Human-human Interaction Classification and Detection in Videos

Master of Science Thesis

By
DELIANG WU

Utrecht University

Department of Information and Computing Science
UNIVERSITY UTRECHT
Supervisor: dr. ir. R.W. Poppe

Wuhan University

School of Remote Sensing and Information Engineering
WUHAN UNIVERSITY
Supervisor: Prof. dr. Zhenzhong Chen

A dissertation submitted to the University of Utrecht in accordance with the requirements of the degree of MASTER OF SCIENCE in Artificial Intelligence .

JANUARY 2017

Word count: ******

Abstract

Here goes the abstract

Dedication and acknowledgements

Here goes the dedication.

TABLE OF CONTENTS

List of Tables

Table **Page**

List of Figures

Figure **Page**

CHAPTER 1

INTRODUCTION

The technique of automatic video content analysis (VCA) is one of the most important areas of Computer Vision and Artificial Intelligence. With this technique, machines are able to recognize objects, human activities and events in videos. Thus, it can be widely used in many domains, such as human-computer interaction, video classification, entertainment, self-driving, public safety and security, home automation, etc. Action and interaction analysis is one of the most common uses of VCA which focuses on the analysis of human activities, such as the recognition/detection of the action of a single person or recognition/detection of the interaction of two or more people.

1.1 Background

When we talk about human action, we usually mean the activity of a single person. But in truth, human interaction is more complex and generally contains three cases: human-human interaction between two or more people, such as hand-shaking; human-object-human interaction like a person passing an object to another person; and human-object interaction like a person pushing a table.

The goal of **human action/interaction recognition** in videos is to determine the action/interaction labels. And the goal of **human action/interaction detection** in a video is to locate a specific action/interaction spatially and temporally in the video.

Although a lot of excellent methods and datasets have been published in the last decade in the area of action recognition, it is still a challenging task and is far from being solved. Comparatively, the related work in the domain of interaction recognition and detection is relatively scarce. This is because the job of interaction recognition and detection is even more challenging than that of action recognition and detection.

The main challenges of action and interaction recognition and detections in real scenes mainly include:

1. Various camera views. The videos which will be analysed could be taken from different viewpoints which may have never been seen before in the training data. For example, Figure 1.1 (a) illustrates four videos of the same biking little girl being filmed from four different camera views. The features extracted from these videos can be different, which would then make it hard for the classifier to learn to discriminate them as the same activity.

2. Complex background. The background of the action/interaction could vary and even be totally different. For example, Figure 1.1 (b) illustrates four diving videos taken with totally different backgrounds with large camera motions. For those feature descriptors that extract features from not only the segmented person but also from the background. Then the features extracted from the background would largely confuse the classification results.

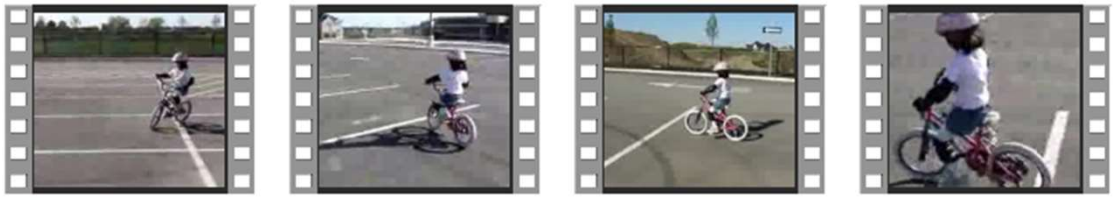
3. Usually, a single action video clip contains at least hundreds of frames. Therefore, there might exist many irrelevant frames which would, in turn, confuse the analysis process. For instance, a video clip of a basketball game may contain frames of commentators and the audiences which wouldn't be relevant.

4. It is hard to get decent performance on a small training set. But sometimes we only have small scale target datasets with very limited training samples. Such situation is even worse for the task of interaction video analysis since the related dataset is scarce compared with the action video analysis.

5. Compared with action video analysis, extra features like relative position and orientation between the people involved in the interaction need to be taken into consideration, which makes the representation of features more complex.

Some of the related works for interaction analysis are [26], [35], [22] and [2]. The works of Patron-Perez et al. [26] and Gemeren et al. [35] focus on the relative information between the people involved in the interaction, such as their relative position and orientation in relevant body parts and, then uses hand-crafted feature descriptors, such as histograms of orientated gradients (HOG [3]) and histograms of optical flow (HOF [5]), to describe those interaction features. The work of Narayan et al. [22] combines the improved trajectory features and the foreground motion map features to describe interaction features. In contrast, Choi et al. [2] introduce a hierarchical method, which first detects and tracks the location of each person and then learns to analyse the atomic action of each person. Lastly, it uses the atomic action to infer the collective interaction label. The atomic action is also represented by the hand-crafted feature descriptors, such as HOG and bag of video words (BoV [6]).

In this thesis project, we adopt a similar hierarchical architecture with the work of Choi et al. with two main changes: 1). we use the deep learning feature descriptors to represent the atomic



(a). A same biking girl filmed in 4 different views

(b). 4 diving videos with totally different backgrounds

Figure 1.1: Illustration of some challenges of video analysis.

action for each person involved in the interaction instead of hand-crafted feature descriptors; 2). Besides having the networks learning atomic features of each person, we add an additional deep learning network to learn the features of relative position and orientation.

1.2 Project Goals

In this thesis project, we focus on the recognition and detection of two-person interaction. The goals of this thesis project include:

1. To do interaction recognition based on the target dataset. The inputs are the segmented specific interaction videos and the outputs are the predicted class labels.

2. To do interaction detection based on the target dataset. The input is the un-segmented videos and the output is the spatial and temporal location of specified classes.

3. Construct a hierarchical multi-level network to learn the features of interaction. Hierarchical network means that we first learn atomic action features for each person involved in interaction, then combine these features to learn interaction features. Multi-level network means that we have a global first level network which learns global interaction features, such as the relative position and orientation between the people involved in the interaction and a second level network which learns the atomic actions of each person.

1.3 Contributions

We mainly have the following contributions for the interaction video analysis:

1. We introduce deep learning feature descriptors that address the interaction recognition and detection with only a small scale target interaction video dataset available.
2. We introduce a hierarchical multi-level framework that will be used for interaction video analysis tasks.

1.4 Outline

In the next chapter, we will introduce action and interaction video analysis related works including various methodologies and related datasets. In Chapter 3, we will describe our overall architecture and low-level design of this project in detail. In Chapter 4, we will introduce the experiments designs, their experimental results, and analyse them. At last, we will give the conclusions and possible future works of this project.

CHAPTER 2

RELATED WORK

Due to the large potential practical values in many domains and the big technical challenges, video analysis has become a very hot research topic in both the academic and industrial communities in recent years. Since related works of interaction video analysis, such as [26] [35] [22] [2], are relatively scarce compared to those in closely related areas, like action video analysis [12] [23] [34] [15] [7] [13] [31], we will introduce both the architectures used in interaction and action video analysis related works in Section 2.1. The general architecture of a video recognizer is usually comprised of a feature descriptor and a classifier. The video feature description research can be mainly divided in two directions: hand-crafted feature descriptors and learning based feature descriptors. We will introduce these two types separately in Section 2.2 and 2.3 respectively. Another very important factor that determines the performance of the video analysis is the training datasets. Without a rich and nicely annotated training dataset, it is hard to get a decent performance even for the best-built algorithm. So, we will introduce popular publicly available action and interaction video datasets in Section 2.4.

2.1 Architectures of Interaction video analysis related works

Choi et al. [2] introduce a hierarchical network that recognizes the collective activity performed by a group of people. The hierarchical model is illustrated in Figure 2.1 (b). O_i and O_j are the features of each individual in the video, and O_c is the crowd context feature which represents the overall information of the video. A is the atomic action of each individual, I is the interaction between two individuals and C is the collective activity of the group of people. For example, the collective activity "$gathering$", illustrated in Figure 2.1 (a), is characterized as a collection of interactions (such as "$approaching$") between individuals. Each interaction is described as pairs of atomic activities (for example "$facing-right$" and "$facing-left$"). Each atomic activity is

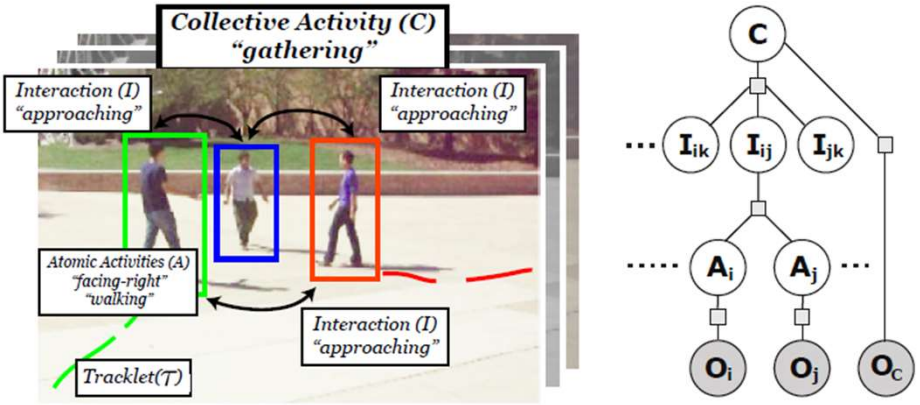


Figure 2.1: The hierarchical activity recognition model and an example. Reprinted from [2].

associated with a spatial-temporal trajectory. [2] use HOG to represent the appearance features and use bag of video words (BoV [6]) to represent the spatial-temporal features.

Compared with the work of Choi et al. which uses a hierarchical model to recognize group activities, Gemeren et al. [35] extract the interaction features from the local body parts of the interacting people and focus on those parts of the videos that characterize the interaction. This is helpful in distinguishing between the interactions that only have very slight differences. HOG (appearance) and HOF (movement) are combined as feature descriptors in this work.

Patron-Perez et al.'s work [26] is similar to [35]. Patron-Perez et al. detect and track each person's upper body in the interaction video and crop the tracklet to generate a person-centered descriptor with HOG and HOF. Besides, the head orientation is also taken into account in the feature descriptor because the head orientation is also an important cue for the interaction video analysis.

There are also some methods that use a bag of the local features in order to describe the features of interaction, like the work of Yimeng et al. [41] and Yu et al. [16]. Yimeng et al. propose the concept of spatio-temporal phrase which is a combination of the local features in a certain spatial and temporal structure including their order and relative positions, then the video is represented by a bag of spatio-temporal phrases. Yu et al. propose the use of a set of attributes and the pair-wise co-occurrence relationship of two attributes to describe the videos. An attribute is a binary representation of a body part, such as that 'the torso is still'. A co-occurrence relationship of two attributes is a binary representation of the association between the two attributes. For example, the co-occurrence relationships between the attributes "torso bending" and "still leg" in the activity "bow".

2.2 Hand-crafted Feature Descriptor

The scale-invariant feature transform (SIFT [19] [20]) and the histogram of oriented gradients (HOG [3]) feature descriptors achieve great success and are widely applied in the image content analysis.

SIFT is a local feature describing algorithm which detects the interest points in an image and computes the gradients for each interest point in order to construct the features of an image. The SIFT algorithm adopts the difference of Gaussian (DoG) algorithm to detect the interest points by constructing an image pyramid with several scales and blurring images with several different Gaussian filters in each scale. The interest points are those points that have maximum values compared with their neighbor pixels in the image pyramid. Thus, those points with edges and corners which represent the main feature of an object are most likely to be found as interest points. Because the interest points are detected in different scales, the SIFT feature descriptor is scale invariant. Gradient computing is applied to the interest point centered blocks for each interest point. The computed histogram of gradient features are rotated to the main orientation for each interest point. Thus, it is rotation invariant. And finally, the features are normalized in order to further eliminate the effect of different lighting.

For the HOG algorithm, the image is divided into several small cells. The histograms of gradient (orientation and magnitude) are computed for each cell. This is because the appearance and shape of an object can be nicely described by the gradient. The HOG feature descriptor is somewhat transform invariant because the features are computed in local cells, and because the features are normalized over the sliding overlap blocks which contain several cells in a single block so it is lighting invariant.

Though SIFT and HOG can efficiently describe appearance and shape features in images, but they can not be directly used for video analysis tasks, because they not only require appearance and shape description but also motion description. Thus, they are also extended to 3D-SIFT[7] [30] and 3D-HOG[15] in order to describe the video features. Among all current hand-crafted feature descriptors, the improved Dense Trajectories(iDT)[37][38] have been shown to perform the best on a variety of datasets.

2.2.1 3D-SIFT Feature Descriptor

Compared with 2D-SIFT [19] [20] which only considers x and y dimensions, 3D-SIFT [7] [30] takes another dimension, in this case, $time(t)$, into consideration because the motion information contained in time dimension is an essential cue for video analysis. There are two steps involved in the constructing of the SIFT feature descriptor. The first step is the localization of the key points and the second step is the calculation of the sub-histograms for each key point. The method of key points localization in the 3D-SIFT is the same as that in the 2D-SIFT. The main differences between 2D-SIFT and 3D-SIFT is the calculation of the sub-histogram for each key point.

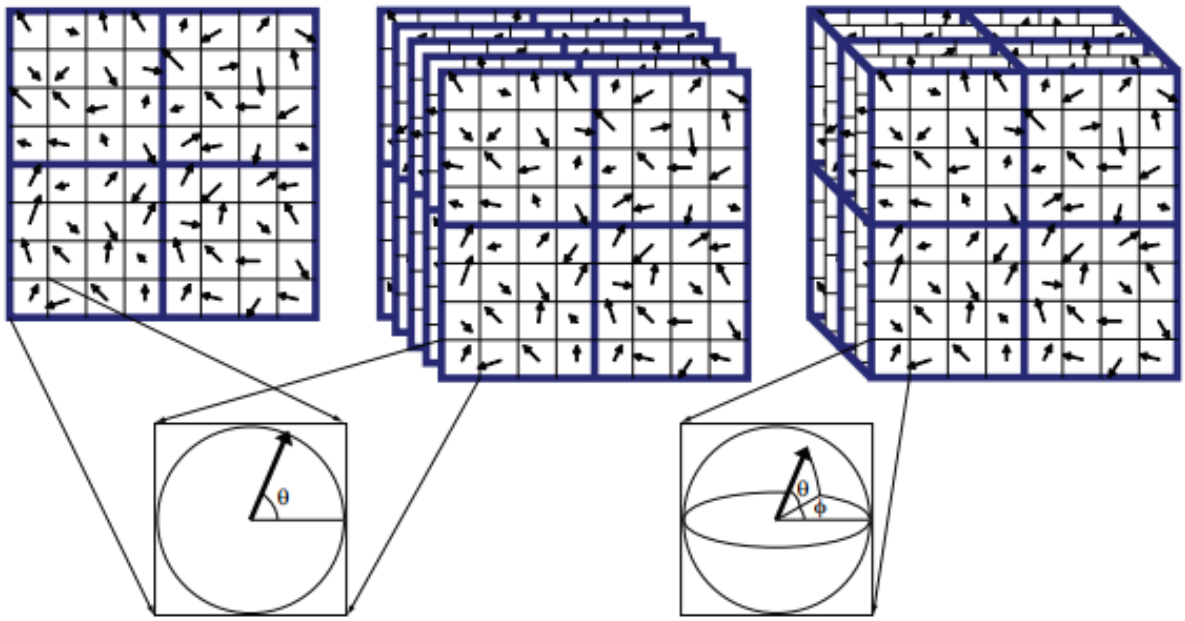


Figure 2.2: The SIFT descriptor. The left image shows the 2D SIFT descriptor. The center image shows how multiple 2D SIFT descriptor could be used on a video without modification to the original method. The right image shows the 3D SIFT descriptor with its 3D sub-volumes, each sub-volume is accumulated into its own sub-histogram. These histograms are what makes up the final descriptor. Reprinted from [7].

The 2D gradient magnitude and orientation for each pixel are calculated in x and y dimensions, while 3D-SIFT calculates gradients in three dimensions x, y and t. Thus, the motion information along the time dimension is also well presented by the sub-histogram of 3D gradients.

The 3D-SIFT feature descriptor is illustrated in Figure 2.2. After getting the sub-histogram, the orientation of each key point could be fixed. In 3D-SIFT, the dominant orientation of the key point could be represented by θ and ϕ. To keep the orientation invariant, all neighbourhoods of each key point are rotated to the dominant orientation. 3D-SIFT also inherits the method of key points detection and histogram normalization from 2D-SIFT. Thus, it is also scale and lighting invariant.

2.2.2 3D-HOG Feature descriptor

2D-HOG is widely used for the purpose of object detection in static images. 2D-HOG calculates the gradients for each pixel and accumulates them as the histograms of orientated gradient over cells. Klaser et al. extend it to 3D-HOG [15]. In Klaser's work, the main difference between 2D and 3D HOG is the calculation of the gradient. In 2D-HOG, only two dimensions x and y are taken into account, the cell is a square. While in 3D-HOG, additional time dimension t is taken into consideration, so the cell is a cube. The blocks which are used to contrast-normalization

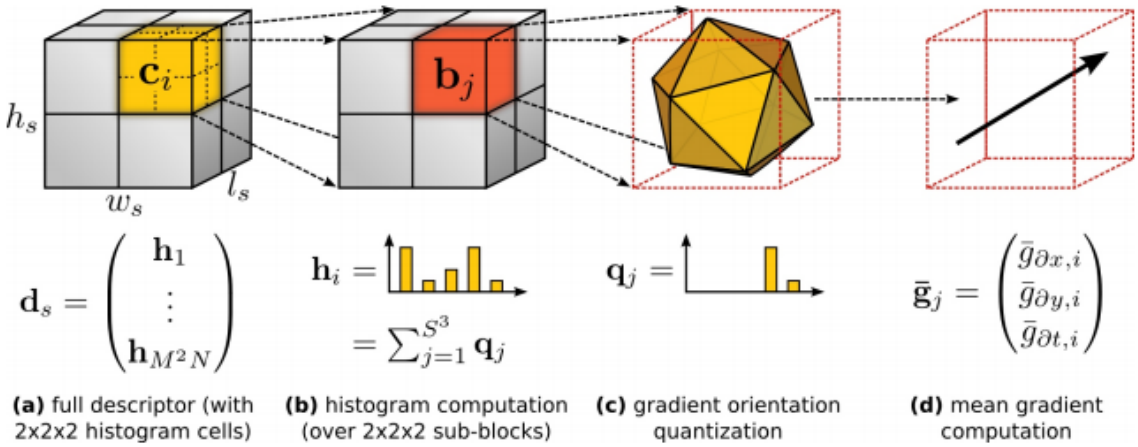


Figure 2.3: Overview of 3D-HOG descriptor computation; (a) the support region around a point of interest is divided into a grid of gradient orientation histograms; (b) each histogram is computed over a grid of mean gradients; (c) each gradient orientation is quantized using regular polyhedrons; (d) each mean gradient is computed using integral videos. Reprinted from [15].

also transform from a 2D square to a 3D cube. The overview of the descriptor computation is illustrated in Figure 2.3.

For similar reasons found in 3D-SIFT, 3D-HOG calculates the gradients in both spatial and temporal dimensions, so, the motion information contained in the time dimension can be nicely represented by the 3D-gradients.

2.2.3 Improved Dense Trajectories feature descriptor

The Improved Dense Trajectory (iDT) is a very successful algorithm used for video action recognition among all hand-crafted feature descriptors which was proposed by Wang et al. [37] [38]. Wang et al. [37] introduce the Dense Trajectory (DT) algorithm and they improve the DT algorithm by eliminating background optical flow trajectory caused by the camera motion in [38].

The overview of the DT feature descriptor is illustrated in Figure 2.4, which includes dense sampling, key points tracking and features description. The video is firstly extended into several different spatial scales in order to keep this feature descriptor scale invariant.

Feature points are densely sampled on a grid spaced by W pixels and the sampling is carried out on each spatial scale separately. Since it is hard to track feature points in homogeneous areas, Wang et el. remove those points which have very small eigenvalues of the auto-correlation matrix.

Feature points are then separately tracked on each spatial scale. Given a point $P_t = (x_t, y_t)$ in frame I_t, its tracked position in the I_{t+1} frame is calculated by the position in I_t frame and the components of optical flow computed w.r.t. I_t and I_{t+1}. Due to the fact that it is unstable to track a feature point for a long time, all the feature points are re-sampled every L frames. Then, for every feature point, the trajectory feature vector is $(P_t, P_{t+1}, P_{t+2}...., P_{t+L-1})$. The trajectory

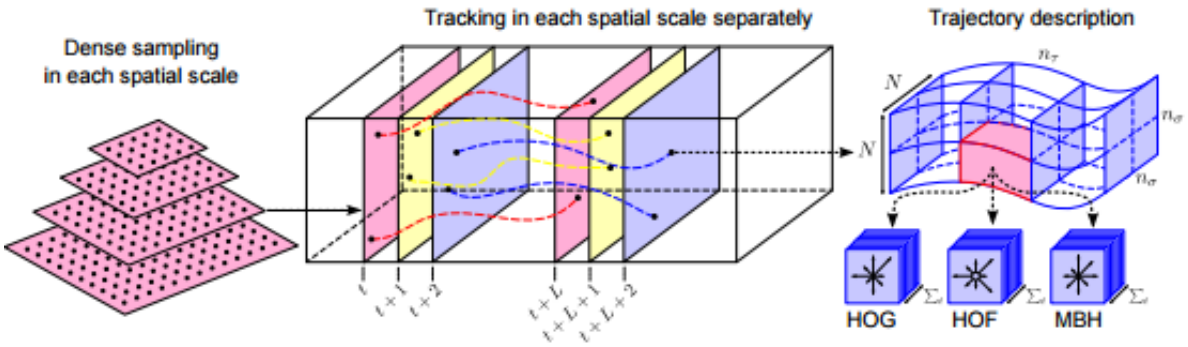


Figure 2.4: Illustration of DT algorithm to extract and characterize dense trajectories. Left: Feature points are densely sampled on a grid for each spatial scale. Middle: Tracking is carried out in the corresponding spatial scale for L frames by median filtering in a dense optical flow field. Right: The trajectory shape is represented by relative point coordinates, and the descriptors (HOG, HOF, MBH) are computed along the trajectory in a $N * N$ pixels neighbourhood, which is divided into $n\sigma * n\sigma * n\tau$ cells. Reprinted from [37].

feature vector contains the motion information of the feature points thus it can represent motion information for the videos.

Since just the trajectory features are not enough to describe all the video features, Wang et al. introduce another three feature descriptors: Histograms of Orientated Gradient (HOG [3]), Histograms of Optical Flow (HOF [5] and Motion Boundary Histograms (MBH [5]). The HOG feature descriptor is similar to 3D-HOG but it is calculated along the trajectory. The HOF features represent the optical flow (movements) of objects in videos and the MBH features are based on the derivatives of optical flow which is a simple and effective way to suppress the camera motion. 3D-HOG can well represent both the appearance/shape and the motion information for objects in videos. The combination of HOF and MBH can further improve the performance of video analysis as they represent zero-order (HOF) and first-order (MBH) motion information. All of these three feature descriptors are calculated in a $N * N * L$ spatial-temporal volume around the feature point along the trajectory, see the right image in Figure 2.4. The spatial-temporal volume is sliced into smaller $n\sigma * n\sigma * n\tau$ spatial-temporal cells. $N = 32, n\sigma = 2, n\tau = 3$ in the work of Wang et al. The histogram of HOG, HOF and MBH are calculated over all the pixels contained in each cell.

The work of iDT [38] improves the DT algorithm mainly by eliminating the camera motion. In the DT algorithm, due to the camera motion, many trajectories in the background are found. But those trajectories of background are useless for action recognition and usually giving confusing results. By assuming that the difference between 2 consecutive frames is small, the iDT algorithm assumes that the frame I_{t+1} can be calculated from frame I_t and a transform matrix H, that $I_{t+1} = I_t * H$. Then, we can calculate $Iwarp_{t+1} = H^{-1} * I_{t+1}$, where $Iwarp_{t+1}$ is the frame I_{t+1} after eliminating the camera motion. Since the transformation matrix H is calculated over

the whole image I_t and I_{t+1} which include both background and interest human. So, the large movement of the human body in consecutive frames will largely affect the accuracy of matrix H. Wang et al. use the human detection technique to detect the human body in all images and mask these areas to get a more accurate transform matrix H and use this matrix to eliminate the useless trajectory of the background and the human body that is caused by camera motion.

2.3 Deep Learning Based Feature Descriptor

Though hand-crafted feature descriptors achieve very nice performance in terms of image and video content analysis, they are all based on pre-defined rules to extract features. Thus, they usually ignore the potential cues for video analysis which are not implemented by the feature designer. A deep learning based feature descriptor has complex parameters and flexible structure. A well trained deep learning feature descriptor can represent similar features represented by hand-crafted feature descriptor. For example, the Convolutional Neural Network (CNN or ConvNet [18]) can learn about edges in lower layers which is similar to HOG. Furthermore, a well trained deep learning feature descriptor has the potential abilities to represent some features which are hard for hand-crafted features descriptors which means that learning feature descriptors are more general than hand-crafted feature descriptors. Due to the significant development of data science, deep learning based feature descriptors, especially CNN, show powerful ability for feature representation and have achieved state-of-the-art performance in image classification/recognition [9].

The Convolutional Neural Network is one type of deep artificial neural network in which the connectivity patterns between its neurons is inspired by the organization of animal's visual cortex. The architecture of one of the very first ConvNets, the LeNet5 by Yann LeCun et al. [18], is illustrated in Figure 2.5. A typical ConvNet includes three main parts: the convolutional layers, the pooling/subsampling layers and the fully connected layers. The convolutional layer extracts features from input image by applying convolution. The convolution can be understood as a feature filter applied over the input image. We can perform operations such as edge detection by just setting different filter parameters. So, one filter (convolutional kernel) can extract one type of features. The more filters, the more features that can be represented by the ConvNet. An intuitive example of convolution is illustrated in Figure 2.6. The pooling/subsampling layer reduces the dimensionality of each feature map but retain the most important information. There are different types of pooling: Max pooling which takes the largest element in a $n \times n$ window; and Average pooling which takes the average value of all elements in the window. The pooling operation can simplify the features and thus has the following advantages: 1). Making the network smaller and more manageable; 2). Reducing the number of parameters and computation, therefore controlling over-fitting; 3). Making the network invariant to small transformations, distortions, etc. The fully connected layer is a traditional multi layer perceptron in which every

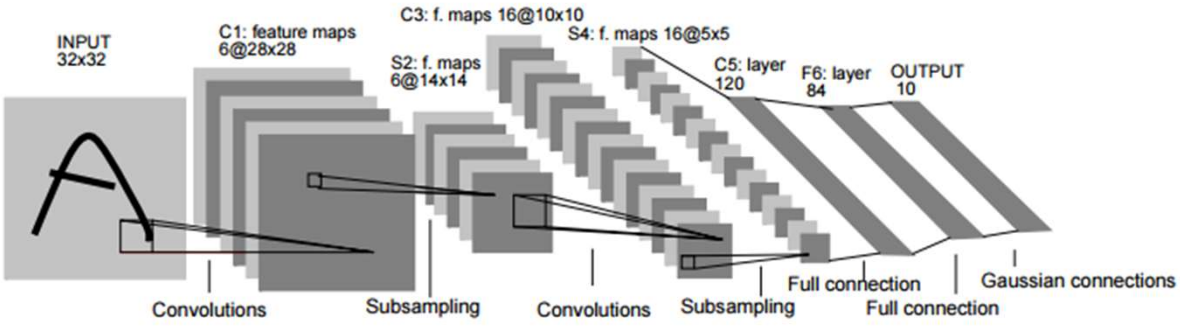


Figure 2.5: The architecture of a CNN example: LeNet5. Reprinted from [18].

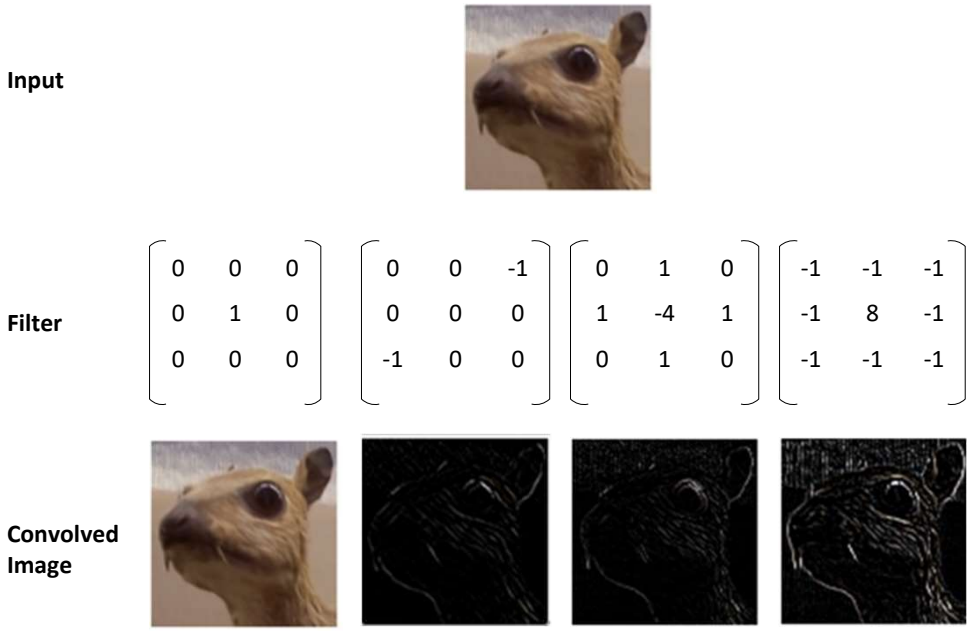


Figure 2.6: The intuitive illustration of convolution over image.

neuron in the previous layer is connected to every neuron in the next layer. The output of the convolutional and pooling layers represents high-level features of input images while the fully connected layers use these features to classify the input images into various classes.

There is related work [24] in the earlier stages which just simply applies CNN and treats the videos as a set of single frames, then averages the results over all frames as the overall result. The performance of such method is not very satisfying since it doesn't take the important temporal information into consideration and there may be many irrelevant frames in the video will confuse the result. In recent years, many new related works have been proposed in order

to solve these problems, like the Spatial-Temporal CNN[13], Two-Stream ConvNet[31], and 3D Convolutional Networks(3D-ConvNet) [1] [12] [34].

2.3.1 Spatial-Temporal CNNs feature descriptor

Since spatial information represents the appearance and the temporal information represents motion, they are both essential for video analysis. Karpathy et al. suggested a spatial-temporal CNN in their work[13]. They studied several approaches to extend the connectivity of CNNs to time domain in order to make use of the spatial-temporal information contained in videos, including late fusion, early fusion and slow fusion. The architectures of fusion CNNs are illustrated in Figure 2.7.

Single frame architecture is used as a baseline in the work [13]. It is a standard CNN framework and is similar to the ImageNet challenge winning model [17] just with a different input image resolution. The architecture of the single frame network is illustrated in Figure 2.7 (a).

The late fusion model, illustrated in Figure 2.7 (b), places two separate single frame networks that share the network parameters. The inputs of the two networks are two streams which have a distance of 15 frames. The two networks are merged in the first fully connected layer. Neither single network alone can detect any motion, but the first full connected layer can compute global motion characteristics by comparing the outputs of the two networks.

The early fusion model, illustrated in Figure 2.7 (c), is similar to the single frame model except that the input is consecutive T frames instead of a single frame. T was set to 10 in the work [13]. The early and direct connectivity to pixel data allows the network to precisely detect direction and speed of the local motion.

The slow fusion model, illustrated in Figure 2.7 (d), combines the early and late fusion model. This model slowly fuses the temporal information throughout the network so that higher layers get access to progressively more global information in both the spatial and temporal dimensions.

2.3.2 Two-Stream ConvNet feature descriptor

The Two-Stream ConvNet proposed by Simonyan et al. [31] is another extension of ConvNet for action recognition in video data. The Two-Stream ConvNet introduces a different architecture with the Spatial-Temporal CNNs [13] based on two separate recognition streams (spatial and temporal), which are combined through late fusion. The spatial stream learns the appearance features from the still video frames while the temporal stream learns the motion features in the form of the dense optical flow of input videos. So, the combination of the spatial and temporal streams can represent both the appearance/shape and motion for the input videos. The architecture of Two Stream ConvNet is illustrated in Figure 2.8.

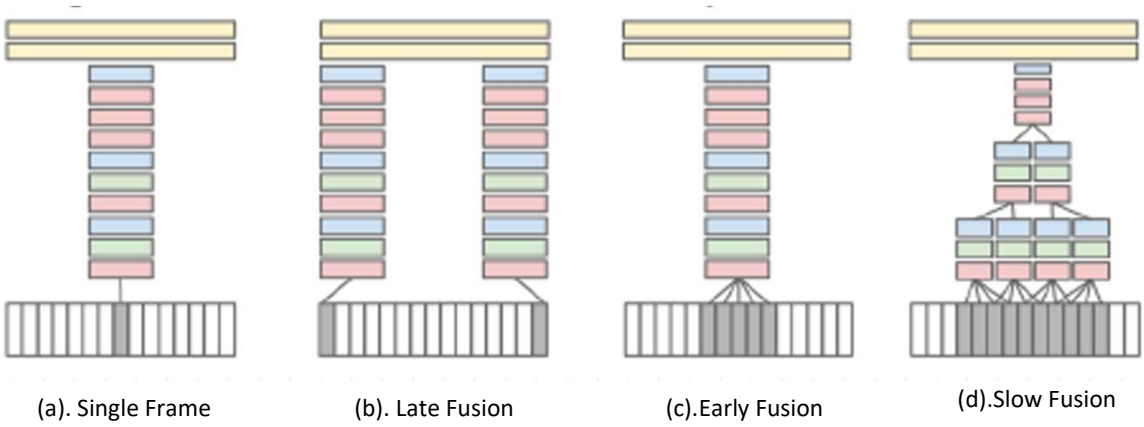


Figure 2.7: Explored approaches for fusing information over temporal dimension through the network. Red, green and blue boxes indicate convolutional, normalization and pooling layers respectively. In the Slow Fusion model, the depicted columns share parameters. Reprinted from [13].

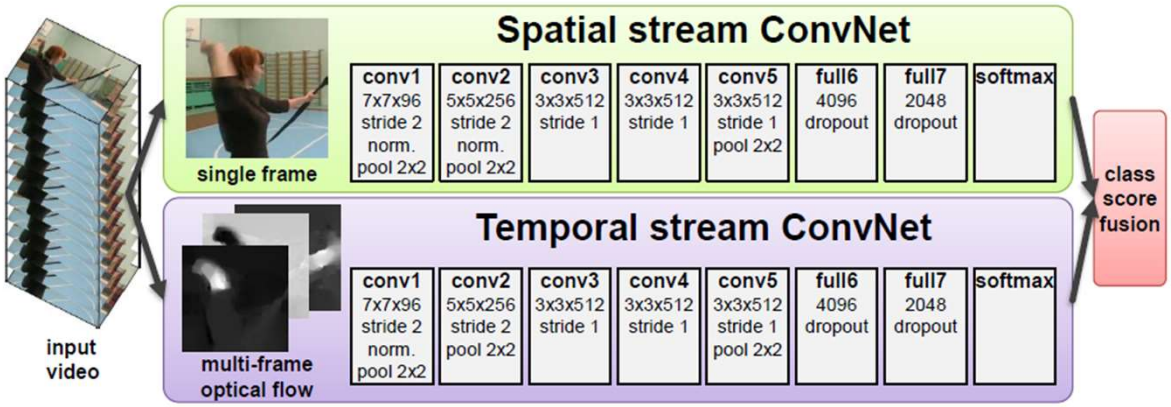


Figure 2.8: The architecture of Two Stream ConvNet. Reprinted from [31].

2.3.3 3D ConvNet feature descriptor

3D ConvNet was first proposed by Ji et al. [12] with human body segmented video volumes as input and Tran et al. [34] improved it by making it accept full raw video volumes as input without any pre-processing. The main difference between 2D ConvNets and 3D ConvNets is that the convolution and pooling are applied in three dimensions (x,y,$time$) instead of two (x,y). As illustrated in Figure 2.9, 2D convolution applied on a single image or a video volume (multiple frames as multiple channels) all result in an image, while 3D Convolution on a video volume results in another video volume. So, the video temporal information is lost after applying 2D convolution, while the 3D convolution preserves both the spatial and temporal information.

The network architecture used in [34] annotated as C3D is illustrated in Figure 2.10. The

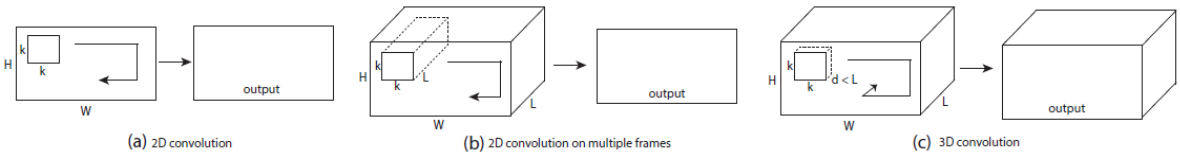


Figure 2.9: 2D and 3D convolution operations. a) Applying 2D convolution on an image results in an image. b) Applying 2D convolution on a video volume (multiple frames as multiple channels) also results in an image. c) Applying 3D convolution on a video volume results in another volume, preserving temporal information of the input signal. Reprinted from [34].

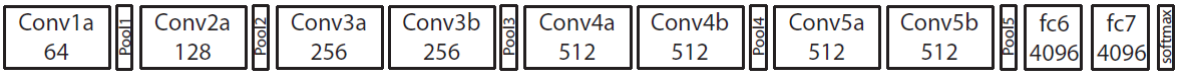


Figure 2.10: The architecture of C3D. C3D has 8 convolution, 5 max-pooling, and 2 full connected layers, followed by a softmax output layer. All 3D convolution kernels are $3 \times 3 \times 3$ with stride 1 in both spatial and temporal dimensions. Number of filters are denoted in each box. The 3D pooling layers are denoted from pool1 to pool5. All pooling kernels are $2 \times 2 \times 2$ except for pool1 is $1 \times 2 \times 2$. Each fully connected layer has 4096 output units. Reprinted from [34].

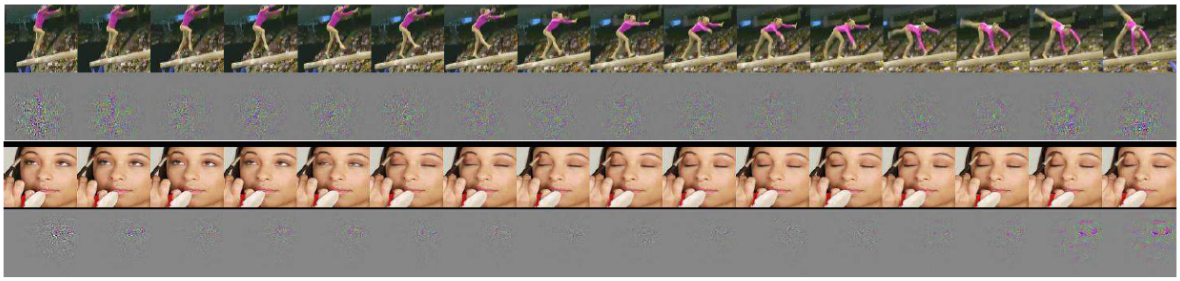


Figure 2.11: Visualization of C3D model. Interestingly, C3D captures appearance for the first few frames but thereafter only attends to salient motion. Reprinted from [34].

input of the network is a 16 frames video volume. Videos with frame number larger than 16 will be split into several 16 frames video clips with an overlap of 8 frames for each video clip. Features are calculated separately for all video clips, and the features of video clips which belong to the same video are simply averaged to generate the finally features of the video.

The Figure 2.11 illustrates what the 3D ConvNet learns by using the de-convolution method explained in [40]. In both examples, the features first focus on the appearance and gradually track the motion over the rest of the frames.

2.4 Datasets

Training dataset is one of the key factors needed to train a high-performance feature descriptor. At the same time, published and widely used datasets can be helpful for comparing between

different approaches. After the publication of the KTH [29] dataset which contains 6 action classes and 600 videos (2391 sequences) in 2004, more and more human activity datasets have been published. In recent years, the development of datasets has the following characteristics:

1. More action classes. KTH has 6 classes, UCF101 [32] has 101 action classes and ASLAN [25] has 432 classes.

2. More training and testing samples. KTH has 192 videos for training and 216 videos for testing, while ActivityNet 200 [10] has 10024 videos for training and 5044 videos for testing.

3. The video scenes become more and more complex. The videos of KTH are acted by very limited number of actors, while recent datasets are cut from realistic scenes, like youtube, cctv, BBC etc.

4. More challenges in the video content; from fixed backgrounded without camera motion to non-static camera, multi-viewpoints, more complex backgrounds; from single person action to human-human interaction, human-object interaction, etc.

2.4.1 List of human activity video datasets

Part of the widely used action and interaction video datasets are illustrated in Table 2.1 and Table 2.2, respectively.

2.4.2 UT-Interaction dataset

We use UT-Interaction dataset [28] as the main target interaction dataset to train and evaluate our interaction classification and detection model. We first introduce the basic information of this dataset followed by some special challenges about it and the last for the evaluation methodology.

Introductions of the basic information of the UT-Interaction dataset

The UT-Interaction dataset contains videos of continuously executions of 6 classes of human-human interactions, including hand shaking, hugging, kicking, pointing, punching and pushing, as shown in Figure 2.12.

There are 20 video sequences whit about 1 minute lengths for each. Each video contains at least one execution per interaction, providing us 6 executions of human-human interactions per video on average. Several participants with more than 15 different clothing conditions appear in the videos. The videos are taken with the resolution of 720×480, and the height of a person in the video is about 200 pixels. Ground truth labels of interaction are provided, including time intervals and bounding boxes.

These 20 videos are divided into two sets. The set1 is composed of 10 video sequences taken on a paring lot. The videos of the set1 are taken with slight different zoom rate, and

Table 2.1: List of human action video datasets

Dataset	Classes	Videos	Annotations	Properties
KTH[29]	6 action classes	• Training : 192 • Validation: 192 • Testing: 216 • Resolution: 160 × 120 @ 25fps	• Action labels • Temporal segments	• Static camera • Simple background • Acted by 25 subjects, 6 actions and 4 scenarios
Hollywood2[21]	12 action classes and 10 scene classes	For actions: • Training : 823 • Testing: 884 For scenarios: • Training : 570 • Testing: 582 Resolution: 400 – 300 × 300 – 200	• Action labels • Scene labels	• Non-static camera • Realistic scenarios from movies
UCF101[32]	5 types, 101 action classes	• Training : 9537 • Testing: 3783 • Resolution: 400 – 300 × 300 – 200	• Action labels	• Realistic scenarios from YouTube • Large variations in camera motion, object appearance and pose, object scale, viewpoints, cluttered background, illumination conditions, etc.
ActivityNet 200[10]	200 action classes	• Training : 10024 • Validation: 4926 • Testing: 5044	• Action labels • Temporal segments • Hierarchy activities relationship	• Realistic scenarios • Large variations in camera motion, object appearance and pose, object scale, viewpoints, cluttered background, illumination conditions, etc.

Table 2.2: List of human interaction video datasets

Dataset	Classes	Videos	Annotations	Properties
UT-Interaction[28]	6 interaction classes	• 20 videos • 2 sets, 10 for each set. Evaluated by leave one out cross validation. • Resolution = 720 × 480	• Interaction labels • Temporal segments • Spatial segments	• Two-person interaction • Static camera • Acted in different background
ShakeFive2[36]	8 interaction classes	• 153 videos • Resolution = 1280 × 720	• Interaction labels • Joint position • Temporal segments	• Homogeneous background • Static camera
Multi-modal & Multi-view & Interactive [39]	9 interaction classes and 13 person-object interaction classes	• 1760 RGB videos • 1760 depth videos • Resolution = 320 × 240	• Interaction labels • Joint position • Foreground mask	• Homogeneous background • Static camera

their backgrounds are mostly static with little camera jitter. While the set2 (the other 10 video sequences) are taken on a lawn in a windy day. The backgrounds are moving slightly, for example tree moving, and they contain more camera jitters. The illustrations of the different backgrounds between the two sets are shown in Figure 2.13.

The evaluation methodology

We evaluate our models with two different experimental settings: classification and detection.

For the 'classification ' task, the two sets of videos are evaluated separately. There are 120 video segments which are cropped based on the ground truth and each contains exactly one execution of one type of interaction. So, for each set, we have 60 segmented videos that will be used for the training and testing. The cropped video segments are illustrated in Figure. 10-fold

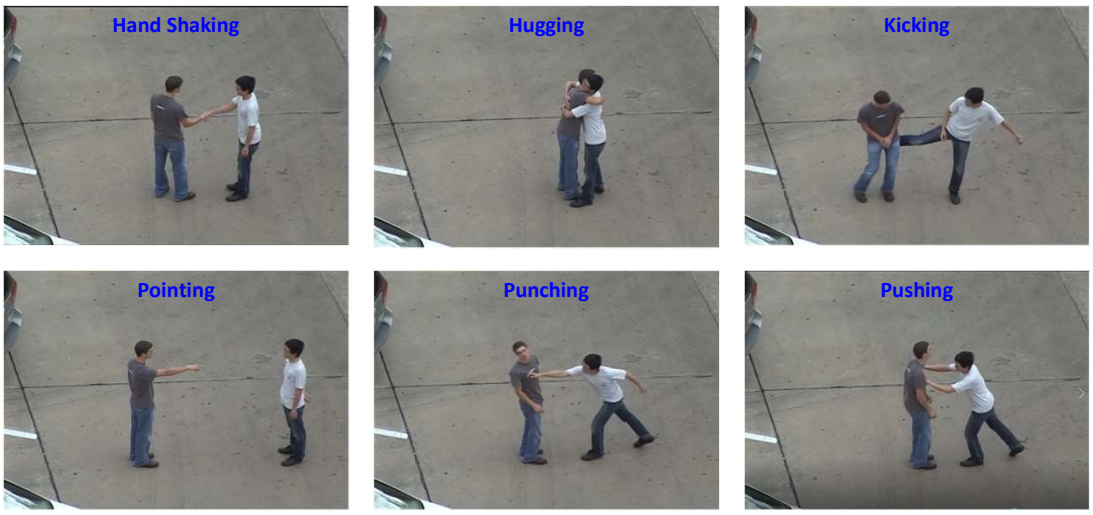


Figure 2.12: The illustration of the 6 classes of human-human interactions in the UT-Interaction dataset.

leave-one-out cross validation is employed to evaluate the classification accuracy. That is, for each set, we leave one among 10 sequences for the testing and use the other 9 for the training. The performance is measured ten times and the average accuracy is used as the overall accuracy.

$$\text{Classification Accuracy} = \frac{\sum_{i=1}^{10} Accuracy_i}{10}$$

Where $Accuracy_i$ is the classification accuracy of the ith sequence.

For the 'detection' task, the interaction detection is measured to be correct if and only if the network correctly annotates an occurring interaction's time interval and spatial bounding box. If the detection overlaps the ground truth more than 50% spatially and temporally, the detection is treated as a true positive. Otherwise, it is treated as a false positive. i.e.

$$\frac{\text{Detection} \cap \text{Ground Truth}}{\text{Detection} \cup \text{Ground Truth}} > 50\%$$

The 'Detection ' here is a 3D volume, its label, location and size are given by the output of the interaction detector. And 'Ground Truth ' here is also a 3D volume which has the same class label with the 'detection '. The location and size of the 'Ground Truth ' are provided by the ground truth table. The illustration of 'Detection ' and 'Ground Truth ' is shown in Figure 2.15.

We use general Precision-Recall graph to evaluate our detection model. Where precision and recall are defined as below:

$$\text{Precision} = \frac{\text{The number of correct detections}}{\text{The number of founded detections}}$$

$$\text{Recall} = \frac{\text{The number of correct detections}}{\text{The number of GT detections}}$$

a). Some frames randomly sampled from the 10 video sequences of set1

b). Some frames randomly sampled from the 10 video sequences of set2

Figure 2.13: The illustration of the different backgrounds between the two sets

a). Some frames randomly sampled from the 60 video segments of set1

b). Some frames randomly sampled from the 60 video segments of set2

Figure 2.14: Some frames randomly sampled from the 120 segmented videos.

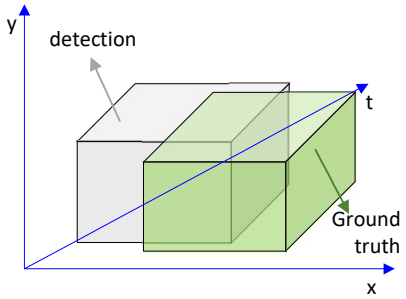


Figure 2.15: Illustration of detection and ground truth.

The challenges

Comparing with image analysis focus on still images, we are expected to recognize ongoing complex human activities from continuous videos, taken in realistic settings. There are some challenges for the classification task as below:

1. The backgrounds of set1 and set2 are different illustrated in Figure 2.16 a). This requires the classifier focuses on foregrounds and ignore the influences of different backgrounds.

2. The background is not still even in a same video segment which is illustrated in Figure 2.16 b). This may causes troubles for the classifier to learn to discriminate between the background and the foreground.

3. The resolutions of different video segments are different which is illustrated in Figure 2.16 c). The classifier usually requires all its inputs with the unified size. we can unify the sizes for them by cropping and resizing. But, the width-height ratio and scale for each interaction may still be different.

4. The time duration for each interaction execution in different video segments varies much from 30 frames to 154 frames.

5. The lighting conditions of different video segments are various which is illustrated in Figure 2.16 d).

Compared with the classification task, there are some extra challenges for the detection task, including:

1. Both interacting people and irrelevant people are present in the scene in some videos as shown in Figure 2.17 a). This will causes troubles for interacting people detection.

2. Several pairs of interacting people execute interactions simultaneously in some videos as shown in Figure 2.17 b), but the ground truth only contains one pair interacting people for each video.

a). Example frames illustrate interactions with same class label executed in different backgrounds

b). Example frames illustrate dynamic background in a same segmented video

c). Example frames illustrate different resolutions of the video segments

d). Example frames illustrate different lighting of the video segments

Figure 2.16: Some challenges for the classification task.

a). Both interacting and Irrelevant people are present in the scene

b). Two pairs of interacting people execute interactions simultaneously

Figure 2.17: Illustrations of some extra challenges for the detection task.

CHAPTER 3

ARCHITECTURE

This project focuses on two-person human-human interaction classification and detection in videos. Almost all of the few previous works [26] [22] [2] of the two-person human-human interaction recognition adopt hand-crafted feature descriptors. For example, Narayan et al. [22] combine improved trajectory features and the foreground motion map features in order to present interaction features while Patron-Perez et al. [26] and Choi et al. [2] use a hierarchical model to present interaction features. In the hierarchical model, each individual is tracked throughout the videos, and the atomic activity for each individual and their relative position and orientations are computed first and then the collective interaction is represented. Our work adopts the hierarchical model similar to [26] but we replace the hand-crafted feature descriptors with the deep learning feature descriptors since deep learning based methods [12] [34] [31] [23] have already achieved better performance in the related domains, such as single person action recognitions in videos, than hand-crafted feature descriptors[7] [15] [30] [37] [38].

3.1 Overall Framework

Since we define two tasks including interaction classification and detection in our project and we set the classification as the basis for the detection, we will first introduce the overall framework of the interaction classification followed by the introduction of the architecture of the interaction detection.

3.1.1 Interaction classification

Due to lack of sufficient large-scale two-person interaction datasets to just train a single deep learning network, we adopt the hierarchical model similar to [2] and [26] which learn atomic

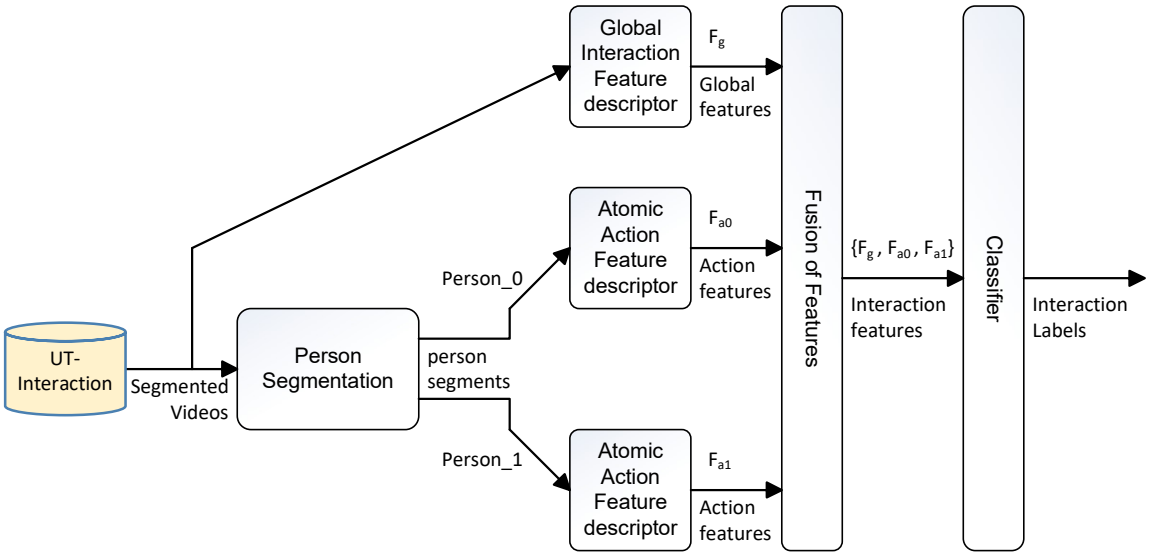


Figure 3.1: Overall framework of the interaction classification.

action features for each individual in the video and concatenate these features to represent the collective interaction features. In contrast to [2] and [26] which adopt hand-crafted features descriptors, such as HOG [3] and BoV [6], we employ the deep learning networks as the feature descriptors.

The overall framework of our interaction classifier is illustrated in Figure 3.1. In our architecture, we first apply the person detection to locate and track the position of each person in the input segmented videos of the UT-Interaction dataset, see Section 2.4.2, and then crop each video into two video segments, each of which contains one person's activity. Then we design an atomic action spatial-temporal feature descriptor for each person. Since there are lots of large scale single person action datasets available, we can pre-train the atomic action feature descriptors with those datasets, such as **the ucf101 action dataset** [32], and finetune the parameters on our target interaction dataset.

One drawback of this hierarchical model is that it ignore the important relative information between the two people, such as their relative position and orientation, etc. To fix this, we introduce another global feature descriptor to learn those relative information which can be trained directly on our target dataset **UT-Interaction**.

The features extracted by the global interaction feature descriptor (F_g) and the features extracted by the atomic action feature descriptors (F_{a0} and F_{a1}) are concatenated and fed to the Classifier.

3.1.2 Interaction detection

The overall architecture of the interaction detection is illustrated in Figure 3.2. We first perform the spatial detection of interacting people followed by the temporal detection of interaction

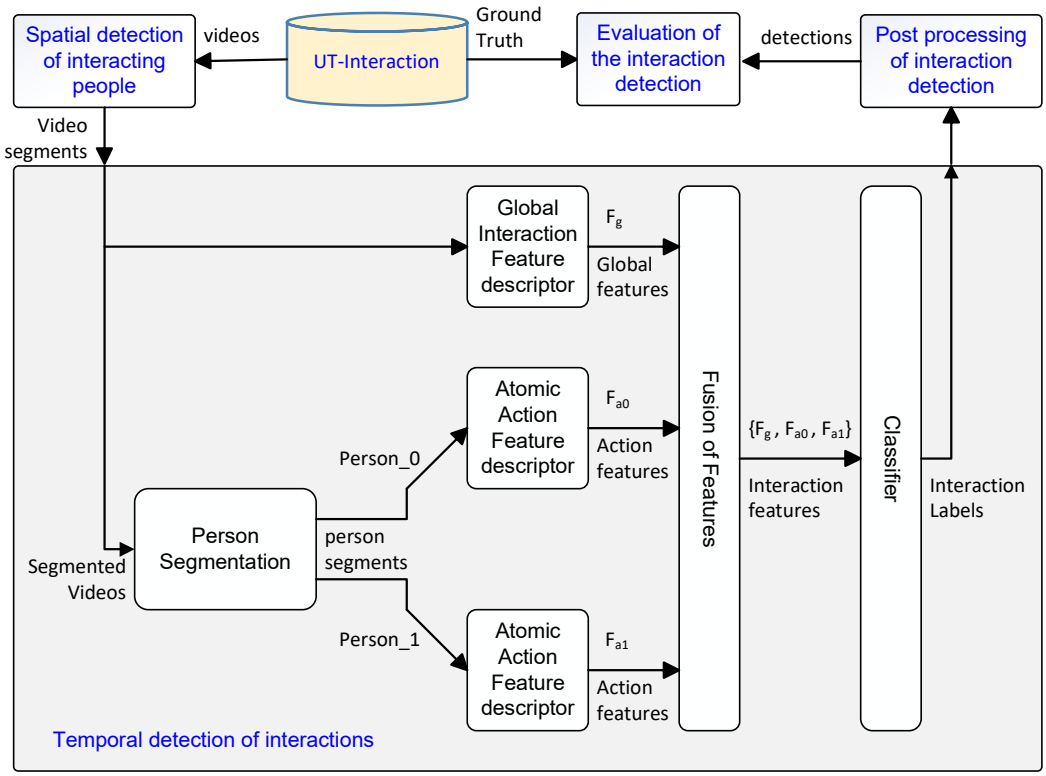


Figure 3.2: Overall framework of the interaction detection.

instead of directly applying a spatial-temporal variable size sliding window. Because the later one costs lots of computation resources and is very timing consuming. We define the spatial detection of interacting people as spatially locating the interacting people in each frame and tracking them throughout the video. Then we can make use of these spatial locations of the interacting people to generate candidate video clips by temporally sliding a 3D (x, y, t) window. Then we apply temporal detection of interactions based on the interaction classification model defined in the Section 3.1.1.

3.2 Models

We introduce the main models employed in the interaction classification and detection. Since there are many model overlaps between interaction classification and detection, we introduce the main models in flat mode rather separately introduce them for classification and detection.

3.2.1 Person Segmentation

We locate the position of each person in each frame of the input videos, and track them throughout the video. Then we can use the bounding boxes of each person to split each input video into two

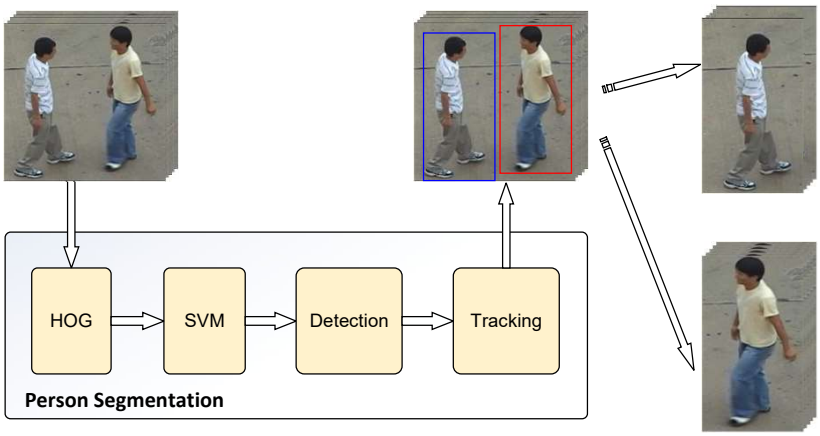


Figure 3.3: The architecture of the person segmentation

video segments. The structure of the person segmentation is illustrated in Figure 3.3. There is no special challenges to perform person detection and tracking in the videos of UT-Interaction dataset because the background is relative simple and there is no much occlusions between people. So, for simplicity, we employ the classical Histogram of Oriented Gradients (HOG) feature descriptor plus a linear Support Vector Machines (SVM) framework to construct the person detector [4]. Due to there may be some missing detections of one or two people in some frames, we employ the Kalman Filter to track each person throughout each video. Due to the fact that the relative position of the interacting people will be kept, i.e. if one of the interacting people first appear at left then he will kept at left in the whole video, we can simply match the interacting people according to their relative position.

3.2.2 Spatial detection of interacting people

For the interaction detection task, since there are both interacting and irrelevant people are present in the scene, we need to locate the positions for the interacting people and omit the irrelevant people. The overall architecture of the spatial detection of interacting people is illustrated in Figure 3.4. We adopt the same model to detect all people in each frame and to track interacting people as the person detection and tracking in the Section 3.2.1. The only difference here is that we need the extra works to locate the interacting people and omit the irrelevant people which we will introduce in detail in the next chapter 4.2.3.

3.2.3 Feature Descriptor

We employ the hierarchical feature descriptors, including a global feature descriptor which learns the global features, such as the relative position and orientation between two-person, and two atomic feature descriptors which learn the local atomic action features for each person. By doing this, our hierarchical feature descriptor is expected to learn both the hight-level complex

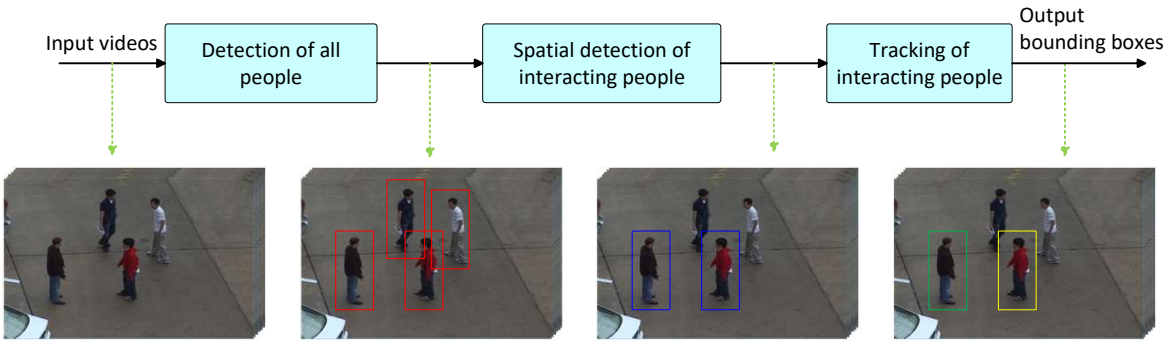


Figure 3.4: The architecture of the person segmentation

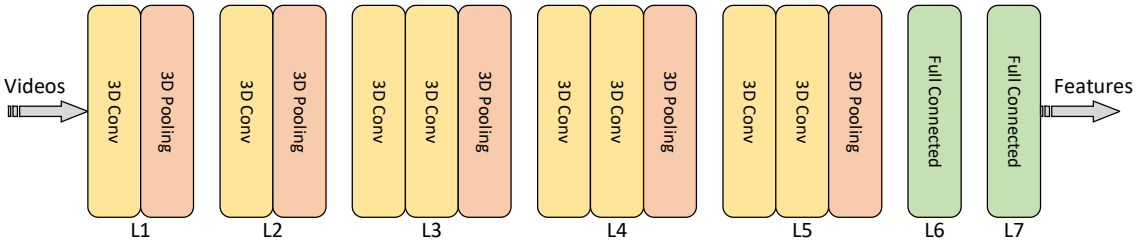


Figure 3.5: Overall Architecture of the 3D-ConvNet.

interaction features and low-level detail atomic action features. The global feature descriptor and the atomic action feature descriptors share similar network frameworks. We use different network parameters between the global feature descriptor and the atomic feature descriptors.

Our feature descriptors are all designed based on the 3D ConvNet [34]. 3D ConvNet learns both spatial and temporal features at the same time by applying three-dimensional (x,y,time) convolution and pooling. The architecture of our 3D-ConvNet based feature descriptor is similar to Tran et al.'s 3D-ConvNet[34] . The architecture of our 3D-ConvNet is illustrated in Figure 3.5.

3.3 Training

Since the performance of a learning algorithm is highly dependent on the training datasets, it is important to select proper training datasets for each network and balance the performance and computational complexity at the same time.

3.3.1 Train the person detection network

For simplicity, we directly use a pre-trained HOG plus linear SVM classifier trained on **INRIA Person Dataset** [4] as our person detector. The INRIA Person Dataset is one of the most used datasets for static people detection, which provides original images and the corresponding

annotations. The training set contains 614 (2416 people) positive images and 1218 negative images.

3.3.2 Train the feature descriptor

There is an extra class for the interaction detection task because the video clips which contain no interaction need to be annotated an extra label. So, we train the feature descriptor for the interaction classification and the detection with different training set.

Train the feature descriptor for the interaction classification task

We adopt different training strategies between the global feature descriptor and atomic action feature descriptors.

For the global feature descriptor, we train it directly on the training data of the **UT-Interaction** dataset, see 2.4.2. Before feeding training data to optimize the parameters, we first resize all segmented videos to 128×144 pixels; randomly sampling a 112×128 region; augment the training videos by horizontal flipping and multi-time random cropping; temporally down-sample the frames and finally split each input video into several 16-frame video clips with 8 frames overlap. We use the adaptive moment estimation (ADAM) to optimize our models. We use mini-batch of 16 examples (video clips) and initial learning rate of $1e^(-4)$ which is further divided by 2 for every 4 training epochs.

For the atomic action feature descriptors, we will test both the pre-training plus fine-tuning strategy and the directly training strategy. Since we have large scale single person action datasets available, we will observe whether the pre-trained network can further improve the performance. For the pre-training plus fine-tuning strategy, large scale action video dataset **UCF-101** [32] will be used to pre-train the atomic action feature descriptor and the video segments generated by the person segmentation, see 3.2.1, will be used to fine-tune it. For the directly training strategy, we train the network directly on the video segments generated by the person segmentation. We adopt the same method of augmentation and pre-processing which we have applied to train the global feature descriptor except that the final resolution of the video clips is 112×80 instead of 112×128. The method of optimization is also as same as that of the global feature descriptor.

Train the feature descriptor for the interaction detection task

We adopt the same strategy as the classification task to train the feature descriptor for the detection task except that the some negative training videos are added to the training set and there are 7 class labels instead of 6. Those negative training videos are generated by randomly cropping in the unsegmented videos, see 2.4.2, and for each that does not overlap with any positive bounding boxes, see 4.2.5 for detail.

3.3.3 Train the Classifier

All the output features of the global feature descriptor F_g and the atomic action feature descriptors F_{a0} and F_{a1} are concatenated as $\{F_g, F_{a0}, F_{a1}\}$ and fed to the softmax classifier. These features are extract from the target dataset **UT-Interaction**.

CHAPTER 4

DESIGN

In this chapter, we present the designs and their theories in detail. We use the video interaction classifier as a part of video interaction detector, thus, we will first introduce the methodologies of the video interaction classifier followed by the video interaction detection. We present our approaches in top-down style and along the data-flow.

4.1 Interaction Classification

We present the overall data-flow and low-level designs of the interaction classifier. We start off with the overall data-flow, then present how we pre-process the training data; how we design the feature descriptor, the classifier and the optimizer.

4.1.1 Data-flow of the Interaction Classification

The overall data-flow of the interaction classifier is illustrated in Figure 4.1. We use the segmented videos of the UT-Interaction dataset, see Section 2.4.2, to train and test our interaction classifier in the classification scenario. The data is first sent to the data pre-processing module which is designed to split the dataset into training and testing samples; augment the training data and convert the videos to video clips with the standard shape which can be accepted by our feature descriptor. In the off-line training phase, the feature descriptor extracts features from the input video clips, the classifier predicts the class labels according to the extracted features, and the optimizer optimize the parameters of feature descriptor and classifier according to the loss between the predicted labels and the given labels. In the on-line testing phase, the parameters learned in the off-line training phase are loaded into feature descriptor and classifier. The

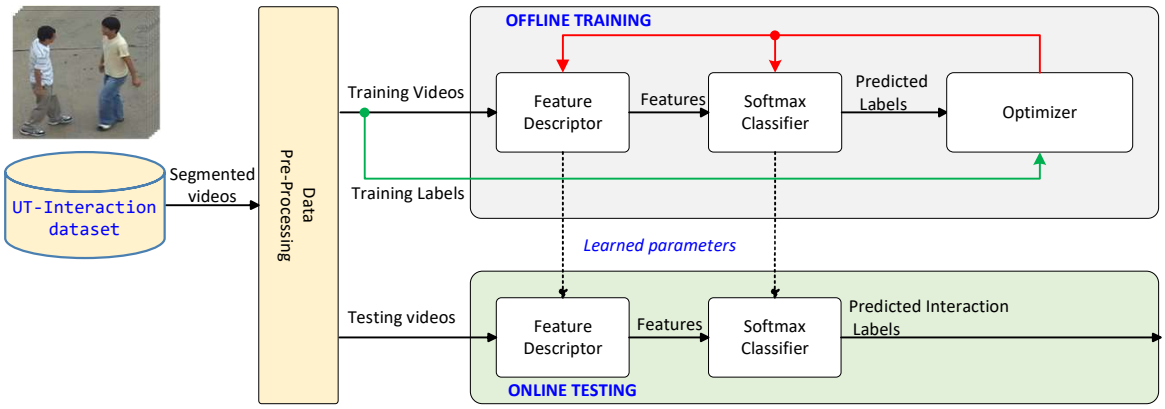


Figure 4.1: The overall data-flow of the interaction classification

classifier predicts the interaction labels for the input videos according to the features extracted by the feature descriptor.

4.1.2 Data pre-processing

The input videos of the global interaction feature descriptor contain two interacting people while the input videos of the atomic action feature descriptors contain only one of the two interacting people. So, the data pre-processing module need to generate different type of training and testing videos according to their corresponding feature descriptors.

The overall data-flow of data pre-processing module is illustrated in Figure 4.2. The person segmentation module is designed to detect and track each person in the interaction videos and to segment each video into two videos which contain only one person in each video. There are three low-level data pre-processing modules, one for global interaction feature descriptor and the other two for the atomic action feature descriptors. These three low-level feature descriptors are synced to each other in order to ensure that they are processing the same video at the same time. The low-level data pre-processing module first splits the input videos into training set and testing set, then augments the training data by horizontal flipping and random chopping since the UT-Interaction dataset is a relative small scale interaction dataset for training a deep learning network, and finally resizes and clips the augmented videos into the video clips with proper size which can be accepted by the corresponding feature descriptors.

Person Segmentation

The tracking-by-detection methodology is employed in the person segmentation module. We first detect all people in each frame with a classical Histogram of Orientated Gradient (HOG) feature descriptor plus a linear support vector machine (SVM) classifier [3]. Then a Kalman filter is applied to track each person throughout a video. Since there are two and only two interacting

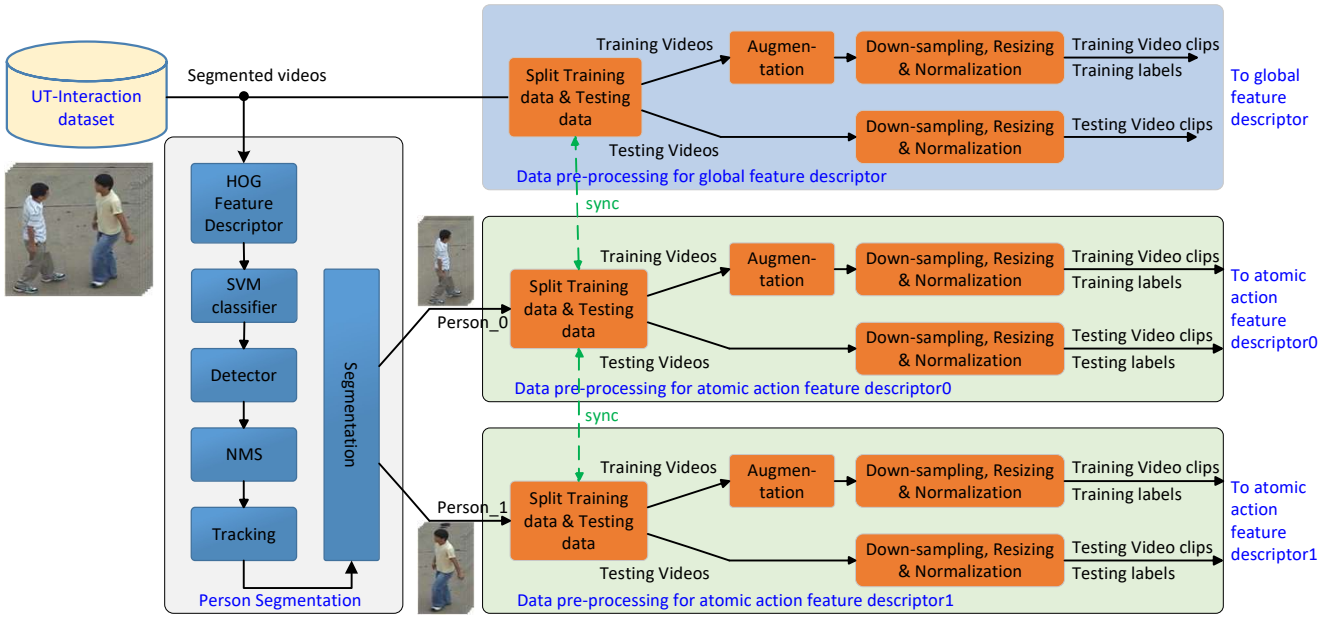


Figure 4.2: The overall data-flow of the data pre-processing module

people in the input videos, we get two-bounding boxes in each frame after tracking. And finally, we chop the video according to the bounding boxes and generate two segmented videos from each input video.

The HOG feature descriptor The HOG feature descriptor is widely used to represent the appearance and shape of objects by calculating and collecting the gradients. The steps of generating HOG features include:

1. Convert the input colourful image to gray scale.

2. Calculate the gradients of each pixel. Specifically, the gradients for a pixel point at (x, y) is calculated with following formulas:

$$G_x(x,y) = H(x+1,y) - H(x-1,y) \tag{4.1}$$

$$G_y(x,y) = H(x,y+1) - H(x,y-1) \tag{4.2}$$

$$G(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2)} \tag{4.3}$$

$$\alpha(x,y) = tan^{-1}(\frac{G_y(x,y)}{G_x(x,y)}) \tag{4.4}$$

Where $G_x(x,y)$ and $G_y(x,y)$ are the horizontal and vertical gradients respectively and $H(x,y)$ is the pixel value of the point (x,y). $G(x,y)$ and $\alpha(x,y)$ are the amplitude and orientation of the gradient of the pixel point (x,y), respectively.

3. Compute the histogram of gradients in the cell unit. A cell is usually composed of 6×6 pixels and we use 9 bins to compute the histogram for these 36 pixels. The value of orientation determines which bin need to add and the value of amplitude determines the weight of adding.

4. Normalize the histogram in a bigger block unit. A block is usually composed of 3×3 neighbor cells by doing which we can further eliminate the effects of different lighting and contrast of the raw images. There are $3 \times 3 \times 9$ features for each block.

5. Slide the block horizontally and vertically with stride $j = 1\ cell$, we get many blocks and their features x_i. All those features are combined together as $\{x_0, x_1, ..., x_n\}$ as the finally HOG features of an image.

The linear SVM classifier In our project, we use a linear SVM to determine whether the input image is a person or not according to the input features extracted by the HOG feature descriptor. The object of training such a liner classifier is to find a hyper-plane which is defined as:

$$w^T x + b = 0 \tag{4.5}$$

Where x is the input features, w and b are the weights and biases respectively. We get the optimal values of w and b by training the classifier on the INRIA dataset [4]. Then we use $h_{w,b}(x) = g(w^T x + b)$ to predict whether the input is a person or not. The input is predict as a person if $h_{w,b}(x) \geq 0$, and vice versa.

Person detection A sliding window is used to detect the locations of the interacting people in each frame. A size variable window slides over the whole image with stride $(4, 4)$, and the pixels in the sliding window will be converted to HOG features and predicted by the SVM classifier. For the purpose of scale invariant, we apply the detection in multi-scale image pyramid. The multi-scale image pyramid is composed of the images with different scales with the below formula.

$$img\_cols_{i+1}, img\_rows_{i+1} = \frac{img\_cols_i}{scale}, \frac{img\_rows_i}{scale} \tag{4.6}$$

Where img_cols_i and img_rows_i are the width and height, respectively, of the image in the ith layer of the image pyramid, and $scale = 1.03$ controls the shape and the number of layers of the image pyramid. The location, window size as well as the confidence will be recorded if it is predicted as a person. Since there are many overlapping windows after the whole detection process, we use non-maximum-suppression (NMS) to eliminate the overlapping windows and find out those windows with maximum confidence. The overlap threshold is set to 0.65 in the project.

Tracking of people The detection of one or two person may loss in some frames. So, we employ the second order Kalman filter to track the location of each person throughout the whole videos. There are exactly two interacting people to track in our project, so, we have two independent Kalman filters. Instead of tracking a whole bounding box, we use the Kalman filter to track the left-top point location of each bounding box, and reconstruct bounding boxes with the filtered left-top point locations plus the average width and height of all the bounding boxes in the video.

Another special point in our project is that the relative left-right position between the two interacting people will keep unchanged. That is, if the person A first present at the left side of the two interacting people, then he will always be at the left side throughout the whole video. So, we do not need to specially design a algorithm to match the people when tracking but just simply assign the left bounding box to person A and the right one to person B. The detail design of the Kalman filter for a single person is introduce as below.

The system state of each person includes position, velocity, and acceleration. We do not use acceleration in our Kalman filter since it varies very much when people are interacting. We notate the position at time t as $p(t) = (x(t), y(t))$ and the system state at time t is $x(t) = [p(t), v(t)]$. Then we implement the Kalman filter with following steps:

1. Initialize the Kalman filter.

$$\begin{aligned} t &\leftarrow 0 \\ Q(0) &= I_4 * 10^{-6} \\ R(0) &= I_4 * 10^{0} \end{aligned}$$

   Where I_4 is a 4×4 identity matrix. $Q(0)$ and $R(0)$ are the initial value of the noise covariance matrices for estimation and measurement, respectively.

2. Estimate the system states at time t according to the optimal value of the system state at time $t-1$.

$$\begin{bmatrix} p(t|t-1) \\ v(t|t-1) \end{bmatrix} = \begin{bmatrix} I_2 & I_2 \\ O_2 & I_2 \end{bmatrix} \begin{bmatrix} p(t-1|t-1) \\ v(t-1|t-1) \end{bmatrix} + w(t) \tag{4.7}$$

   Where I_2 and O_2 are 2×2 identity and null matrices, respectively. $w(t)$ represents the uncertainty of the estimation, and is assumed to be a random variable with a zero mean and its covariance matrix $Q(t) = E[w(t)w^T(t)]$.

3. Update the covariance matrix for the estimation of the system states at time t according to the covariance matrix of the system sate at time $t-1$.

$$P(t|t-1) = \begin{bmatrix} I_2 & I_2 \\ O_2 & I_2 \end{bmatrix} P(t-1) + Q(t) \tag{4.8}$$

   Where $P(t|t-1)$ is the covariance matrix of the estimation at time t, and $P(t-1)$ is the covariance matrix of the optimal system states at time $t-1$.

4. Measure the system states at time t.

$$\begin{bmatrix} \bar{p}(t) \\ \bar{v}(t) \end{bmatrix} = \begin{bmatrix} I_2 & O_2 \end{bmatrix} \begin{bmatrix} \bar{p}(t) \\ \bar{p}(t) - p(t-1|t-1) \end{bmatrix} + m(t) \tag{4.9}$$

Where $\bar{p}(t)$, $\bar{v}(t)$ are the measured system states at time t. If the detection of the person is missing in some frames, we use the measured values at the time $t-1$ as the measure value at time t. $m(t)$ is the uncertainty of the measurement, and is assumed to be a random variable with a zero mean and its covariance matrix $R(t) = E[m(t)m^T(t)]$.

5. Update the Kalman gain at time t according to the covariance matrix of the estimation of the system states at time t.

$$Kg(t) = \frac{P(t|t-1)H^T}{HP(t|t-1)H^T + R(t)} \tag{4.10}$$

$$H = \begin{bmatrix} I_2 & O_2 \\ O_2 & I_2 \end{bmatrix} \tag{4.11}$$

Where $Kg(t)$ is the Kalman gain at time t.

6. Calculate the optimal system states according to the estimation, measurement and the value of Kalman gain.

$$\begin{bmatrix} p(t|t) \\ v(t|t) \end{bmatrix} = \begin{bmatrix} p(t|t-1) \\ v(t|t-1) \end{bmatrix} + Kg(t) \left(\begin{bmatrix} \bar{p}(t) \\ \bar{v}(t) \end{bmatrix} - \begin{bmatrix} I_2 & O_2 \\ O_2 & I_2 \end{bmatrix} \begin{bmatrix} p(t|t-1) \\ v(t|t-1) \end{bmatrix} \right) \tag{4.12}$$

7. Update the covariance matrix for the optimal system states at time t according to the Kalman gain and covariance matrix of the estimated system sate at time t.

$$P(t|t) = \left(1 - Kg(t) \begin{bmatrix} I_2 & O_2 \\ O_2 & I_2 \end{bmatrix} \right) P(t|t-1) \tag{4.13}$$

8. Repeat above steps from 2 to 7 at time $t+1$.

We generate a bounding box for each person in each frame after tracking, and the width and height of all bounding boxes are the same because we replace the width and height of each bounding box with the average value of the width and heigh of all bounding boxes in a video based on a reasonable assumption that every people appeared in the video have similar width and height. Finally, we crop the interaction video according the bounding boxes and generate two segmented videos which contain one person in each. See the inputs and outputs of the Person Segmentation module in the Figure 4.2.

Data pre-processing

The functions of data pre-processing include: splitting of training and testing data, augmentation of training data and converting the training and testing videos to proper size which can be accepted by the corresponding feature descriptors. The general steps of data pre-processing are:

1. The splitting of training and testing data. Since UT-Interaction is a relative small scale interaction video dataset, we use 10-fold leave-one-out cross validation to evaluate the performance. There are 10 sequences in each set, and 6 videos in each sequence. So, we leave one sequence for testing and other nine sequences for training in each run. See 2.4.2 for detail.

2. Augmentation of training data. After the splitting of training and testing videos, we get 56 videos for training and 6 videos for testing in each run. But, 56 videos is not sufficient to train our deep learning network. So, we employ two methods, horizontal flipping and random chopping, to augment the training videos. Horizontal Flipping is used to generate a new video by horizontally flipping each frame from the original video which is illustrated in Figure 4.3. We further enlarge the scale of the training set n times by N-time random cropping. In the random cropping module, we chop the video by a window with fixed size and random position of left-top point in a reasonable range. Specifically, in our project, we first resize each training video to $l \times 128 \times 144$, where l is the number of frames for each video, then random chop each video with size $l \times 112 \times 128$. An example of the random cropping is illustrated in Figure 4.4.

3. Down sampling. The number of frames of each individual interaction video is usually in the range from 60 to 150 frames, but our feature descriptor can only accept the video clips with only 16 frames due to the limitation of GPU memory and computation time. If we directly chop each video into 16-frame video clips, then there will be only very limited temporal information contained in each video clip. So, we first down-sample the video with N frames stride to make each video clip contains more temporal information. Besides the fixed stride temporal down-sampling, we have an additional temporal down-sampling mode, that is, we sample exactly 16 frames from each input video with equal distances if the number of frames of the input video is larger than 16 by setting $N = 0$.

4. Resizing. The number of frames for each video may be still larger than 16 after down sampling of frames, then we chop them to many 16-frame video clips with 8 frames overlapping.

5. Normalization. Since our training data are videos, the most common method of normalization videos is to subtract the mean value of all pixels which is calculated across all training videos. To further eliminate the affect of different lighting, we divide the value of every

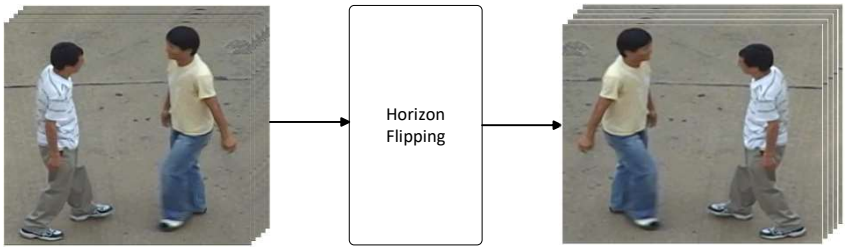


Figure 4.3: Horizontal flipping of a video

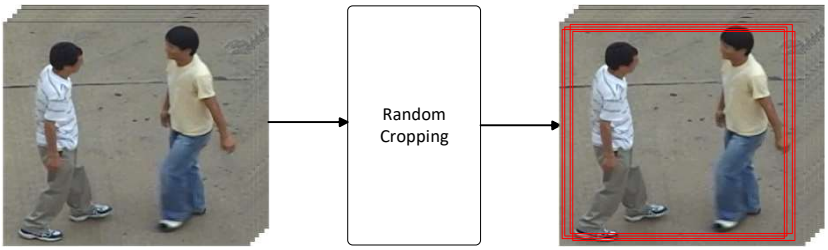


Figure 4.4: Random cropping a video

pixel by the standard deviation with following formulas:

$$\mu = \frac{\sum_{m,l,h,w,c} p_{m,l,h,w,c}}{N} \tag{4.14}$$

$$std = \sqrt{\frac{\sum_{m,l,h,w,c} (p_{m,l,h,w,c} - \mu)^2}{N}} \tag{4.15}$$

$$p_{Norm} = \frac{p_{m,l,h,w,c} - \mu}{std} \tag{4.16}$$

Where m,l,w,h,c are the number of video clips, the number of frames per video clip, the width, height and the number of channels per frame respectively. $p_{m,l,h,w,c}$ is the pixel value, μ and std are the mean value and standard deviation of pixels over all training videos. And the p_{Norm} is a generated pixel value after normalization.

4.1.3 Feature descriptor

To fully represent the features of the input interaction videos, we employ a hierarchical feature descriptor which contains one global interaction feature descriptor and two atomic action feature descriptors. All the three feature descriptors are based on a same structure 3D ConvNet but with different size of input videos and network parameters. The input video's size for global interaction feature descriptor is $n \times 16 \times 112 \times 128 \times 3$ while that for the atomic action feature descriptors is $n \times 16 \times 112 \times 80 \times 3$, where n is the number of video clips, $l = 16$ is the number of frames of each video clip, $112 \times 128 \times 3$ and $112 \times 80 \times 3$ are the frame sizes for the global interaction feature descriptor and the atomic action feature descriptors respectively. The structure of feature descriptor is illustrated in Figure 4.5.

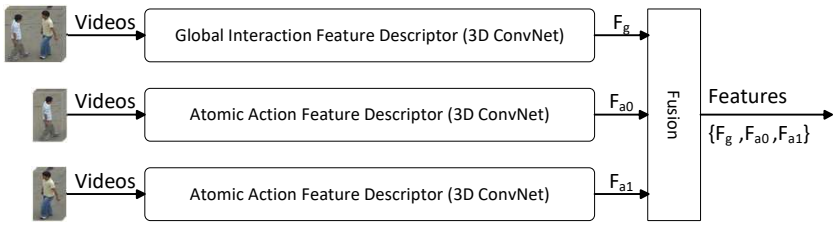


Figure 4.5: The structure of the feature descriptor

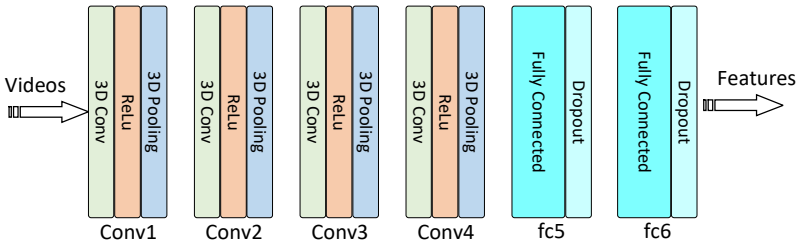


Figure 4.6: The architecture of the 3D ConvNet

The Architecture of the 3D ConvNet is illustrated in Figure 4.6. The 3D ConvNet feature descriptor contains 4 3D convolutional and pooling layers and 2 fully connected layers.

The 3D convolutional and pooling layer

Each 3D convolutional and pooling layer contains three main operations: 3D convolution, Rectified Linear Unit (ReLU) and pooling as shown in Figure 4.6. We also introduce two Batch Normalization [11] layers after the first two pooling layers.

3D convolution: Different from a 2D convolutional layer which only applies convolution spatially, the 3D convolutional layer applies convolution spatially and temporally, thus, it can extracts both spatial and temporal features. In our project, the input of the 3D convolutional layer is a 5D volume with size $[n,l,h,w,c]$ and the size of the 3D filter is $[d,k,k,c]$. The 3D filter convolves a 3D volume which has the same shape with it in the input video. Then the 3D volume slides spatially and temporal across over the whole input video to generate a new output 3D volume which has the same shape as the input video (if boundary padding is applied) except all input channels are summed. The 3D convolution can be described as the following formula:

$$Output_{n,i,j,k} = \sum_{c} \sum_{di,dj,dk}^{h,w,l} In_{n,i+di,j+dj,k+dk} * Filter_{di,dj,dk} \tag{4.17}$$

Where n is the number of video clips, l is the number of frames per video clip, h,w,c are the height, width and number of channels per frame respectively. $In_{n,l,h,w,c}$ is the pixel value at the

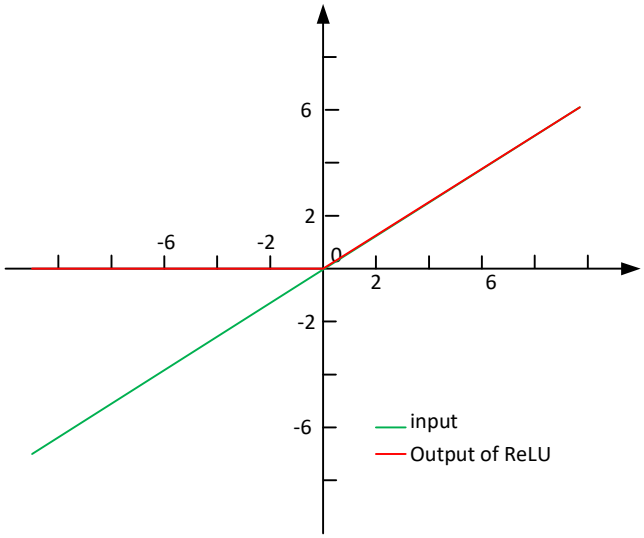


Figure 4.7: The illustration of the ReLU operation

specified position. i,j,k are the offsets of the sliding window with strides 1. $Filter[di,dj,dk]$ is the value of the filter at point $[di,dj,dk]$.

There are nof_conv filters in each convolutional layer. Each filter convolves the input volume separately, and all the outputs of the convolutional results are concatenated together as the final output of a convolutional layer. For example, $[16,16,112,128,3]$ is the shape of an input volume which has 16 video clips, 16 frames with size $[112,128,3]$ per video clips , $[3,3,3,3,32]$ is the shape of the filters which have 32 filters with size $[3,3,3,3]$. The last 3 is the number of channels which must be as same as the number of channels of the input volume.

ReLU: An additional operation called Rectified Linear Unit (ReLU) is employed after every convolution operation. The purpose of ReLU is to introduce non-linearity in our 3D ConvNet. The ReLU is an pixel wise operation and replaces all negative pixel values of the output of convolutional layer by zero which is defined in Figure 4.7 and following formula:

$$Output = Max(0, Input) \tag{4.18}$$

Batch Normalization Batch normalization alleviates a lot of headaches with properly initializing neural networks by explicitly forcing the activations throughout a network to take on a unit Gaussian distribution at the beginning of the training. Batch normalization is performed to each neuron. The forward propagation steps are shown as below:

1. Calculate the mean value and variance over the input mini-batch.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{4.19}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{4.20}$$

   Where μ_B and σ_B^2 are the mean and variance of the input mini-batch, respectively.

2. Normalize the input mini-batch.

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{4.21}$$

3. Scale and shift the normalization results to reconstruct the data.

$$y_i = \gamma \hat{x_i} + \beta \tag{4.22}$$

In the training phase, γ and β are updated by the optimizer, and in the testing phase, the mean values of γ and β are used.

Max pooling: According to the previous researches [34] [1], max pooling performs better in the 3D ConvNet than other methods of pooling, so, we also employ 3D max pooling in our 3D ConvNet. The purpose of operating max pooling is to reduce the dimensionality of each feature map but retains the most important information. In our project, we define a spatial and temporal neighbourhood (a $2 \times 2 \times 2$ window) and take the largest element in that window.

Fully connected layer

A fully connected layer is a traditional multi-layer perception where every neurons in the previous layer is connected to every neurons in the next layer. The outputs from the convolutional and pooling layers represent high-level features of the input video clips, while the fully connected layers are designed to learn some non-linear combinations of the output feature maps of the convolutional layers.

To avoid overfitting to the training set, a dropout layer [33] is added after each fully connected layer. While training, dropout is implemented by only keeping a neuron active with some probability p, or setting it to zero otherwise.

Fusion of features

As mentioned at the beginning of this section, we have three sub feature descriptors in our hierarchical feature descriptor. The output features of these three feature descriptors are notated as F_g (features extracted by the global interaction feature descriptor), and F_{a0},F_{a1} (features

extracted by the two atomic action feature descriptors). We concatenate the three sets of features as the final output features of our hierarchical feature descriptor, the final output features are notated as below:

$$Features = \{F_g, F_{a0}, F_{a1}\} \tag{4.23}$$

4.1.4 Softmax classifier

The output of our hierarchical feature descriptor is a vector with 4096×3 dimensions. And we have 6 interaction classes for classification task and 7 interaction classes for detection task (one more class than classification is used to label the class which not belongs to anyone of the 6 interaction classes). As shown in Figure 4.8 we first transform the high dimension features to the scores for each classes:

$$h_{w,b}(X) = w^T x + b \tag{4.24}$$

Where w,b are the weights and biases, respectively, and X is the input features. For example, the dimensionality of the input features is $n \times 12288$ and we have 6 output classes, then the dimensionality of the weights should be 12288×6 and the dimensionality of biases should be 6.

We then apply softmax regression to convert the score of each class to probability distribution with the following formula:

$$\begin{bmatrix} p(y=1|x;w,b) \\ p(y=2|x;w,b) \\ p(y=3|x;w,b) \\ . \\ . \\ . \\ p(y=6|x;w,b) \end{bmatrix} = \frac{1}{\sum_{j=1}^{6} e^{h_{w_j,b_j}(X)}} \begin{bmatrix} h_{w_1,b_1}(X) \\ h_{w_2,b_2}(X) \\ h_{w_3,b_3}(X) \\ . \\ . \\ . \\ h_{w_6,b_6}(X) \end{bmatrix} \tag{4.25}$$

The set the label which gets the maximum probability value as the predicted label for the input feature.

4.1.5 Optimizer

The processes of optimization in our project are composed of initialization of the parameters, optimizing of the parameters with the ADAM optimizer, dynamic adjustment of the learning rate.

Initialization of the parameters

The network parameters, including weights and biases, are initialized before the optimizing process. We expect that weights to be small random numbers close to 0. So, we initialize all weights with a normal distribution with zero mean. And the value of biases are initialized to a constant value 0.

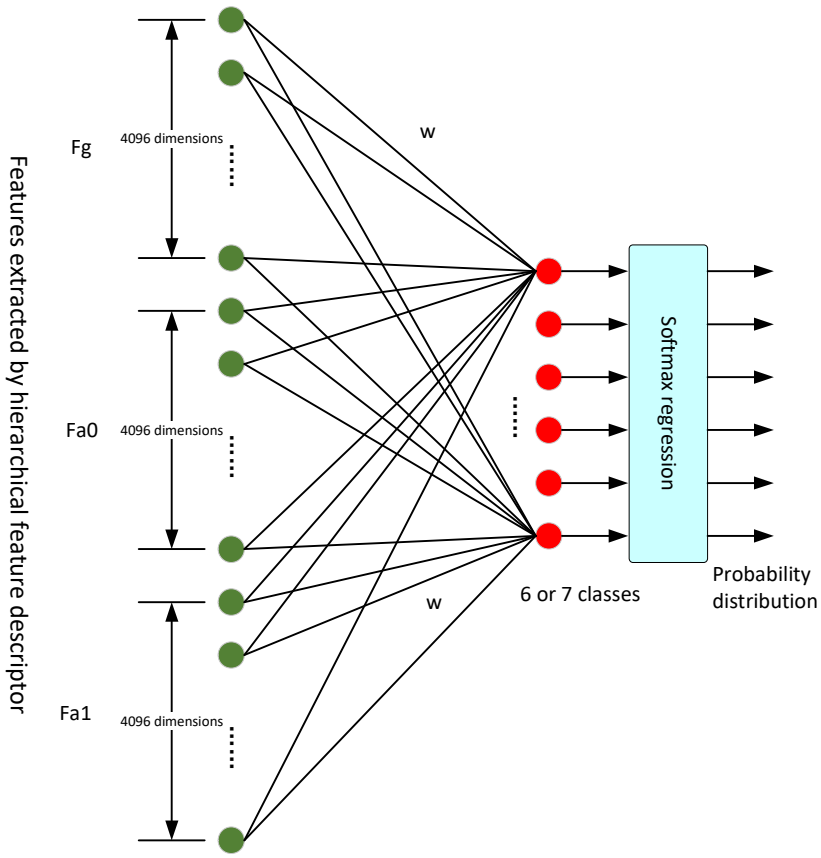


Figure 4.8: The softmax classifier

Optimizing of the parameters

The objective of the optimizer is to minimize the cross-entropy loss between the output of softmax classifier and the expected labels. We use the adaptive moment estimation (ADAM) [14] optimizer to update parameters of the whole network, including feature descriptor and classifier, with following steps:

1. Initialize all weights and biases of the whole network, weights are randomly initialized with normal distribution and biases are initialized with a small constant.

2. $m_0 \leftarrow 0$, $v_0 \leftarrow 0$ and $t \leftarrow 0$. Initialize the first moment vector m_0, second moment vector v_0 and time t.

3. $t \leftarrow t+1$. Increase time t in training process.

4. $lr_t \leftarrow learning\_rate \times \frac{\sqrt{1-\beta 2^t}}{1-\beta 1^t}$. The real time learning rate decay with the training time t. $\beta 1$ and $\beta 2$ are the exponential decay rates for the first and second moment estimations respectively.

5. $m_t \leftarrow \beta 1 \times m_{t-1} + (1-\beta 1) \times gradient$. Update the first moment vectors according to the gradients. Gradients are symbolic partial derivatives of the parameters w.r.t. the loss function and are calculated with back propagation algorithm, [27].

6. $v_t \leftarrow \beta 2 \times v_{t-1} + (1-\beta 2) \times gradient^2$. Update the second moment vectors according to the gradients.

7. $\theta_t \leftarrow \theta_{t-1} - lr_t \times \frac{m_t}{\sqrt{v_t}+\epsilon}$. Where θ_t represents the values of all parameters in the network, including all weights and biases. All parameters are updated with respect to the first and second moment vectors and the real time learning rate.

8. Repeat steps from 3 to 7 until convergence is reached.

Dynamic adjustment of the learning rate

ADAM optimizer can adjust the learning rate by itself. While, in this project, we further adjust the learning rate according to the training epochs.

$$\text{Learning Rate} = \text{Initial Learning Rate} * 2^{-int(\frac{Epoch}{M})} \tag{4.26}$$

Where $Epoch$ is one pass of all training videos, and M is the parameter to control the decay rate of the learning rate of the optimizer.

4.2 Interaction detection

In this section, we describe the data-flow and low-level design of the interaction detector. We first introduce the overall data-flow of the interaction detector followed by how we spatially and temporally detect the interactions.

4.2.1 Data-flow of interaction detection

The overall data-flow of interaction detection is shown in Figure 4.9. The processes of interaction detection mainly include following steps:

1. Detection of all people in each frame of the input videos.

2. Spatial detection of interacting people. Since there are not only the interacting people but also some irrelevant people present in some videos, and all people will be detected in the step 1. We select the bounding boxes of interacting people and drop those of irrelevant people precisely in this step.

3. Tracking of interacting people. Since the interacting people may loss in some frames, we use the Kalman filter to track the interacting people.

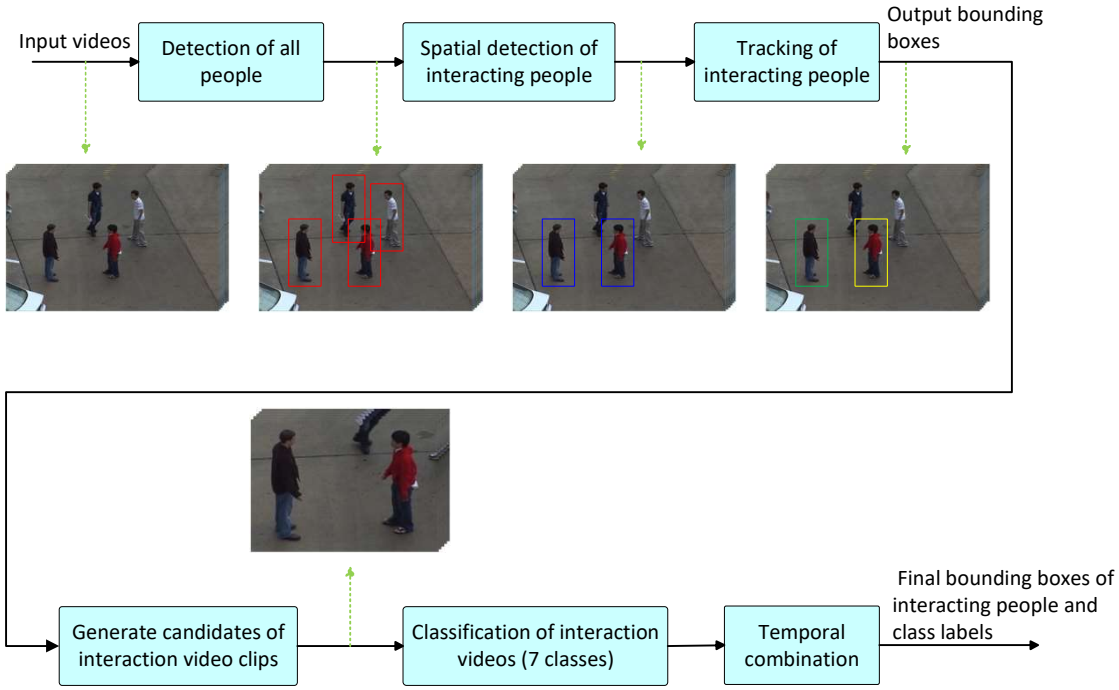


Figure 4.9: The data-flow of interaction detection

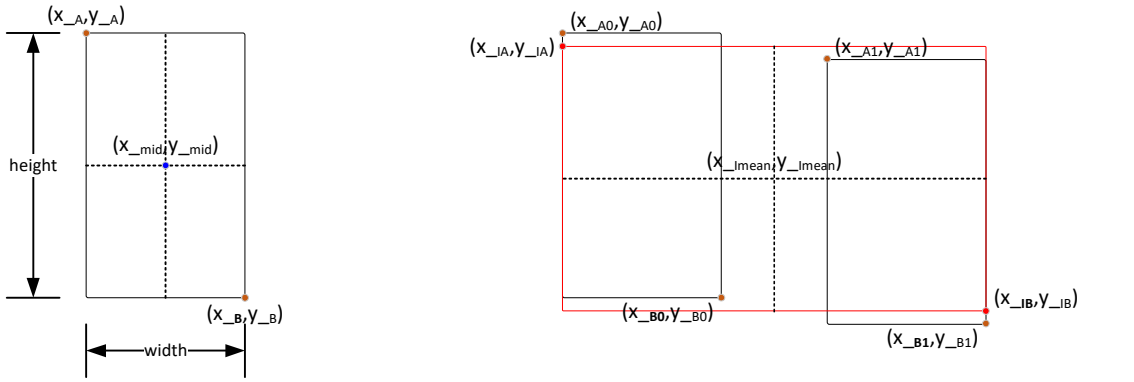
4. Generate candidates of interaction video clips. Since we have already got the spatial location of interacting people in the previous step, we generate candidates of interaction video clips by temporally sliding a fixed length video window with a stride.

5. Classification of candidate interaction video clips. We train an interaction classifier which has 7 classes and predict class labels for each candidate video clip with this interaction classifier.

6. Temporal combination of interaction detection. We find candidate videos clips which are predicted to be one of the interaction classes and temporally combine those video clips which are temporal neighbor with each other. And finally we vote one interaction class label to each cluster with the majority rule.

4.2.2 Detection of all people

We use a person detector which is the same as that in the Section 4.1.2 to locate all people in each frame of the input videos. We input the unsegmented videos provided by UT-Interaction dataset for detection tasks to the person detector, and get the detected locations in each frame of each input video. An example illustration of the input and output of the person detection is shown in Figure 4.9.



a). notations of a bounding box of an individual

b). notations of a bounding box of the two-person interaction

Figure 4.10: Illustrations of notations of bounding boxes, a). notations of a bounding boxes of an individual and b). notations of a bounding box of two-person interaction.

4.2.3 Spatial detection and tracking of interacting people

The output of the person detector are bounding boxes of all detected people of the input videos, those irrelevant people present in the videos will be selected out also as shown in the Figure 4.9. So, the purpose of spatial detection of interacting people is to locate the interacting people and ignore those irrelevant people. We first introduce the main nations and basic assumptions which we apply to spatial detection and tracking of interacting people, followed by the detail methods.

Notations and assumptions

Notations We define some notations for the bounding boxes of individual and two-person interaction as shown in Figure 4.10. For the bounding box of an individual, we notate its left-top and right-bottom points with (x_A, y_A) and (x_B, y_B) respectively, and:

$$width = x_B - x_A \tag{4.27}$$

$$hight = y_B - y_A \tag{4.28}$$

$$(x_{mid}, \quad y_{mid}) = (x_A + \frac{width}{2}, \quad y_A + \frac{height}{2}) \tag{4.29}$$

And for the bounding box of the two-person interaction, we have definitions and notations as below:

$$(x_{IA}, \quad y_{IA}) = (x_{A0}, \quad \frac{y_{A0} + y_{A1}}{2}) \tag{4.30}$$

$$(x_{IB}, \quad y_{IB}) = (x_{B1}, \quad \frac{y_{B0} + y_{B1}}{2}) \tag{4.31}$$

$$(x_{Imean}, y_{Imean}) = (\frac{x_{IA} + x_{IB}}{2}, \quad \frac{y_{IA} + y_{IB}}{2}) \tag{4.32}$$

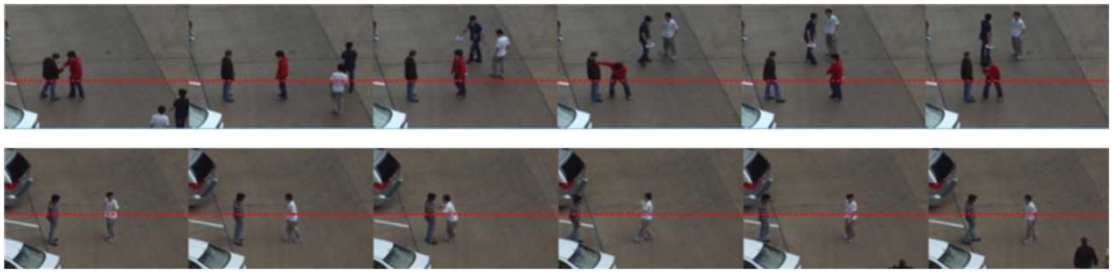


Figure 4.11: Some examples of the horizon positions of the interacting people, sampled from the unsegmented videos of UT-Interaction dataset.

Where $x_{Ai}, y_{Ai}, x_{Bi}, y_{Bi}$ are the coordinates of the ith bounding box of the individuals, $x_{IA}, y_{IA}, x_{IB}, y_{IB}$ are the coordinates of the bounding box of two-person interaction, and x_{Imean}, y_{Imean} are the coordinates of the center point of the interaction bounding box.

Assumptions To find out the interacting people from all people who present in the videos, we have following basic assumptions:

1. All individuals have similar size of bounding boxes. We design a method to learn a stable width and height of bounding box and apply them for all individuals. By doing this, those over-small or over-large bounding boxes caused by the mistakes of the person detection will be eliminated.

2. The horizon positions of the interacting people are very close, which is illustrated in Figure 4.11.

3. The moving speed of an interaction bounding box is slow between two successive frames.

Learn the stable width and height of the bounding boxes for individuals

We assume that the bounding boxes of all individuals have same width and height throughout a whole video. So, we design a method to learn such a width and height. We learn the value of width with the formulas as below and learn the value of height in the same way.

$$w_1^t = Norm\left(\frac{1}{abs(width_i^{t-1} - width_i^t)}\right) \tag{4.33}$$

$$w_2^t = Norm\left(\frac{1}{Min(abs(width_i^t - width_i^t))}\right) \tag{4.34}$$

$$width_t = \sum_{i=0}^{N} width_i^t \times (0.5 \times w_{1i}^t + 0.5 \times w_{2i}^t) \tag{4.35}$$

Where w_1^t is the temporal learning weight which puts more weights on those new bounding boxes of which the value of width is closer to the learned width, w_2^t is the spatial learning weight which puts more weights to those new bounding boxes which has very similar width with each other in the same frame. $Norm(X_i) = \frac{X_i}{\sum_i X_i}$ is a vector normalization function. $width_i^t$ is the value of the width of the ith bounding box at time t, and $width_{\bar{i}}^t$ are the values of the width of the other bounding boxes except the ith one at time t.

By applying above method, the learned width and height can be more general represent the width and height of each individual's bounding box. And those bounding boxes with area less than $0.75 \times widht \times height$ or larger than $1.25 \times width \times height$ are discarded as false positive detections.

Learn the value of X_{Imean} and Y_{Imean}

We have previously defined X_{Imean} and Y_{Imean} as the coordinates of the center point of the interaction bounding box. Since we do not previously know the positions of interacting people, then value of X_{Imean} and Y_{Imean} can not be calculated directly. We learn the value of X_{Imean} and Y_{Imean} from the bounding boxes of all people.

Learn the value of Y_{Imean} We assume that the interacting people usually stand in a very close horizontal position, thus we put more weights to those bounding boxes which have very close values of y_{mid}. And we assume that the value of Y_{Imean} change very slowly, so, we put more weights to those bounding boxes of which the values of y_{mid} are close to current Y_{Imean}. We learn the value of Y_{Imean} with following formulas:

$$w_1^t = Norm\left(\frac{1}{|Y_{Imean}^{t-1} - y_{mid\_i}^t|}\right) \tag{4.36}$$

$$w_2^t = Norm\left(\frac{1}{Min(|y_{mid\_i}^t - y_{mid\_\bar{i}}^t|)}\right) \tag{4.37}$$

$$Y_{Imean}^t = \sum_{i=0}^{N} y_{mid\_i}^t \times (0.5 \times w_{1i}^t + 0.5 \times w_{2i}^t) \tag{4.38}$$

To further stably learn the value of Y_{Imean}, we add an additional parameter $learning\_rate$ to slow the learning rate of Y_{Imean} if its value is stable. We detect the stability of the values of

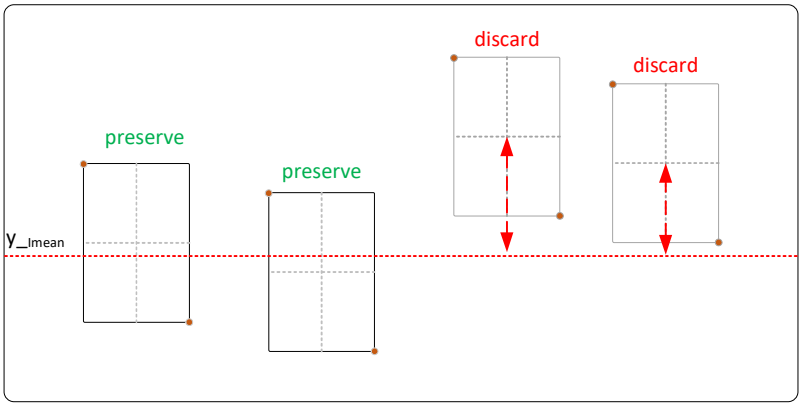


Figure 4.12: An example illustration of preserving and discarding bounding boxes according to the value of Y_{Imean}

Y_{Imean} and calculate $learning\_rate$ with the following formulas:

$$Y_{Imean}\_List = [Y^{t}_{Imean}, Y^{t-1}_{Imean}, ..., , Y^{t-19}_{Imean}] \tag{4.39}$$

$$\mu = \frac{1}{20}\sum_{i=0}^{19} Y_{Imean}\_List_i \tag{4.40}$$

$$std = \sqrt{\frac{1}{20}\sum_{i=0}^{19}(Y_{Imean}\_List_i - \mu)^2} \tag{4.41}$$

$$learning\_rate = Min(1, std/20) \tag{4.42}$$

$$Y^{t}_{Imean} \leftarrow Y^{t-1}_{Imean} + learning\_rate \times (Y^{t}_{Imean} - Y^{t-1}_{Imean}) \tag{4.43}$$

Where Y_{Imean}_List is a vector which contains 20 history values of Y_{Imean}, μ and std are the mean value and standard deviation of the list Y_{Imean}_List. We only preserve those bounding boxes of which the distances between Y_{Imean} and y_{mid} are less than a small value $thld\_y$. An example illustration of preserving and discarding bounding boxes according to the value of Y_{Imean} is shown in Figure 4.12.

Learn the value of X_{Imean} We learn the value of X_{Imean} from the preserved bounding boxes in the previous step with following formulas:

$$X_{mean} = \sum_{i=1}^{M} x_{mid\_i} \tag{4.44}$$

$$learning\_rate1 = \frac{1}{Max(1, \sqrt{|X_{Imean} - X_{mean}|})} \tag{4.45}$$

$$X_{Imean}\_List = [X_{Imean}^{t}, X_{Imean}^{t-1}, ..., , X_{Imean}^{t-19}] \tag{4.46}$$

$$\mu = \frac{1}{20} \sum_{i=0}^{19} X_{Imean}\_List_i \tag{4.47}$$

$$std = \sqrt{\frac{1}{20} \sum_{i=0}^{19} (X_{Imean}\_List_i - \mu)^2} \tag{4.48}$$

$$learning\_rate2 = Min(1, std/20) \tag{4.49}$$

$$X_{Imean}^{t} = X_{Imean}^{t-1} + learning\_rate1 \times learning\_rate2 \times (X_{mean}^{t} - X_{Imean}^{t-1}) \tag{4.50}$$

Where X_{mean} is the mean value of x_{mid} of the preserved bounding boxes in the previous step. $learning\_rate1$ is determined by the distance between the X_{mean} and current X_{Imean} and $learning\_rate2$ is determined by the stability of history values of X_{Imean}. By doing this, we update the value of X_{Imean} slowly if the new X_{mean} is far away from X_{Imean} or the value of X_{Imean} is already get stable. We then use the value of X_{Imean} to discard those bounding boxes which are far away from the vertical axis defined by the value of X_{Imean} and only preserve one bounding boxes in each side of the vertical axis. Some examples illustrate how we preserve and discard bounding boxes according to the value of X_{Imean} are shown in Figure 4.13.

Tracking of interacting people

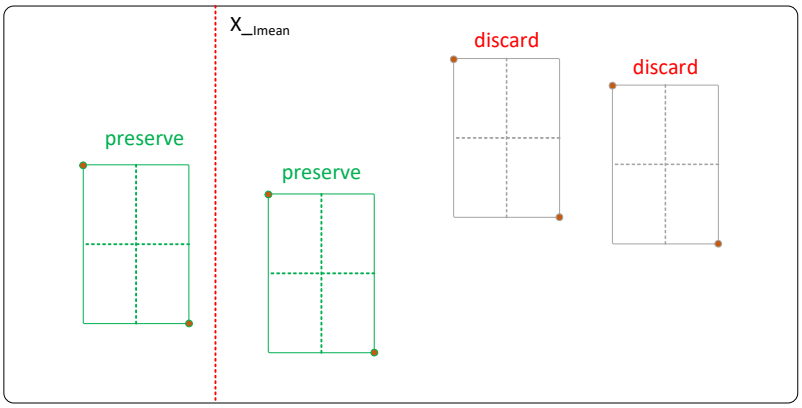
After discarding those bounding boxes according to the learned values of width, height, Y_{Imean} and X_{Imean}, we get the bounding boxes of interacting people. Since detections may loss in some frames, we use the Kalman filters to track the locations for each person involved in the interaction.

4.2.4 Generate candidates of interaction video clips

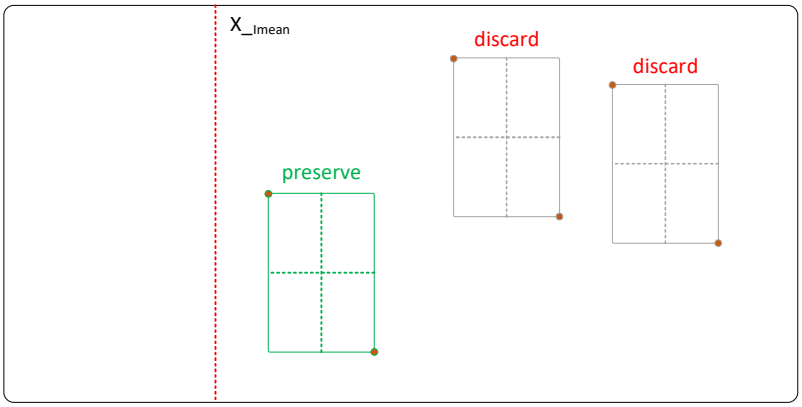
We generate the candidates of interaction videos clips by temporally sliding a 3D window with a fixed length l and temporal stride s. The illustration of sliding the 3D windows which are used to generate candidates of interaction video clips is shown in Figure 4.14. And the coordinates of the sliding windows are calculated with following formulas:

$$(x_{IA\_i}, y_{IA\_i}) = (\frac{\sum_{j=i*s}^{i*s+l} x_{A\_j}}{l}, \quad \frac{\sum_{j=i*s}^{i*s+l} y_{A\_j}}{l}) \qquad i \in \{0, 1, 2, ...\} \tag{4.51}$$

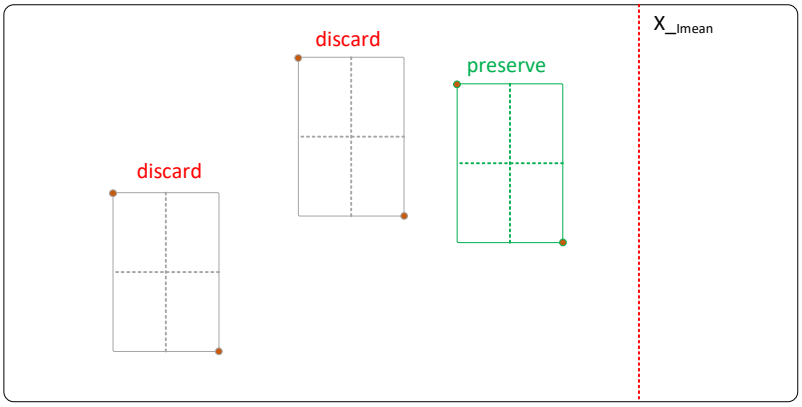
$$(x_{IB\_i}, y_{IB\_i}) = (\frac{\sum_{j=i*s}^{i*s+l} x_{B\_j}}{l}, \quad \frac{\sum_{j=i*s}^{i*s+l} y_{B\_j}}{l}) \qquad i \in \{0, 1, 2, ...\} \tag{4.52}$$



a). Preserve two bounding boxes if there are bounding boxes exist in both side of vertical axis defined by the value of X_Imean



b). Preserve one bounding box if there are bounding boxes only exist in right side of vertical axis defined by the value of X_Imean



c). Preserve one bounding box if there are bounding boxes only exist in left side of vertical axis defined by the value of X_Imean

Figure 4.13: An example illustration of preserving and discarding bounding boxes according to the value of X_{Imean}

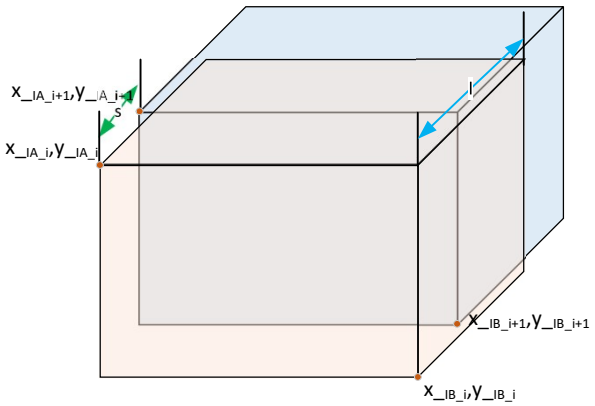


Figure 4.14: The illustration of temporally sliding the 3D windows which are used to generate candidates of interaction video clips.

Where $x_{A\_j}, y_{A\_j}, x_{B\_j}, y_{B\_j}$ are the coordinates of the spatial bounding box of interacting people in the jth frame.

4.2.5 Classification of interaction video clips

The interaction classifier we trained in the classification task has only 6 classes, but in the detection task, we need an additional class to label those video clips which do not belong to one of the 6 positive classes.

Training of interaction classifier for detection task

The UT-Interaction dataset only provides training videos for the 6 positive classes, so we generate a batch of video clips by randomly cropping in the unsegmented videos, and for each that does not overlap with any positive bounding boxes. We keep those new boxes as negative training video clips. For simplicity, we use the average width and height of all positive bounding boxes as the values of the width and height of the negatives and the number of frames for each negative video clip is fixed to 64. We then use all the positive and negative training video clips to train our interaction classifier (7 classes).

We apply the trained interaction classifier (7 classes) to predict the class labels for all candidates of interaction video clips and get the results in the format as below for each candidate: $\{Start\_No., End\_No., x_A, y_A, x_B, y_B, class\_lable\}$.

4.2.6 Temporal combination

We first drop those candidates with negative predicted class_labels, then temporally combine those candidates which have successive $Start\_No$ as shown in Figure 4.15.

After temporal combination of candidates, we get many combined candidates which consist of server original candidates in each. Since the predicted class labels of original candidates in

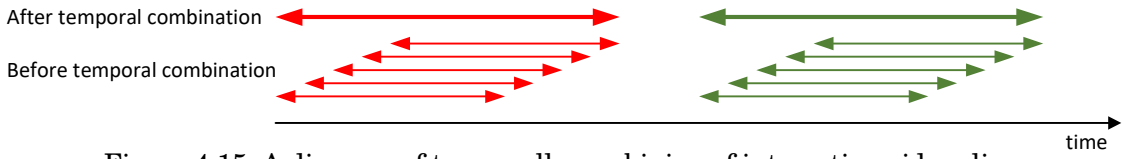


Figure 4.15: A diagram of temporally combining of interaction video clips

a single combination may be different, we use the majority rule to vote a class label as the final class label for the combination. For example, a combined candidate consist of 5 original candidates which have predicted class labels $[2,3,3,3,1]$, then we take 3 as its final class label.

CHAPTER 5

EXPERIMENTAL RESULTS

The performance of the classifications and detections is significantly determined by the pre-processing of training data, the hyper-parameters of the network and the training parameters. A number of experiments are performed to demonstrate how those factors influence the performance. We start off with the results of how the pre-processing of training data and the hyper-parameters of the network affect the classification accuracy base on a singleNet, see 5.1.1. At last, we give the overall experimental results of the classification and detection tasks.

5.1 Searching of the optimal parameters

We first introduce the network model which we employ for searching of the optimal parameters. Then, we will overall introduce the factors which would affect the classification accuracy. Finally, we present the experimental schemes, results and analyses.

5.1.1 Model

Due to limitation of GPU memory and training time, we search the optimal parameters of the data pre-processing and the network on a singleNet which is only composed by a global interaction feature descriptor instead of the hierarchical interaction feature descriptor and a softmax classifier. The architecture of the singleNet is illustrated in Figure 5.1. We assume that if the parameters are optimal for the singleNet, then they are also optimal parameters for the fullNet which is composed by the hierarchical interaction feature descriptor and a softmax classifier, see Section 3.1.1, because they have similar network model. Due to that we only need to search the optimal parameters by comparing the loss and classification accuracy between

Figure 5.1: The architecture of the singleNet

different parameter settings, it is not necessary to evaluate them on all data. For the sake of saving training time, it is reasonable to use part of data instead of the all. So, all training and testing video used in the experiments of this section are from the UT-Interaction dataset set1, see Section 2.4.2.

5.1.2 Assumptions and Factors and the default parameter setting

Since there are many factors which influence the classification accuracy, and many parameters in each factor, it is impossible to search the optimal parameters by going through all the model parameters, at least under our current resource conditions. So, we assume that the effects of each parameter on the classification accuracy are independent. Based on this assumption, we have a set of default parameters as shown in Table 5.1. We only adjust the parameters of one factor, keep the other parameters as default value, and observe the experimental results. At last, we use the combination of each factor's optimal parameter as our network's optimal parameters.

5.1.3 Experiments

5.1.3.1 Initialization of the network parameters

Experimental scheme To observe how much the initial values of the weights affect the classification accuracy, we evaluated two initialization methods, including normal distribution with zero mean and "Xavier" method [8]. We evaluated the "Xavier" initialization with uniform distribution to compare the performance between the two main-stream initialization schemes. We initialize all biases to 0.1 when we evaluate the weights initializations.

It's possible and common to initialize the biases to zero, but we only tested non-zero initial biases because we use ReLU non-linearities and non-zero initialization of biases ensures that all ReLU units fire in the beginning and therefore obtain and propagate some gradient. We evaluated 3 constant value initializations for biases, including 0,001, 0.01 and 0.1.

Experimental results and analysis The experimental results of comparison between "Xavier" and normal distribution initialization ($\mu = 0, \sigma = 0.1$) are illustrated in Figure 5.2. The experimental results of the three different initial biases are illustrated in Figure 5.3.

From the experimental results, we can get following conclusions about the initializations of network parameters (weights and biases):

Table 5.1: Default parameters for network and data pre-processing

No.	Factors	Parameters
Network		
1	Initial values of the network parameters, see Section 4.1.5.	$weights = N(0, \sigma^2)$, $\sigma = 0.01$ $biases = 0$
2	Dynamic adjusting of the learning rate, see Section 4.1.5.	$Learning\ rate = 1e^{-4} \times 2^{-int(\frac{epoch}{4})}$
3	Dropout layers, see Section 4.1.3	$keep\_ratio = 0.5$
4	Batch normalization layers, see Section 4.1.3	Yes
5	The number of convolutional layers, see Section 4.1.3	4
6	The size of convolutional kernels, see Section 4.1.3	$3 \times 3 \times 3$
7	The sizes of the pooling kernels, see Section 4.1.3	L1: $1 \times 2 \times 2$ L2, L3: $2 \times 2 \times 2$ L4: $4 \times 2 \times 2$
8	The number of filters for each convolutional layer, see Section 4.1.3	nof_conv1 = 32 nof_conv2 = 128 nof_conv3 = 256 nof_conv4 = 512
9	The number of output neurons for each fully connected layer, see Section 4.1.3	noo_fc$_*$ = 4096
Data pre-processing		
1	Data augmentation: horizontal flipping, see Section 4.1.2 1	Yes
2	Data augmentation: random cropping, see Section 4.1.2 1	Yes, 4 times
3	Temporal down-sampling of frames, see Section 4.1.2 2	Yes, N=0, see 2
4	Data normalization, see Section 4.1.2 4	Yes, $X_i = \frac{X_i - Mean(X)}{Std(X)}$

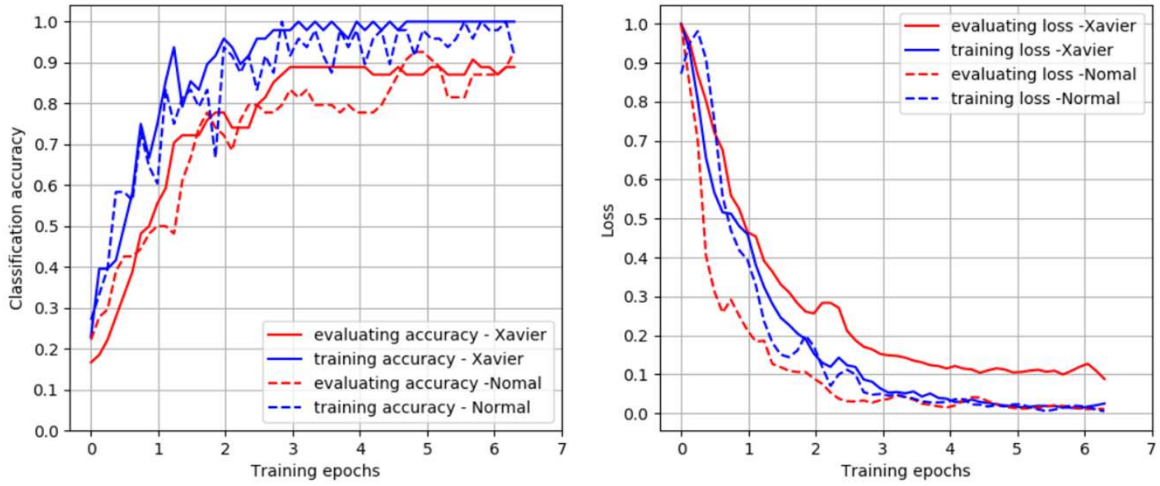


Figure 5.2: The loss and classification accuracy vs. training epochs comparison between "Xavier" and normal distribution initialization.

1. Under the same network condition, "Xavier" initialization gets more stable classification accuracy that the normal distribution initialization.

2. The evaluating loss of the "Xavier" initialization is slight larger that that of the normal distribution initialization. But this may be caused by the far larger initial loss value of the normal distribution initialization. There is no obvious over-fitting for both initializations.

3. For the initialization of biases, network with relative larger initial biases can converge faster and achieve better classification accuracy with the ReLU non-linearity.

So, we will employ "Xavier" initialization in our network and set the initial biases to 0.1.

5.1.3.2 Learning Rate

Experimental scheme We tested both constant learning rate and decaying learning rate over the period of the training process. For constant learning rate, we tested values including 0.001, 0.0001 and 0.00001. We also tested the case that the learning rate is divided by 2 after every 3 epochs with initial value 0.0001.

Experimental results and analysis The experimental results of the loss and classification accuracy vs. training epochs comparison between networks with different learning rate are illustrated in Figure 5.4. According to the experimental results, we can get following conclusion about the learning rate:

1. Larger learning rate, e.g. $1e-3$, accelerates the convergence speed of the network, but it is also easier to over-fit to the training data and even causes unstability.

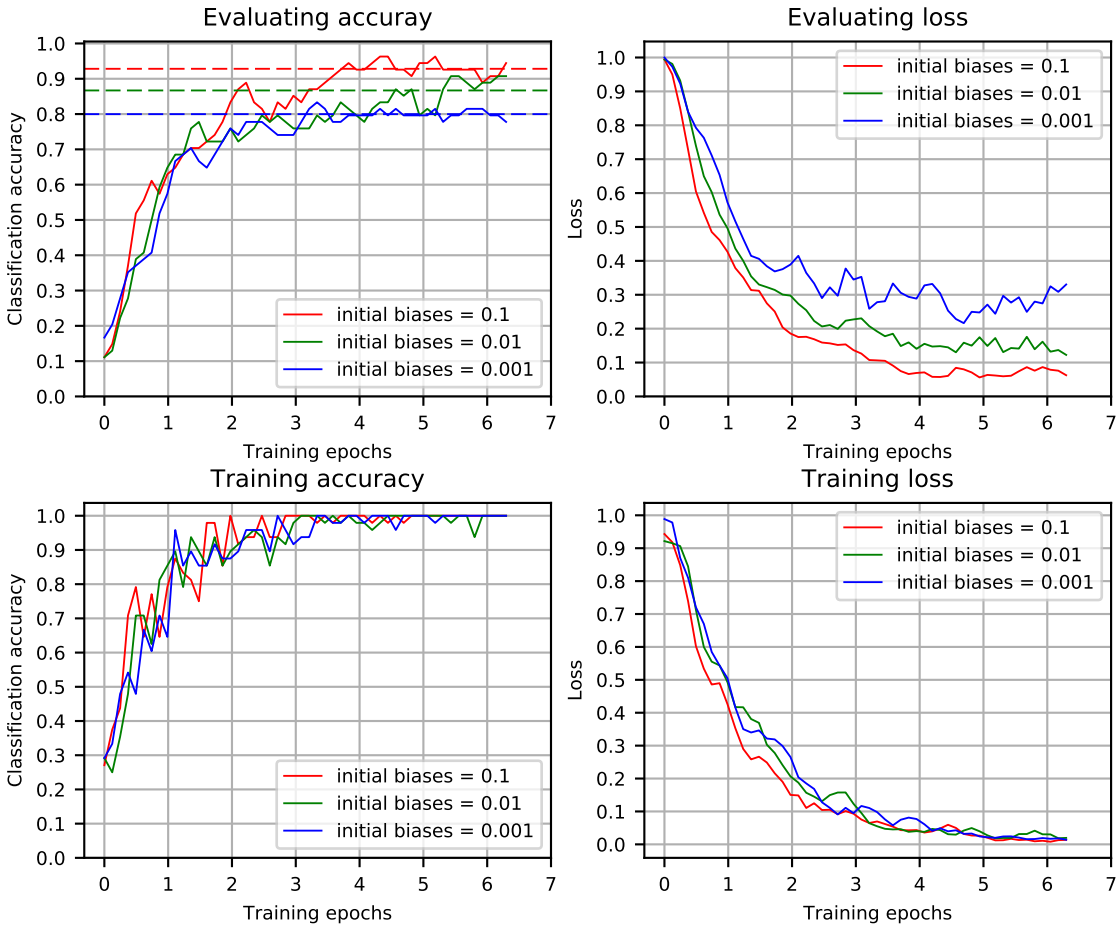


Figure 5.3: The loss and classification accuracy vs. training epochs comparison between different initial biases.

2. The training process with smaller learning rate, e.g. $1e-5$, converge slowly.

3. The training processes with learning rate $1e-4$ is more suitable for our network. And whether decay the learning rate over to the training epochs doesn't affect the performance much.

So, we set the learning rate to $1e-4$ for our network.

5.1.3.3 Dropout layer

Experimental scheme Dropout layer is considered as an extremely effective way to reduce overfitting in neural network,see Section 4.1.3. We tested different values of the probability p, including 0.8,0.5,0.2.

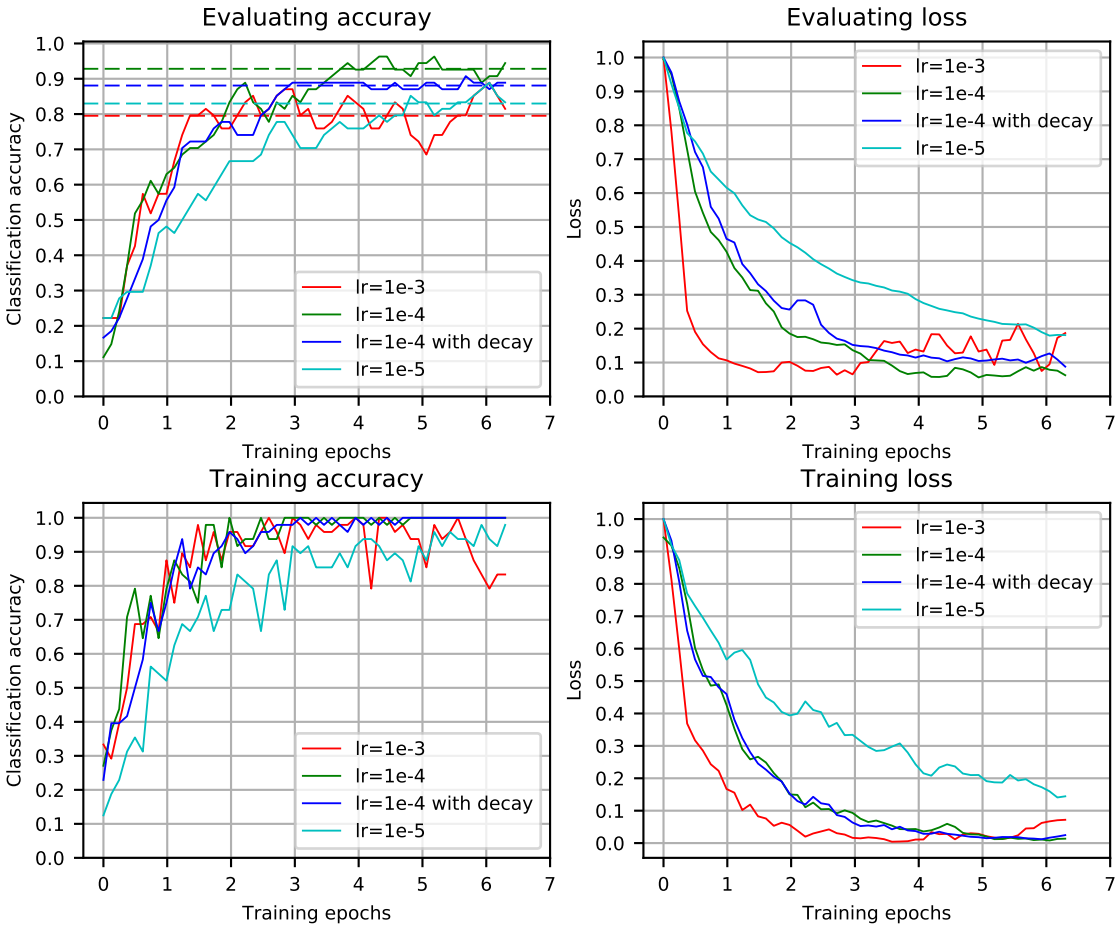


Figure 5.4: The loss and classification accuracy vs. training epochs comparison between networks with different learning rate.

Experimental results and analysis The experimental results of the loss and classification accuracy vs. training epochs comparison between different keep probabilities of the dropout layers are illustrated in Figure 5.5.

According to the experimental results, we can get following conclusion about the keep probabilities in the dropout layers:

1. Larger keep probability (0.8) in the dropout layers makes the network converge faster, but the classification accuracy keeps at a lower level after convergence.

2. Smaller keep probability (0.2) in the dropout layers makes the network converge slower, but the final classification accuracy is higher after convergence.

3. 0.5 is a reasonable value of keep probability because it has good balance between performance and convergence speed.

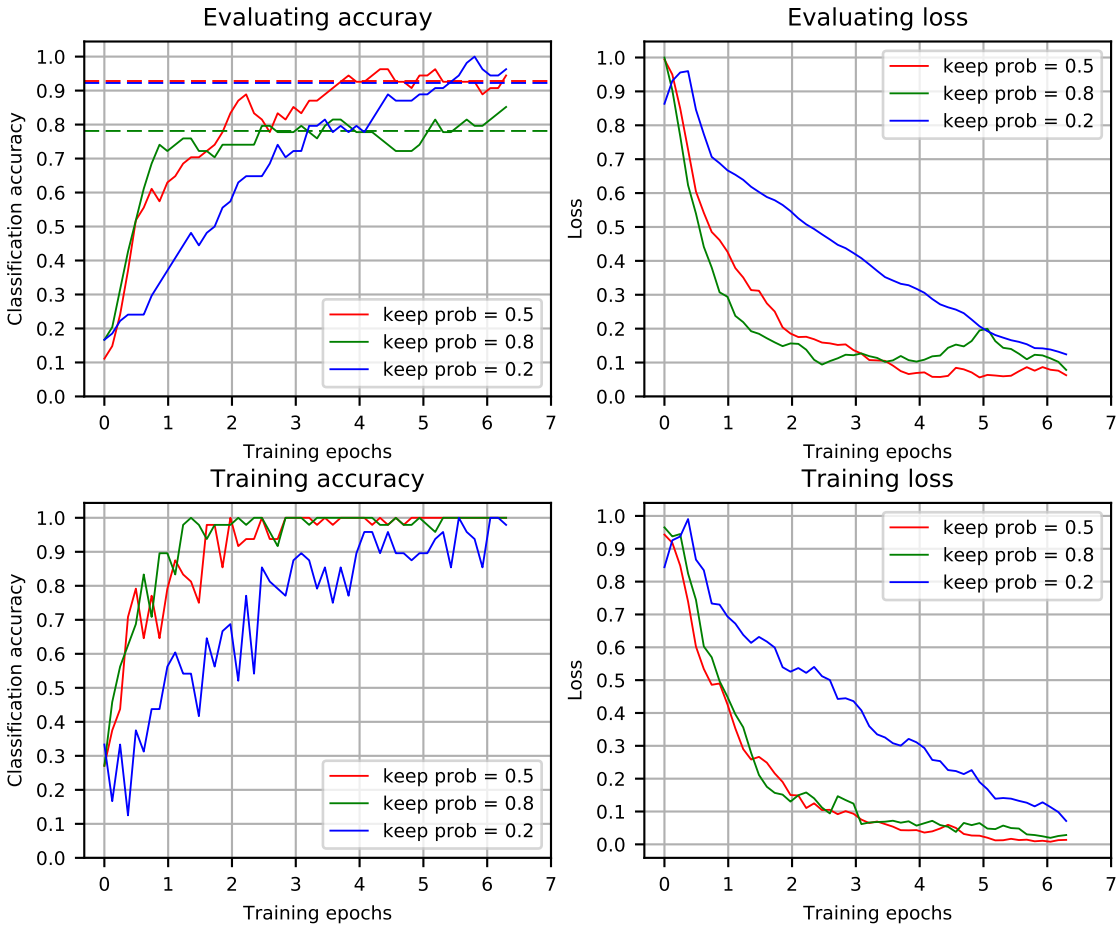


Figure 5.5: The loss and classification accuracy vs. training epochs comparison between different keep probabilities in the dropout layers.

So, we set the keep probabilities to 0.5 in our network.

5.1.3.4 Batch normalization layer

Experimental scheme To observed the effects of batch normalization (BN) layers, see Section 4.1.3, we compared the loss and classification accuracy between the network with and without BN layers. The only difference between the two networks is whether there are BN layers after the first two ReLU layers.

Experimental results and analysis The experimental results of the loss and classification accuracy vs. training epochs comparison between networks with and without Batch normalization layers are illustrated in Figure 5.6.

According to the experimental results, we can get following conclusions about batch normalization layers:

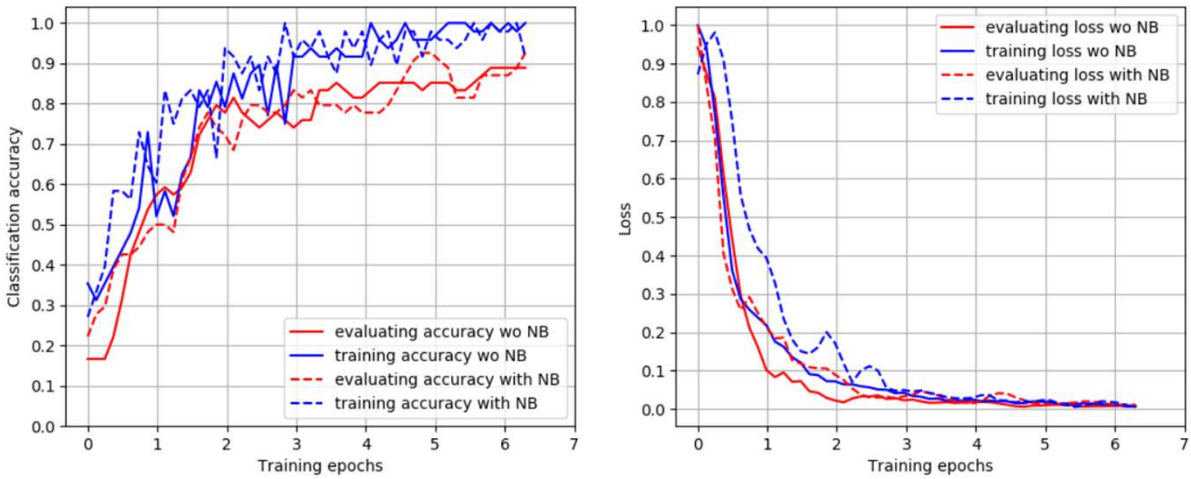


Figure 5.6: The loss and classification accuracy vs. training epochs comparison between networks with and without batch normalization layers.

1. The convergence speed comparison between the network with and without BN layers is similar.

2. The network with batch normalization layers gets slightly higher classification accuracy, about 0.92, that the network without batch normalization layers, about 0.88.

So, we add batch normalization layers in our network.

5.1.3.5 The number of the convolutional layers

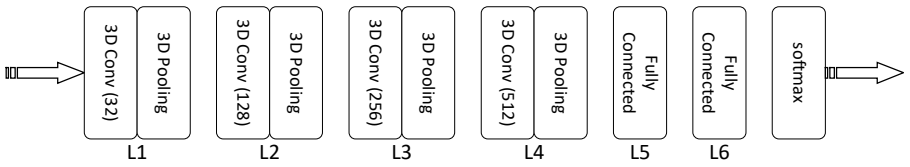
Experimental scheme Generally, the network can get stronger global information representing ability and nicer non-linearity by applying multiple convolutional layers. In our project, we tested 3 networks with different number of 3D convolutional layers, 4, 5 and 6, respectively. The structures and parameters of the three network are illustrated in Figure 5.7.

Experimental results and analysis According to the experimental results, as shown in Figure 5.8, we can get following conclusions about the different number of the 3D convolutional layers:

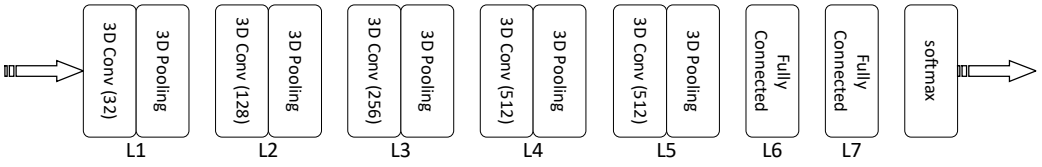
1. The convergence speed is not affected by the number of the 3D convolutional layers.

2. The 3DConvNet with 5 convolutional layers gets the best classification accuracy among the three. This may be caused by that the more layers the stronger global information representing ability. But at the same time, the network with more layers also need to be trained on richer training data, else, it may perform even worse, e.g. the case of the 3DConvNet with 6 convolutional layers.

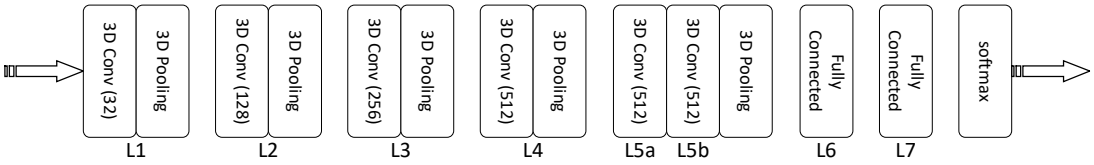
So, we set 5 convolutional layers in our 3DConvNet.



a). 3D-ConvNet with 4 convolutional layers, 4 pooling layers and 2 fully-connected layers



b). 3D-ConvNet with 5 convolutional layers, 5 pooling layers and 2 fully-connected layers



c). 3D-ConvNet with 6 convolutional layers, 5 pooling layers and 2 fully-connected layers

Figure 5.7: The structures and parameters of the three 3DConvNets which have different number of convolutional layers.

5.1.3.6 The size of the convolutional kernel

Experimental scheme We evaluate how much does the performance relate to the size of convolutional kernels by comparing four networks with different convolutional kernel sizes, including $3 \times 3 \times 3$, $3 \times 4 \times 4$, $4 \times 4 \times 4$ and $4 \times 4 \times 3$, see Section 4.1.3 for the format of kernel size. We set all convolutional layers share the same kernel size in each 3DConvNet.

Experimental results and analysis According to the experimental results, as shown in Figure 5.9, we found that there is no obvious difference both in the convergence speed and classification accuracy among the networks with different convolutional kernel sizes. So, we keep to use the the size of $3 \times 3 \times 3$ for all convolutional kernels.

5.1.3.7 The number of filters of each convolutional layer

Experimental scheme We use a 3DConvNet with 4 convolutional layers to evaluate the relationship between the performance and the number of filters of each convolutional layer. We performed four experiments which adjust the number of filters for only one convolutional layer each time, including 1). nof_conv1 $32 \rightarrow 64$, 2). nof_conv2 $128 \rightarrow 64$, 3). nof_conv3 $256 \rightarrow 512$, and 4). nof_conv4 $512 \rightarrow 256$.

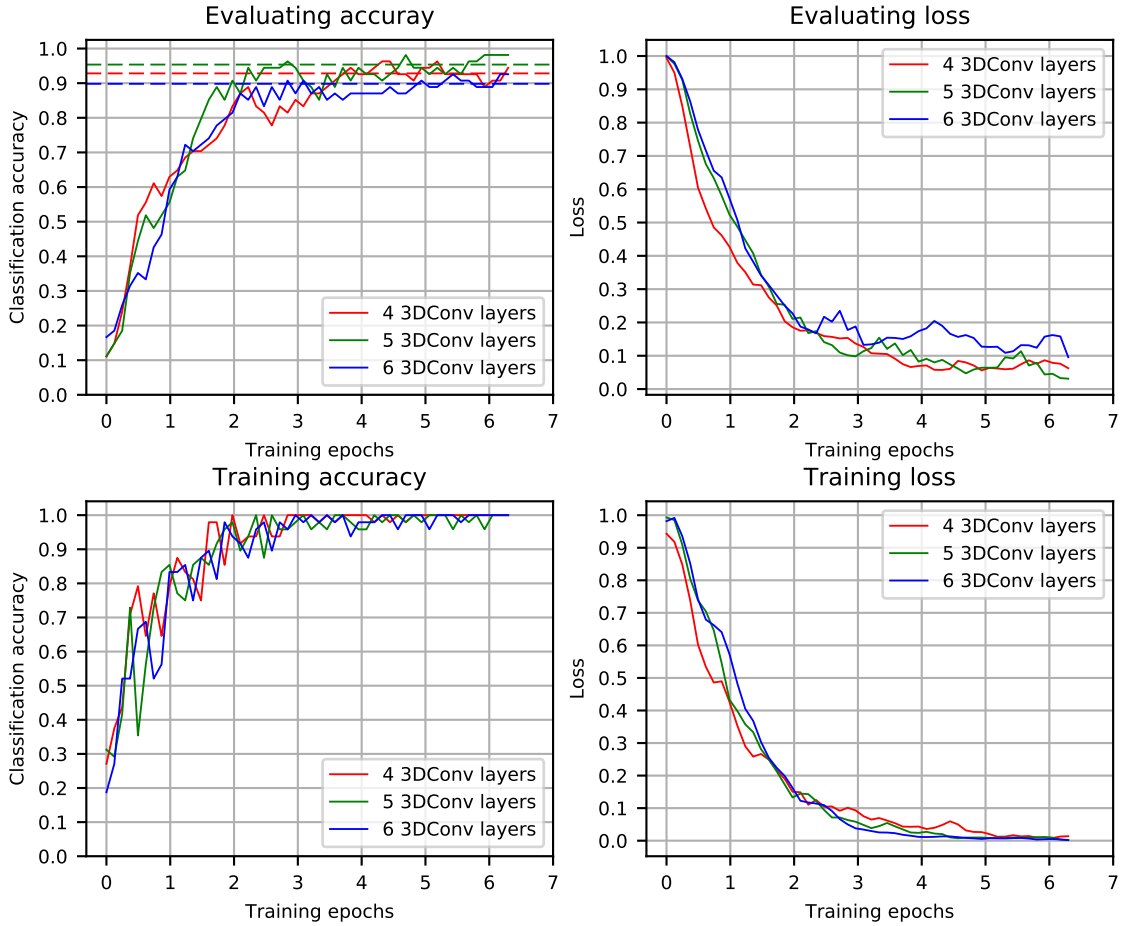


Figure 5.8: The loss and classification accuracy vs. training epochs comparison between networks with different convolutional layers.

Experimental results and analysis The experimental results are illustrated in the Figure 5.10. The numbers listed in the legend represent the numbers of the filters for each convolutional layer, e.g. $[32,128,256,512]$ represents $nof\_conv1 = 32, nof\_conv2 = 128, nof\_conv3 = 256, nof\_conv4 = 512$.

According to the experimental results, the networks perform a little better when the numbers of the filters are $[32,128,256,512]$ and $[32,128,512,512]$, while the performance is a little worse when the numbers of the filters are $[32,64,256,512]$. It is hard to the find the potential laws and explain why they perform better or worse from the limited experimental data. We just use $[32,128,256,512]$ to configure the convolutional layers of our network.

5.1.3.8 The number of neurons of each fully connected layer

Experimental scheme There are two fully connected layers in our network. In this experiment, we compared the networks with different numbers of output neurons in the fully connected layers. Specifically, we compared three cases, including $[2048,2048]$, $[4096,2048]$ and $[4096,4096]$,

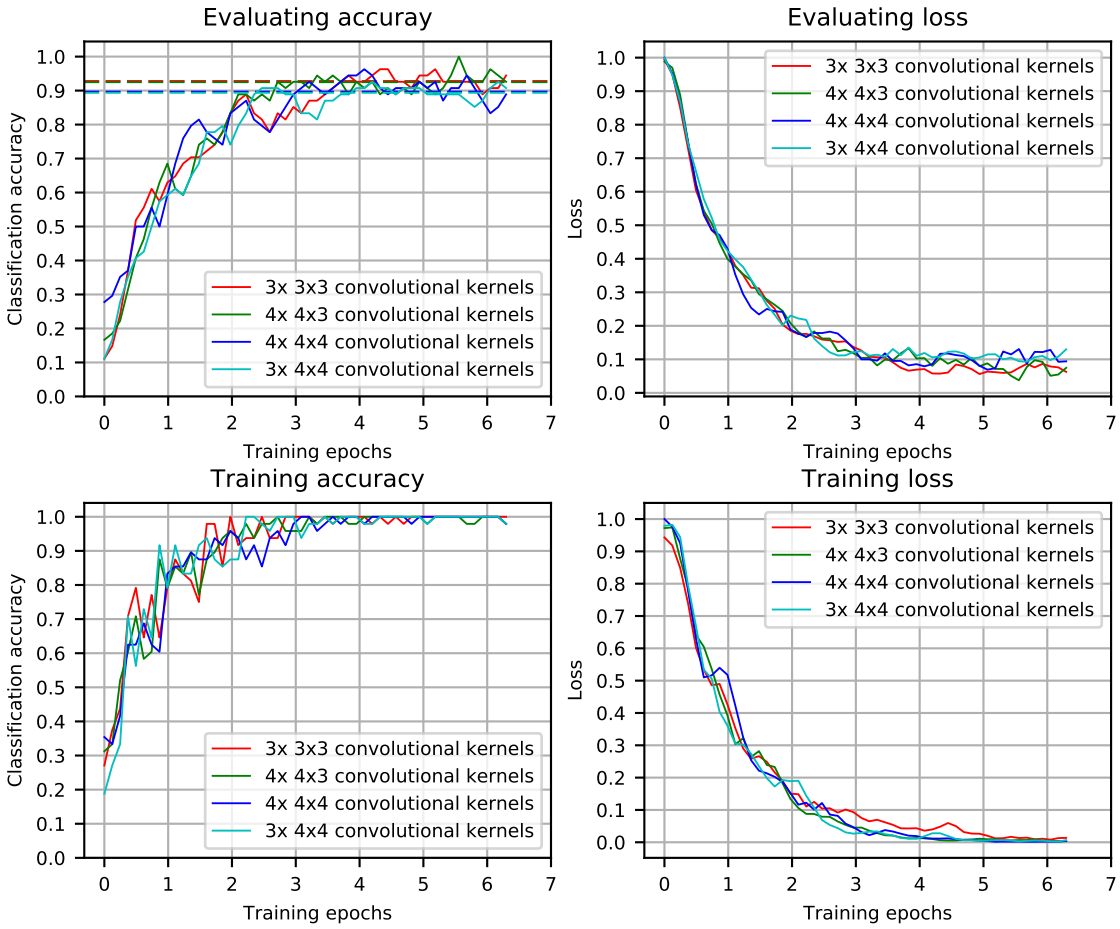


Figure 5.9: The loss and classification accuracy vs. training epochs comparison between networks with different sizes of the convolutional kernels.

where the numbers in the square bracket represent the number of output neurons of the fully connected layers, e.g. [2048, 2048] represents $noo\_fc6 = 2048, noo\_fc7 = 2048$, noo_fc6 is the notation of the number of output neurons of the FC6 layer.

Experimental results and analysis The experimental results are shown in Figure 5.11. According to the experimental results, we can get following conclusions:

1. The network converges a littler faster with [4096, 4096] output neurons, while there is no obvious difference between [2048, 2048] and [4096, 2048].

2. The network with [4096, 4096] output neurons also performs a little better than the networks with [2048, 2048] and [4096, 2048] output neurons.

So, we use [4096, 4096] to configure the fully connected layers of the network.

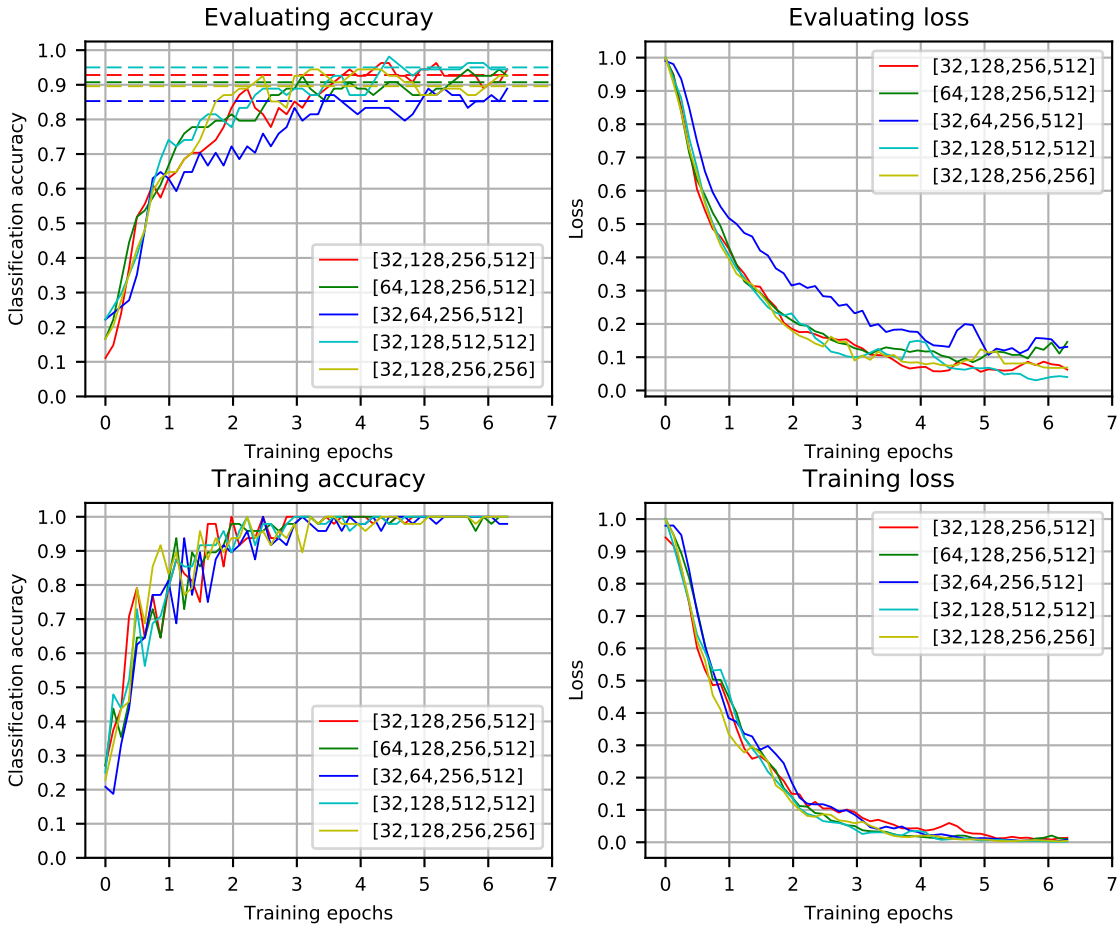


Figure 5.10: The loss and classification accuracy vs. training epochs comparison between networks with different number of filters in each convolutional layer.

5.1.3.9 Data augmentation: horizontal flipping

Experimental scheme Horizontal flipping is a common technique to augment the training data, especially for the small scale training data, see Section 4.1.2 1. We evaluated two networks trained on the data with and without horizontal flipping.

Experimental results and analysis According to the experimental results, shown in Figure 5.12, the network trained on the data with horizontal flipping significant outperforms the network which is trained on the data without horizontal flipping. So, we adopt the horizontal flipping technique to augment our training data.

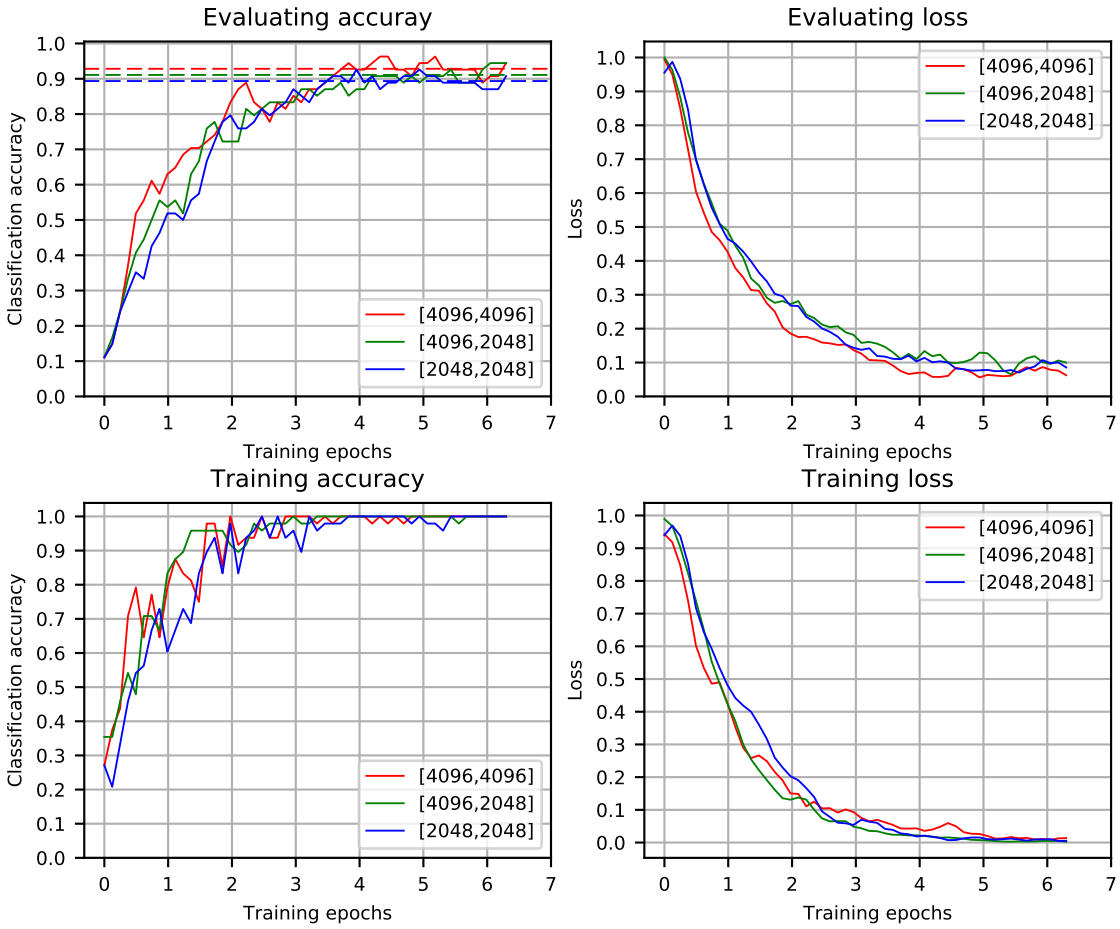


Figure 5.11: The loss and classification accuracy vs. training epochs comparison between networks with different number of output neurons in each fully connected layer.

5.1.3.10 Data augmentation: random cropping

Experimental scheme N-time random cropping is another common used technique to augment small scale training data, see Section 4.1.2 1. We evaluated three networks trained on the data with 3 different random cropping configurations, including 1-time random cropping, 2-time random cropping and 4-time random cropping.

Experimental results and analysis The experimental results are illustrated in Figure 5.13. According the experimental results, the random cropping technique do help much on training the network. The network trained on the data with 4-time random cropping outperforms that with 2-time random cropping, and the network trained on the data with 2-time random cropping outperforms that with 1-time random cropping. So, we use the 4-time random cropping to augment our training data in the project.

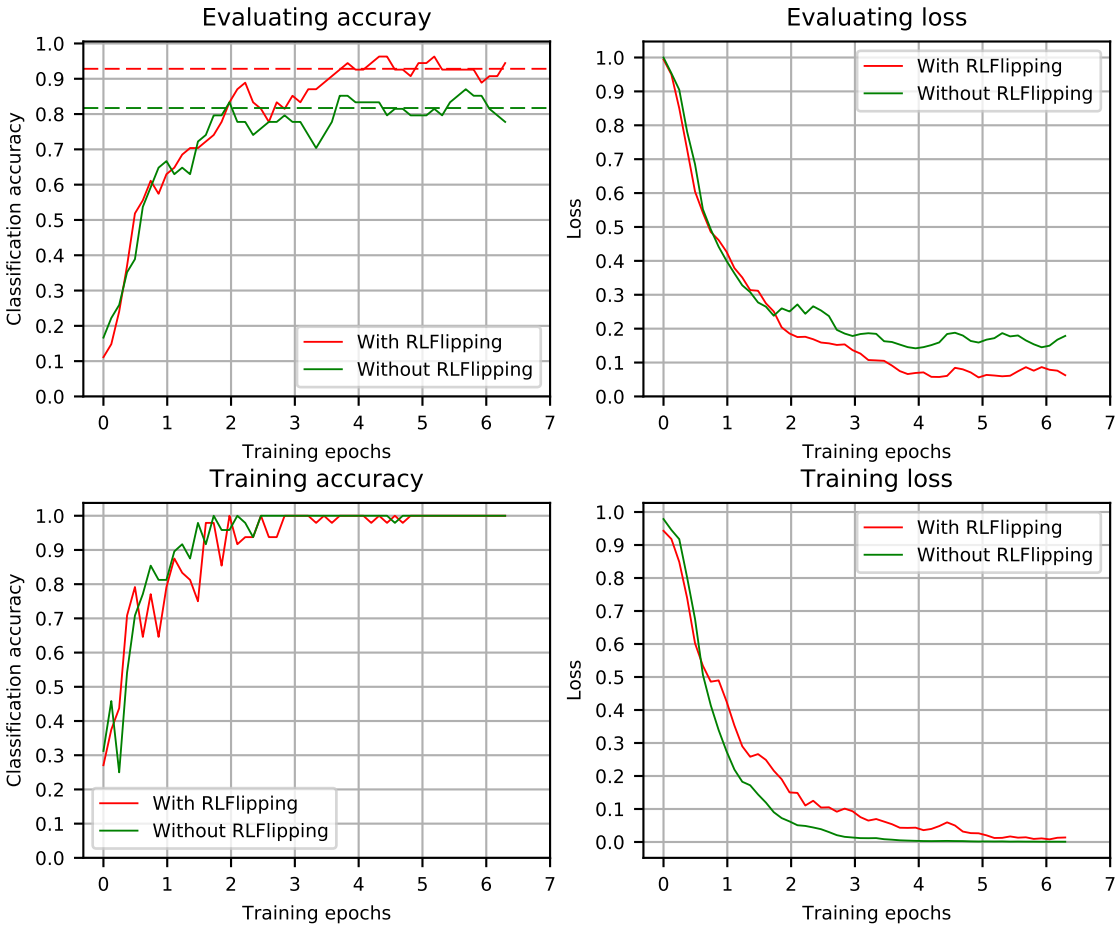


Figure 5.12: The loss and classification accuracy vs. training epochs comparison between networks trained on the data with and without horizontal flipping.

5.1.3.11 Temporal down-sampling of frames

Experimental scheme Temporal down-sampling of frames is a method which we applied to make each video clip contains more temporal information, see Section 4.1.2 2. We evaluated 4 cases: including temporal down-sampling with temporal strides 1 frame, 2 frames, 3 frames and evenly sample 16 frames from each video (temporal down-sampling = 0).

Experimental results and analysis The experimental results are illustrated in Figure 5.14. The network trained on the data which are evenly sampled 16 frames from each input video performs the best. And the other three network have similar performance. So, we set temporal down-sampling mode to 0 to pre-process our training data.

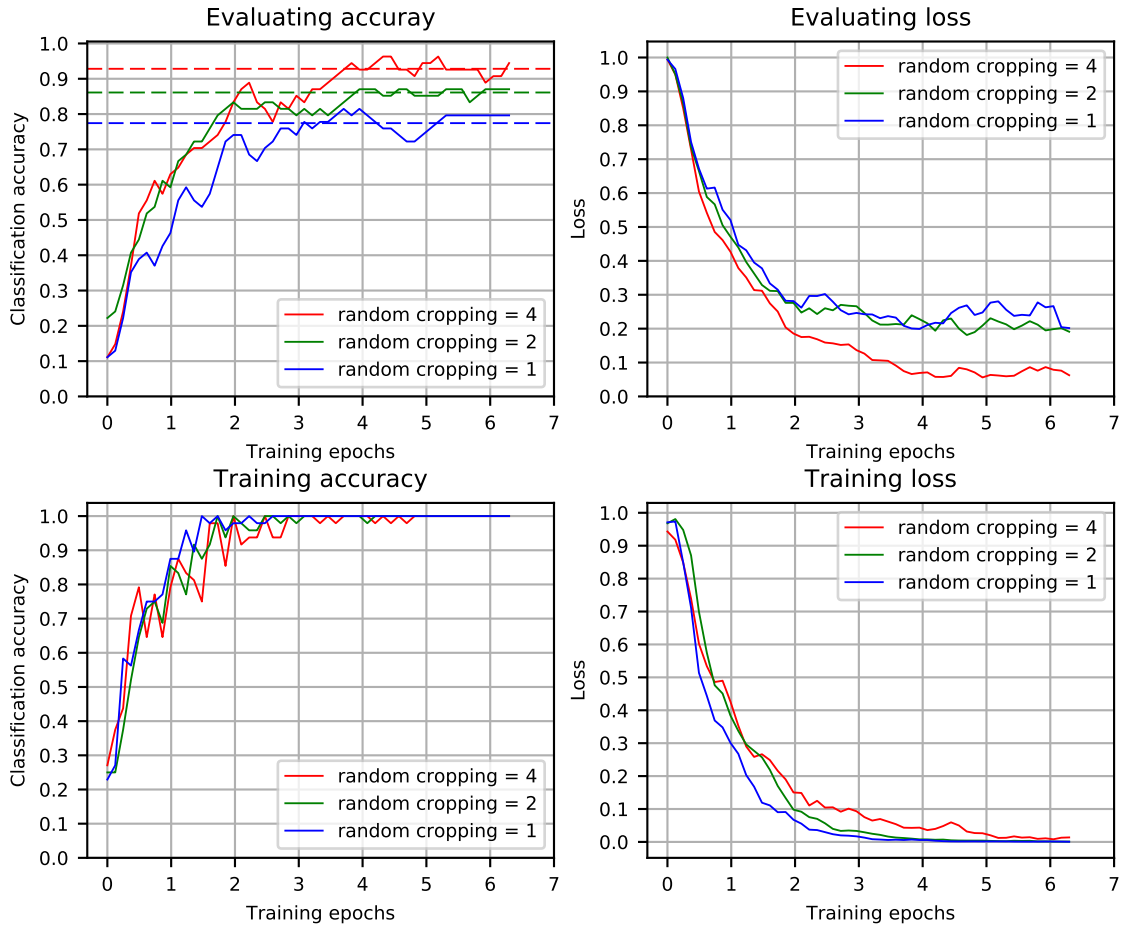


Figure 5.13: The loss and classification accuracy vs. training epochs comparison between networks trained on training data with different random cropping configurations.

5.1.3.12 Data normalization

Experimental scheme Normalizing the data before feeding to the network is a common method, see Section 4.1.2 4. In this experiment, we evaluated three different data normalization methods, including: 1). directly scale the value of X from $0-255$ to $0-1$, 2), subtract the $mean$ value ($X-mean$), 3), subtract the mean value and divide it by the stand deviance value (($X-mean$)/std).

Experimental results and analysis The experimental results are illustrated in Figure 5.15. According to the experimental results, the network trained on the data with normalization mode ($X-mean$) performs best among the three, and the other two networks have similar performance. So, we use the normalization mode ($X-mean$) to normalize the data in our project.

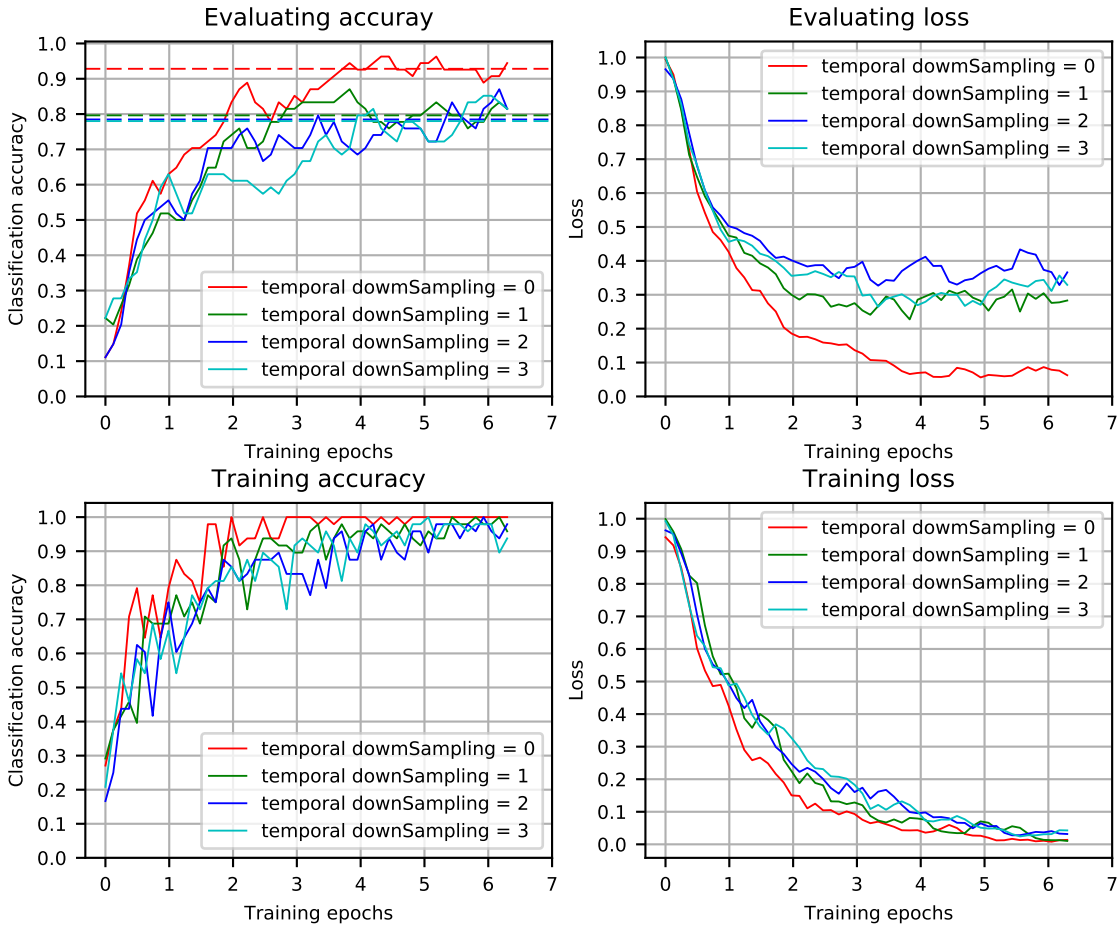


Figure 5.14: The loss and classification accuracy vs. training epochs comparison between networks trained on training data with different temporal down-sampling strides.

5.1.4 Conclusions of parameter searching

According to all of the above experiments which we performed for parameter searching, we can get following conclusions:

1. In the network side, the factors which affect the performance much are not the architecture or the hyper-parameters of the network but those factors in the training processing, including the initialization of network parameters, learning rate and the keep probability of the dropout layers.

2. In the data pre-processing side, almost all factors except the data normalization method affect the performance of the network much. Such conclusion also indicates that the data is the critical factor of our project.

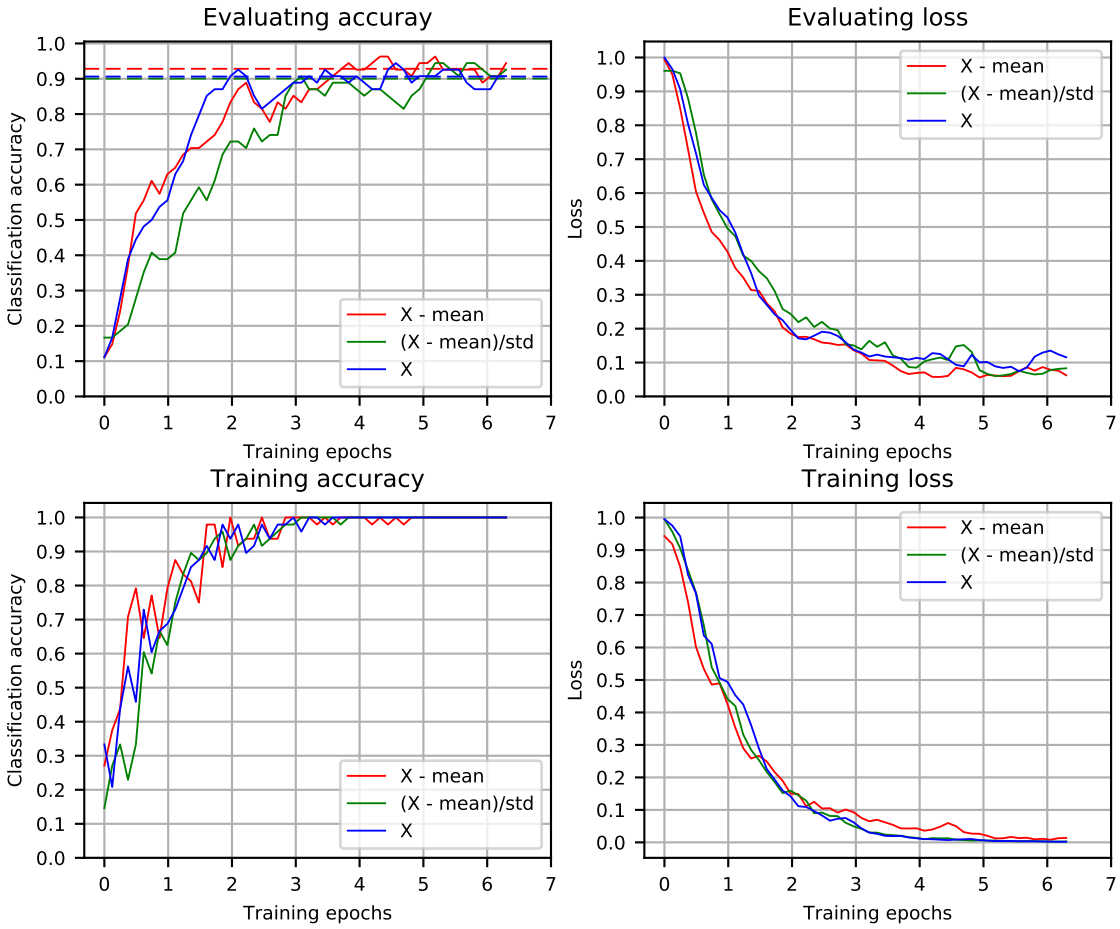


Figure 5.15: The loss and classification accuracy vs. training epochs comparison between networks trained on training data with different normalization methods.

5.2 The experimental results of of the interaction classification

5.2.1 Train the network from scratch

Experimental scheme

In this step, we train the fullNet on UT-Interaction dataset from scratch. We face a new problem when we try to apply all optimal parameter settings from singleNet to fullNet. That is the overall GPU memory allocation requirement for the fullNet is larger than 24GiB which is beyond the GPU resource we have in hand. So, we slightly adjust the hyper-parameters of the fullNet. The adjusted hyper-parameters are the values of noo_fc5 and noo_fc6 , and both from 4096 to 2048. To compare the performance between the fullNet and singleNet, we build three models: a). fullNet with adjusted hyper-parameters notated as fullNet_2048, b). singleNet with optimal hyper-parameters notated as singleNet_4096, and c). singleNet with adjusted hyper-

parameters notated as singleNet_2048. The data pre-processing methods and parameter settings, the architecture and hyper-parameter settings of the network, and the training settings are illustrated in the Table 5.2.

Experimental results

The classification accuracy for these three models are illustrated in Table 5.3.

5.3 The experimental results of of the interaction detection

The goal of interaction detection in UT-Interaction is to locate interactions spatially and temporally in a given un-segmented video which contains 6 interactions in each single video. Since the performance of interaction detection is highly dependent on the quality of the interacting people detection. So, we fist introduce the results of the interacting people detection, then the detail performance of the interaction detection.

5.3.1 The results of the interacting people detection

We don't have the ground truth to report the precision-recall graph directly for the results of interacting people detection. The results are illustrated in Figure 5.16 with bounding boxes annotated for set1. The sequences 1 to 4 are easy since there are only two interacting people appear in the scene, but it is more challenge to locate the positions of interacting people for sequences 5 to 10, because both interacting people and irrelevant pedestrians are present in scene. We adopt the method mentioned in the previous chapter 4.2.3 to exclude the irrelevant people. From the results in Figure 5.16, we successfully locate the spatial positions of interacting people for all videos except the sequence 9, because the relative position between the irrelevant people are very close to the target interacting people and they execute the activities simultaneously.

5.3.2 The results of interaction detection

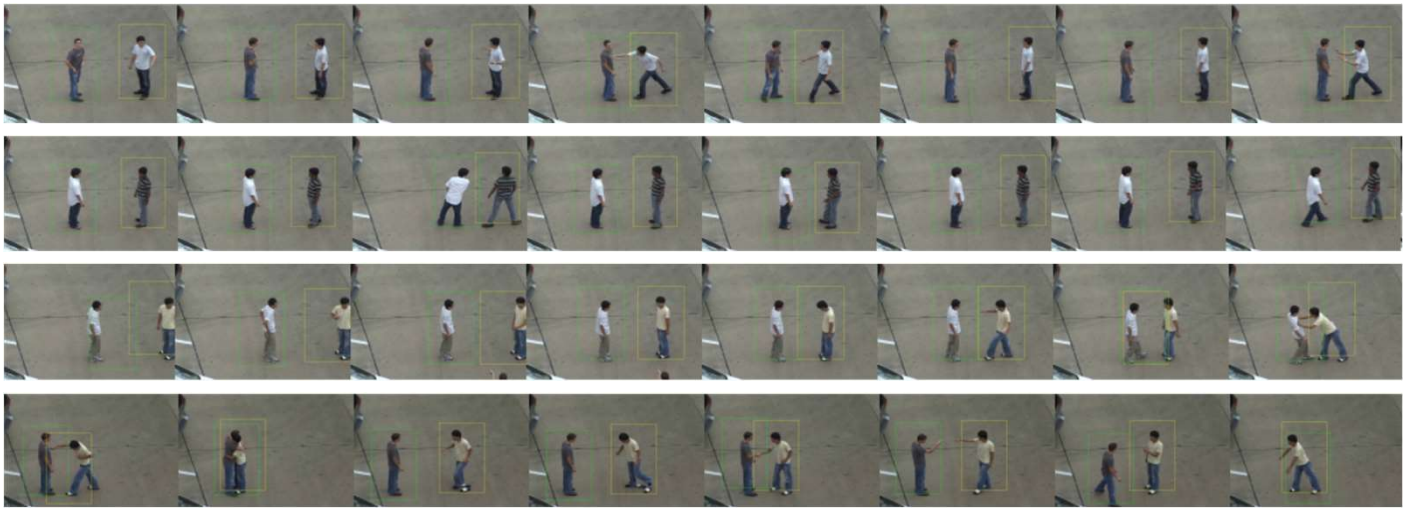
Due to classification accuracy of the singleNet_4096 is slightly better than that of fullNet_2048, we choose the trained singleNet_4096 as the feature descriptor for the interaction detection task.

The results of interaction detection need to be added here.

5.4 Discussion

Table 5.2: Parameter, network architecture and training settings for network and data pre-processing.

No.	Factors	Parameters
Network		
1	Initial values of the network parameters, see Section 4.1.5.	$weights$: "Xavier" initialization $biases$: constant 0.1
2	Dynamic adjusting of the learning rate, see Section 4.1.5.	$Learning\ rate = 1e^{-4}$
3	Dropout layers, see Section 4.1.3	$keep\_ratio = 0.5$
4	Batch normalization layers, see Section 4.1.3	Yes
5	The number of convolutional layers, see Section 4.1.3	5
6	The size of convolutional kernels, see Section 4.1.3	$3 \times 3 \times 3$
7	The sizes of the pooling kernels, see Section 4.1.3	L1: $1 \times 2 \times 2$ L2, L3, L4: $2 \times 2 \times 2$ L5: $2 \times 1 \times 1$
8	The number of filters for each convolutional layer, see Section 4.1.3	nof_conv1 = 32 nof_conv2 = 128 nof_conv3 = 256 nof_conv4 = 512 nof_conv5 = 512
9	The number of output neurons for each fully connected layer, see Section 4.1.3	noo_fc$_*$ = 4096
Data pre-processing		
1	Data augmentation: horizontal flipping, see Section 4.1.2 1	Yes
2	Data augmentation: random cropping, see Section 4.1.2 1	Yes, 4 times
3	Temporal down-sampling of frames, see Section 4.1.2 2	Yes, N=0, see 2
4	Data normalization, see Section 4.1.2 4	Yes, $X_i = X_i - Mean(X)$



a). Sample frames from the results of interacting people detection for seq1 to seq4

b). Sample frames from the results of interacting people detection for seq5 to seq10

Figure 5.16: Sample frames of the results of interacting people detection

Table 5.3: The experimental results of the three models' classification accuracy, evaluated on UT-Interaction set1

Model	Classification Accuracy
fullNet_2048	87.4%
singleNet_4096	88.8%
singleNet_2048	75.6%

CHAPTER 6

CONCLUSION

Appendix A

Begins an appendix

BIBLIOGRAPHY

[1] M. BACCOUCHE, F. MAMALET, C. WOLF, C. GARCIA, AND A. BASKURT, *Sequential deep learning for human action recognition*, Springer, (2011), pp. 29 – 39.

[2] W. CHOI AND S. SAVARESE, *A unified framework for multi-target tracking and collective activity recognition*, ECCV, (2012).

[3] N. DALAL AND B. TRIGGS, *Histograms of oriented gradients for human detection*, CVPR, (2005).

[4] N. DALAL AND B. TRIGGS, *Histograms of oriented gradients for human detection*, CVPR, (2005).

[5] N. DALAL, B. TRIGGS, AND C. SCHMID, *Human detection using oriented histograms of flow and appearance*, ECCV, (2006).

[6] P. DOLLAR, V. RABAUD, G. COTTRELL, AND S. BELONGIE, *Human detection using oriented histograms of flow and appearance*, VS-PETS, (2005).

[7] G. FLITTON, T. BRECKON, AND N. MEGHERBI BOUALLAGU, *Object recognition using 3d sift in complex ct volumes*, Proceedings of the British Machine Vision Conference, (2010), pp. 11.1–11.12.

[8] X. GLOROT AND Y. BENGIO, *Understanding the difficulty of training deep feedforward neural networks*, (2010).

[9] K. HE, X. ZHANG, S. REN, AND J. SUN, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, ICCV, (2015), pp. 1026– 1034.

[10] F. C. HEILBRON, V. ESCORCIA, AND B. GHANEM, *Activitynet: A large-scale video benchmark for human activity understanding*, CVPR, (2015).

[11] S. IOFFE AND C. SZEGEDY, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, (2015).

[12] S. JI, W. XU, M. YANG, AND K. YU, *3d convolutional neural networks for human action recognition*, IEEE TPAMI, (2013).

[13] A. KARPATHY, G. TODERICI, S. SHETTY, T. LEUNG, R. SUKTHANKAR, AND L. FEI-FEI, *Large-scale video classification with convolutional neural networks*, CVPR, (2014).

[14] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, (2015).

[15] A. KLASER, M. MARSZALEK, AND C. SCHMID, *A spatio-temporal descriptor based on 3d-gradients*, British Machine Vision Association, (2008), pp. 275:1–10.

[16] Y. KONG, Y. JIA, AND Y. FU, *Learning human interaction by interactive phrases*, ECCV, (2012).

[17] A. KRIZHEVSKY, I. SUTSKEVER, AND G. E. HINTON, *Imagenet classification with deep convolutional neural networks*, NIPS, (2012).

[18] Y. LECUN, L. BOTTOU, Y. BENGIO, AND P. HAFFNER, *Gradient based learning applied to document recognition*, IEEE, (1998).

[19] D. G. LOWE, *Object recognition from local scale-invariant features*, ICCV, (1999).

[20] ——, *Distinctive image features from scale-invariant keypoints*, IJCV, (2004).

[21] M. MARSZAŁEK, I. LAPTEV, AND C. SCHMID, *Actions in context*, CVPR, (2009).

[22] S. NARAYAN, M. S. KANKANHALLI, AND K. R. RAMAKRISHNAN, *Action and interaction recognition in first-person videos*, CVPR, (2014).

[23] J. Y.-H. NG, M. HAUSKNECHT, S. VIJAYANARASIMHAN, O. VINYALS, R. MONGA, AND G. TODERICI, *Beyond short snippets: Deep networks for video classification*, CVPR, (2015).

[24] F. NING, D. DELHOMME, Y. LECUN, F. P. L. BOTTOU, AND P. E. BARBANO, *Toward automatic phenotyping of developing embryos from videos*, IEEE Trans. on Image Processing, (2005), pp. 1360–1371.

[25] L. W. ORIT KLIPER-GROSS, ORIT KLIPER-GROSS, *The action similarity labeling challenge*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2012), pp. 615–621.

[26] A. PATRON-PEREZ, M. MARSZALEK, A. ZISSERMAN, AND I. REID, *High five: Recognising human interactions in tv shows*, BMVC, (2010).

[27] D. E. RUMELHART, G. E. HINTON, AND R. J. WILLIAMS, *Learning representations by back-propagating errors*, (1986), pp. 696 – 699.

[28] M. S. RYOO AND J. K. AGGARWAL, *Ut-interaction dataset*, (2010).

[29] C. SCHULDT, I. LAPTEV, AND B. CAPUTO, *Recognizing human actions: A local svm approach*, ICPR, (2004).

[30] P. SCOVANNER, S. ALI, AND M. SHAH, *A 3-dimensional sift descriptor and its application to action recognition*, 15th ACM, (2007), pp. 357–360.

[31] K. SIMONYAN AND A. ZISSERMAN, *Two-stream convolutional networks for action recognition in videos*, CVPR, (2014).

[32] K. SOOMRO, A. R. ZAMIR, AND M. SHAH, *Ucf101: A dataset of 101 human actions classes from videos in the wild*, CRCV-TR, (2012).

[33] N. SRIVASTAVA, G. HINTON, A. KRIZHEVSKY, I. SUTSKEVER, AND R. SALAKHUTDINOV, *Dropout: A simple way to prevent neural networks from overfitting*, (2014).

[34] D. TRAN, L. BOURDEV, R. FERGUS, L. TORRESANI, AND M. PALURI, *Learning spatiotemporal features with 3d convolutional networks*, ICCV, (2015).

[35] C. VAN GEMEREN, R. POPPE, AND R. C. VELTKAMP, *Spatio-temporal detection of fine-grained dyadic human interactions*, Human Behavior Understanding. HBU 2016. Lecture Notes in Computer Science, vol 9997. Springer, Cham, (2016).

[36] ——, *Spatio-temporal detection of fine-grained dyadic human interactions*, 7th International Workshop on Human Behavior Understanding 2016, (2016), pp. 116–133.

[37] H. WANG, A. KLASER, C. SCHMID, AND C.-L. LIU, *Dense trajectories and motion boundary descriptors for action recognition*, IJCV, (2013), pp. 60–79.

[38] H. WANG AND C. SCHMID, *Action recognition with improved trajectories*, IEEE International Conference on Computer Vision, (2013).

[39] N. XU, A. LIU, W. NIE, Y. WONG, F. LI, AND Y. SU, *Multi-modal & multi-view & interactive benchmark dataset for human action recognition*, ICME 2015, (2015).

[40] M. D. ZEILER AND R. FERGUS, *Visualizing and understanding convolutional networks*, ECCV, (2014).

[41] Y. ZHANG, X. LIU, M. CHANG, W. GE, AND T. CHEN, *Spatio-temporal phrases for activity recognition*, ECCV, (2012).