views the shape of $f_{\text{in}}$ from $\mathbb{R}^{C_{\text{in}} \times T \times N_0}$ to $\mathbb{R}^{C_{\text{in}} \times T \cdot N_0}$. $W_\theta$ and $W_\varphi$ represent the linear layers for mapping the representations of different nodes into the same space.

The augmented person-level representations $f_{\text{out}} \in \mathbb{R}^{C_{\text{out}} \times T \times N_0}$ are then fed into the max pooling layer to get the raw group-level representations $f_{\text{in}}^g \in \mathbb{R}^{C_{\text{out}} \times T}$. $f_{\text{in}}^g$ is finally fed into a GRU for classification.

7.8 Experiments

7.8.1 Implementation Details

We use the backbone of [81] which is an inception V3 model as the CNN backbone to extract the raw person-level representations. Our method is implemented with PyTorch, and the experiments are conducted on a machine with 4 NVIDIA GTX 1080 Ti with 44GB on-board memory.

7.8.2 Dataset and Experiment Setup

**Dataset.** We evaluate our method on the volleyball dataset [74], since it is the only publicly available dataset for multi-person activity recognition that is relatively large-scale and contains labels for people locations, as well as their collective and
Table 7.3: Performance comparison on the volleyball dataset. The best, 2nd best and 3rd best performances are indicated in red, magenta and blue, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>Group-level</th>
<th>Person-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) two-stage [75]</td>
<td>n/a</td>
<td>81.90</td>
</tr>
<tr>
<td>(b) CERN [82]</td>
<td>69.00</td>
<td>83.30</td>
</tr>
<tr>
<td>(c) key-parts-person [76]</td>
<td>n/a</td>
<td>85.10</td>
</tr>
<tr>
<td>(d) participation [77]</td>
<td>n/a</td>
<td>87.70</td>
</tr>
<tr>
<td>(e) grid-pooling [80]</td>
<td>76.65</td>
<td>83.47</td>
</tr>
<tr>
<td>(f) co-memory [78]</td>
<td>n/a</td>
<td>88.40</td>
</tr>
<tr>
<td>(g) message-passing [79]</td>
<td>81.90</td>
<td>89.30</td>
</tr>
<tr>
<td>(j) ours w/o GCN</td>
<td>82.40</td>
<td>89.90</td>
</tr>
<tr>
<td>(k) ours</td>
<td>83.20</td>
<td>91.70</td>
</tr>
</tbody>
</table>

individual actions. This dataset consists of 55 volleyball games with 4830 labelled frames, where each player is annotated with the bounding box and one of the 9 individual actions, and the whole scene is assigned with one of the 8 collective activity labels, which define which part of the game is happening. For each annotated frame, there are multiple surrounding unannotated frames available. We resort to the same appearance based tracker proposed by the authors of the dataset [74] to get the people’s trajectories.

**Compared Methods** are discussed in detail in section 7.5.

**Evaluation Metric.** We adopt the recognition accuracy as the evaluation metric for group activity recognition.

7.8.3 Results and Analysis

In table 7.3 from (a) to (d), we show the performance of methods that do not have explicit inter-person dependency modeling; from (e) to (g), we show the performance of methods with the distracted inter-person dependency modeling. We show the performance of our method in table 7.3 (k) and its ablative version (without GCN) in (j).

It shows that as the inter-person dependency improves, the overall performance also improves. The reason why the performance of (e) is weaker than those of (f) and (g) is because restrict local constraints posed by the grid-pooling strategy significantly
limits the model flexibility.

Another interesting phenomenon is that not only our method outperforms the other methods, but our ablative version also does so. We argue that this is because of the different backbones used. For others’ method, they use either the Alex net or the VGG as backbones. However, we use the inception V3 model which has better multi-scale modeling. We argue that the multi-scale modeling is a critical issue here. Since the person regions are quite small, as the depth of the backbone increases, the actual receptive field of the high-level layers will be very large. At this time, the cropped high-level person feature maps will contain much “background” information, which will affect the feature expressiveness significantly.

Finally, we can conclude based on the comparison between (j) and (k) that GCN can bring obvious performance gain thanks to its outstanding capability of inter-person dependency modeling.
CHAPTER 8
Future Directions

Despite the great progress achieved by the existing pose based activity recognition methods, there are several severe issues that prevent the landing of these technologies. Thus, we summarize the following directions that future research efforts can be put on.

8.1 Integration of Other Modalities

As argued in section 2.1.2 of Chapter 2, different modalities are suitable for different scenarios. Therefore, how to integrate the advantages of different modalities and avoid their disadvantages remains a open and challenging problem. A general approach is to adaptively estimate the suitable modality for each input activity instance. However, the explicit estimation in the inference stage would induce new challenges which is not an ideal approach. Therefore, the future research may focus on studying how to achieve the modality integration and the implicit modality estimation through a novel training strategy.

8.2 Solving the Curse of Large Scale

A good training dataset is crucial to training a generalized and robust model. However, to create such a balanced dataset is quite difficult and there is also lacking metrics for measuring the dataset quality. Therefore, based on the available datasets, it remains an important open problem for how to smartly select the training samples from the dataset to train a model.

8.3 Avoiding the Trap of the Discriminative Model

The goal of training a recognition (or discriminative) model is to learn a robust mapping from the input observations to the predefined class labels. Unlike other
recognition tasks, *e.g.*, object recognition, the intra-class variations is much larger for activity recognition. For instance, a slight stylized variations in performing an activity would cause the failure. Therefore, researchers should not be absorbed in overfitting their models on the existing datasets, but put more efforts in designing models for easing the aforementioned mapping process. Generation is a harder task compared to the discrimination, and it is also a dual problem of the discrimination task. Thus, if we have a good feature representation for generation, the same representation should be useful for lower the difficulty of the discrimination task.
BIBLIOGRAPHY


[52] CMU: CMU graphics lab motion capture database. \url{http://mocap.cs.cmu.edu/} (2013)


