

# ROBUST VIDEO EVENT RECOGNITION

by

Feifei Chen

Bachelor of Engineer, Zhejiang University, 2012

A thesis

presented to Ryerson University

in partial fulfillment of the  
requirements for the degree of  
Master of Applied Science  
in the Program of  
Computer Networks

Toronto, Ontario, Canada, 2016

©Feifei Chen 2016



I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.



Robust Video Event Recognition

Master of Applied Science 2016

Feifei Chen

Computer Networks

Ryerson University

## **Abstract**

Video Event Recognition is an important problem in video semantic analysis. In Video Event Recognition, video scenes can be summarized into video event through understanding their contents. Previous research has proposed many solutions to solve this problem. However, so far, all of them only target high-quality videos. In this thesis, we find, with the constraints in modern applications, that low-quality videos also deserve our attention. Compared to previous works, this brings a greater challenge, as low-quality videos are lacking essential information for previous methods to work. With the quality degraded, technical assumptions made by previous works no longer stay intact. Thus, this thesis provides a solution to address this problem. Based on the generic framework proposed by previous work, we propose a novel feature extraction technique that can work well with low-quality videos. We also improve the sequence summary model based on previous work. As a result, comparing to previous works, our method reaches a similar accuracy but is tested with a much lower video quality.



## Acknowledgements

I would like to thank all the people who have supported me during the completion of this thesis. Without their help, I would never complete this thesis. Meanwhile, I sincerely appreciate all the people who have helped and supported me during my graduate study in Canada. Their help made my study here an enjoyable experience.

First and foremost, I would like to express my deepest gratitude to my supervisor Dr. Xiao-Ping Zhang, whose expertise and enthusiasm for research set an excellent example for me. I appreciate all his inspiration, understanding, and patience. Thank you for providing me such a great research atmosphere and thank you for sharing knowledge with me.

Also, I am really grateful to Dr. Bobby Ma, Dr. Alagan Anpalagan, and Dr. Lian Zhao. Thank you for the time, efforts, and contributions to my defense as well as this work. For this work, I really appreciate JunFeng Jiang and Mike Qin for providing training data. Manually recognizing video event is time-consuming and, yet, it is the only way we can calculate the accuracy rate.

It has been a pleasure working with all my colleagues in Communications and Signal Processing Applications Laboratory Lab (CASPAL), who are always willing to discuss and share their knowledge with me.

A very special thanks goes to Dr. Bobby Ma, without whose professional advice and encouragement, I would not have been able to complete my graduate study in the Computer Networks program. Thanks again for giving me a chance to enjoy here and meet with interesting people.

Finally, my heartfelt thanks to my parents for their continuous support, understanding and encouragement.





# Contents

<i>Declaration</i>	iii
<i>Abstract</i>	v
<i>Acknowledgements</i>	vii
<i>List of Tables</i>	xiii
<i>List of Figures</i>	xv
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Objective	1
1.2 Example Applications	3
1.2.1 Mobile Vision Video Event Recognition	3
1.2.2 Cloud Environment Video Event Recognition	4
1.2.3 Surveillance Video Event Recognition	5
1.3 Review of Previous Works	5
1.3.1 ICAMHMM Video Event Recognition	5
1.3.2 ICAMHCRF Video Event Recognition	6
1.3.3 Sub-Graph Based Video Event Recognition	6
1.4 Our Insight and Main Contributions	7
1.5 Organization of Thesis	8
<b>2 Background</b>	<b>9</b>
2.1 Problem Formulation	9
2.1.1 Pre-processing	10
2.1.2 Problem Statement	10
2.2 General Framework by Previous Work	11
2.3 Technical Details in State of Art Work	12

2.3.1	Feature Extraction . . . . .	12
2.3.2	Sequence Summary Model . . . . .	14
2.3.3	Summary and Limitations . . . . .	16
2.4	Existing Tools in This Thesis . . . . .	17
2.4.1	Sparse Representation and Sparse Coding . . . . .	17
<b>3</b>	<b>Self-Taught Feature Extraction</b>	<b>19</b>
3.1	The Need for Self-Taught Feature Extraction . . . . .	20
3.2	Definition of Feature Function . . . . .	21
3.3	General Process . . . . .	21
3.4	Feature Training . . . . .	22
3.5	Feature Extraction . . . . .	25
3.6	Chapter Summary . . . . .	28
<b>4</b>	<b>Hidden States Sequence Summary Model</b>	<b>29</b>
4.1	Definition of Sequence Summary Model . . . . .	30
4.2	Introducing Hidden States . . . . .	31
4.3	Constructing Hidden States . . . . .	33
4.3.1	Mathematical Property of Hidden States . . . . .	33
4.3.2	Physical Meanings of Hidden States . . . . .	34
4.4	Feature Extraction with Hidden States . . . . .	35
4.4.1	Feature Training with Hidden States . . . . .	36
4.4.2	Feature Function with Hidden States . . . . .	38
4.5	Chapter Summary . . . . .	39
<b>5</b>	<b>Experimental Results</b>	<b>41</b>
5.1	Implementation . . . . .	41
5.2	Golf Video Benchmark . . . . .	42
5.2.1	Video Data Description . . . . .	42
5.2.2	Overall Results . . . . .	42
5.2.3	Confusion Matrix . . . . .	44
5.2.4	Hidden States Results . . . . .	44
5.2.5	Summary of Golf Video Benchmark . . . . .	45
5.3	Table Tennis Teaching Video . . . . .	45

5.3.1	Video Data Description . . . . .	45
5.3.2	Overall Results . . . . .	46
5.3.3	Summary for Table Tennis Teaching Video . . . . .	48
<b>6</b>	<b>Conclusion and Future Work</b>	<b>51</b>
	<b>Bibliography</b>	<b>54</b>



# List of Tables

2.1	Example Video Events in Golf Videos . . . . .	10
3.1	Comparison of Video Qualities between Previous Work and Our Goal . .	20
5.1	Video Quality Degradation . . . . .	43
5.2	Accuracy Comparison . . . . .	43
5.3	Confusion Matrix . . . . .	44
5.4	Accuracy with respect to number of hidden states . . . . .	45
5.5	Video Quality Degradation for Table Tennis Video . . . . .	46
5.6	Events of Table Tennis Teaching Video . . . . .	47
5.7	Confusion Matrix for Table Tennis Teaching Video . . . . .	48



# List of Figures

2.1	Example of Video Event Recognition Problem Statement . . . . .	11
2.2	Graphical Representation of HCRF . . . . .	15
2.3	Graphical Representation of Simplified Model in Previous Work . . . . .	16
3.1	General Process of our Feature Extraction Method . . . . .	22
3.2	Feature Training Process . . . . .	23
3.3	Feature Extraction Process . . . . .	27
4.1	Example of Video Sub-Event . . . . .	35





# Chapter 1

## Introduction

### 1.1 Motivation and Objective

There are many ways to analyze the semantic of a video. Among them, Video Event Recognition has gained a lot of attention in recent years [1, 2, 3, 4, 5, 6]. Video Event Recognition is a scene recognition problem. As a prerequisite, videos are first preprocessed into a set of scenes. Each of these video scenes has a semantic meaning. Video Event Recognition recognizes which type of semantic the scene expressed by looking at the content of the scene.

Video Event Recognition has many application scenarios, making it a very worthwhile problem to solve.

Some previous works[3, 4] use Video Event Recognition to recognize sporting events in sports videos. These works both suggest that Video Event Recognition can accurately recognize sporting events. Experiments have showed that Video Event Recognition can accurately process ice hockey, golf, and bowling videos. Video events are recognized and classified by the content of the video, which is the technical actions of the sport movements and features. Taking golf videos as an example, video events are classified into long drive (long shot), putting (short shot), and non-related (video events which are not part of golf sports) events. With these recognized events, it is helpful to use these statistics as feedback for athletes to improve their skills.

Aside from sports videos, surveillance videos are also an application scenario for Video Event Recognition. Recognized events could reflect the status of the office: idle, busy

or meeting. This can be used to collect statistic information such as office utilization, average work hours, and etc. Previous research also works on traffic surveillance videos [2] to detect abnormal traffic activities.

From these past research, the data set and the accuracy has been widely focused on high-quality videos. These videos are at least television quality, and some of them are even higher quality videos, such as 720p. Therefore researchers assume target videos sharing similar characteristics: clear resolution, clear texture, and colors. Most of these assumptions remain true for sports videos at a television broadcast company, and in fact, at such scenario, the qualities of these videos would get even higher with the trend of modern video technology.

However, low resolution videos also deserve our attention. In a lot of scenarios, video decoding/encoding and network transmission are constrained by energy consumption and storage devices. These scenarios often occur with battery backed video capture devices (such as smart phones, smart watches, or smart glasses); or long span videos (like days of surveillance video) with a limited storage. In these cases, videos are produced and transferred with very limited resources, and as a result, videos are of very low-quality. We define videos that have the following properties as low-quality videos:

- They are greyscale instead of color
- They have very limited resolution
- They have very low frame rate

Most of low-quality videos are greyscale videos since these videos are often much smaller than color videos. Limited resolution and low frame rate could result in a even smaller video file, and moreover, according to some studies[7], it can significantly reduce energy consumption when capturing the video. For these videos, the limitations of previous work begin to expose: thus, they often need good quality videos to make it work.

While a lot of other existing computer vision algorithms are able to work with these low-quality videos[8, 9, 10], to the best of our knowledge, Video Event Recognition is still at the stage of assuming high-quality videos to extract features and do further processing. How to recognize video events from low-quality videos is a major problem in these scenarios.

This research is aiming at making Video Event Recognition system work under lowquality videos. By being able to process videos with much lower quality, this will open Video Event Recognition to a wider set of videos. This is a challenging task since the technical assumptions made by previous work on high-quality videos no longer stand. Low-quality videos contain less information than high-quality videos, therefore, making new technical assumptions is much harder than before.

## **1.2 Example Applications**

The impact of this thesis will benefit many modern applications. Moreover, these applications all have external constraints during video capturing or encoding, which forces Video Event Recognition system to deal with low-quality videos. Here we list some examples of such applications.

### **1.2.1 Mobile Vision Video Event Recognition**

There is an evidence that mobile devices are becoming more and more popular in the past decade. Starting from mobile phones, other cutting edge mobile devices such as smart watches, smart glasses are also part of the mobile device family. These devices can capture videos at any time, and are much more convenient than professional high resolution cameras. With these devices, mobile computer vision becomes possible. However, these devices usually have very limited computation, energy, and network resources.

Take a famous mobile video device as an example: Google Glass. Google Glass is equipped with an OMAP 4430 processor, 2GB RAM and only a 570mAh battery. Although Google Glass has a decent camera, it is impossible to do any HD-video processing on this device. In the mobile vision environment, video technology is not the bottleneck, instead, the hardware resources is.

However, a recent research[7] suggests it that these types of devices are still capable of processing videos. By tweaking the image sensor and software system, this kind of mobile device can process very low-quality, greyscale videos. This makes video processing applications working on a portable device. The assumption made by this paper [7]is that some video processing algorithms are robust and fast enough to process low-quality videos, including object detection, face recognition, etc.

The objective of this thesis is focusing on bringing Video Event Recognition to low-quality videos. With the work done in this thesis, it would be possible to recognize video event on a portable mobile device, opening a new possibility for Video Event Recognition.

### 1.2.2 Cloud Environment Video Event Recognition

With the cloud environment becoming more and more common, large amount of computation and storage resources are available for video processing algorithms. This certainly can help some video processing algorithms, such as Video Event Recognition, being faster and/or more accurate.

The setup of this cloud environment is different than previous ones as well. Notice that the cloud is connected with a massive number of thin-clients: most of these clients have very weak computation resources such as laptops, mobile devices or even tiny sensors. Unfortunately, video capturing, encoding, and even storage are originated from these thin-clients, while video processing algorithms are running on the cloud.

In order to bridge the gaps between clients and cloud, developers usually do code offloading[11, 12, 13, 14]. Code offloading means that the clients upload the input to some part of code which could be running on the cloud. In the case of video processing, the inputs are the videos that are captured by the clients, and the clients need to upload the videos to the cloud for further processing.

Compared to other types of data flow in the network, videos are usually considered very large to upload. First of all, from the data center's perspective, this kind of data costs a great deal of network resource usage. Second, from client's perspective, such a large amount of network transmission is often very expensive under mobile network. Meanwhile, the larger the data is, the slower it is to upload. Usually, videos are first degraded to low-quality videos for uploading.

Therefore, for the video processing algorithm, all the input videos to the cloud are low-quality. With the work done in this thesis, Video Event Recognition is capable of processing low-quality videos arriving at the cloud. This will open the Video Event Recognition algorithm to the modern computational infrastructure.

### 1.2.3 Surveillance Video Event Recognition

Surveillance videos are another type of videos that are often low-quality. This is due to two reasons. First, surveillance camera is recording images far away from the target. The further the target is, the more blurry the image captured by the camera is. Second, surveillance cameras usually generate days of surveillance videos, creating a storage space bottleneck. As surveillance storage is less maintained, users want their surveillance storage to last long enough before it gets full and needs to be replaced.

With the work done in this thesis, video events in the surveillance videos could be easily recognized, regardless of the low-quality which results from surveillance equipment's limit.

## 1.3 Review of Previous Works

### 1.3.1 ICAMHMM Video Event Recognition

ICAMHMM[3] (Independent Component Analysis Mixture Hidden Markov Model) is proposed by Jian, et al. It is one of the early ground-breaking works aimed at Video Event Recognition. It proposes a two-step framework for Video Event Recognition: feature extraction and sequencing summary model. In both of the two steps it proposed, it uses novel techniques to approach each step.

In feature extraction, by observing the color histogram distributions of some sports videos, the authors observe that the distribution does not follow a Gaussian distribution, which most of previous work assumes. Therefore, in the feature extraction step, the paper uses ICA[15] (Independent Component Analysis) as a feature extractor. By running ICA on color histograms, the paper is able to extract coefficients for each independent components, and use these coefficients as the feature.

In the sequence summary model phase, the paper uses HMM (Hidden Markov Model) as a sequencing model. HMM is a statistical model. The idea of using statistical model is novel, because that allows the sequence summary model to learn from existing data. Before recognizing video events, it uses some part of data to train the sequence summary model. This allows the sequence summary model to be adaptive: the model can adapt to the data set.

This is an early stage contribution towards Video Event Recognition. The solution presented in this paper has a lot of limitations. First, the policies it chose might not be optimal. For example, the follow up work proposes a new model ICAMHCRF, that out-performs this work by simply switching the sequence summary model. Second, the method is known to break under low-quality videos. The technical assumptions made by the authors are by manually observing the patterns in the testing video, which made this merely a domain specific knowledge rather than a model. These technical assumptions might break when the testing video changes. For example, in the scope of this thesis, the videos are low-quality videos, and these assumptions break.

### 1.3.2 ICAMHCRF Video Event Recognition

ICAMHCRF[4] is an improvement based on previous ICAMHMM work. It follows the two-step framework that ICAMHMM paper presented, and also it uses the same feature extractor – ICA as the first step.

The second step, known as the sequence summary model, instead of using HMM (Hidden Markov Model), it uses HCRF (Hidden States Conditional Random Field) model. HMM is a generative model. It often suffers from over-fitting and sparseness issue. HCRF is a smooth, regression model, which solves some issues of HMM. In general, HCRF model often out-performs HMM model.

This paper suggests that simple improvements on the sequence summary model can out-perform the ICAMHMM model. It shows that in ICAHMM model, some of choice of policies are not optimal, and by optimizing these policies, accuracy of Video Event Recognition can increase dramatically. However, it still ignores the limitations of the ICAMHMM work. It does not improve the feature extraction phase, and the feature extraction is still based on the author’s domain specific knowledge. At the same time, this work does not fully exploit the full potential of the HCRF model. For example, in this work, there is no multiple hidden states.

### 1.3.3 Sub-Graph Based Video Event Recognition

A more recent work[6] suggests that instead of considering the time sequence of the videos, another solution to Video Event Recognition is to construct feature points and to

track how these feature points move. It follows a three-step procedure. The first step is to extract features. Unlike ICAMHMM work, it does not extract feature scalar, instead, it extracts feature points or key points in the frame. The second step is to construct a graph to see how these points move. Third step is the query step. The query step will check the similarity between graphs to recognize the type of events in the videos.

In the feature extraction step, it uses traditional SIFT algorithm to detect feature points. In the graph construction phase, it constructs both per-frame graph and timeline graph, and then uses a novel graph-comparing technique to recognize the graph and the video events.

However, this method does not improve significantly compared to ICAHMM method, and it under-performs compared to the ICAHCRF method. This method is not a statistical model either, making it hard to adapt to data set changes. A traditional graph model replaces the sequence summary model, which makes the work a bit backwards to modern tools and techniques. This work is also hard to improve, as SIFT, the feature extractor, is a black box, rather than a model. This means that, when SIFT fails to recognize good feature points, it is impossible to improve it.

## 1.4 Our Insight and Main Contributions

In Section 1.1, we mention that our objective is to recognition video event for low-quality videos, and we explained the major challenge for such objectives is to find new technical assumptions from low-quality videos. However finding new technical assumptions from low-quality videos is much harder than before, for there is less information inside them for us to observe.

This thesis gives an answer to this challenge: it is still possible. By looking at these videos, humans are still able to recognize the events easily, so should computers. Fortunately human vision system is better understood in recent research[16, 17, 18, 19]. The insight of this thesis is: algorithms should borrow ideas from human vision system because human vision system is able to recognize video events in very low-quality videos.

The major contributions of this thesis are the following.

**Self-taught Feature Extraction** Inspired from unsupervised learning method and human vision mechanism discovered by neuroscientist[19]. This thesis proposes a new

feature extractor that uses Sparse Coding to learn features. Instead of letting the video experts be the feature extractor, the feature extraction in this thesis can make the computer teach itself to extract features.

**Introducing Hidden States for Sequence Summary** Although previous research [4] uses HCRF model, it does not leverage the hidden states for sequence summary. In fact, in previous work, the number of hidden states is one, or, in other words, there is no hidden state at all. This thesis shows, how to leverage hidden states for Video Event Recognition in HCRF model, and how hidden states improve the overall accuracy.

With our solution, we can achieve similar accuracy comparing to previous works, which are evaluated on high-quality videos. Our solution only need low-quality videos.

## 1.5 Organization of Thesis

In Chapter 2, we describe the background of this thesis. We first formalize the Video Event Recognition problem by introducing formalized concepts and notations. Second, under the well specified problem formulation, we discuss some technical details of previous works. Third, we introduce some of the preliminaries of our work, including the existing tools and frameworks we rely on.

In Chapter 3, we describe in detail on how our feature extraction works, why it works, and why it is self-taught. We cover the details in both feature training and feature extraction.

In Chapter 4, we introduce our improved sequence summary model. We introduce the existing model used in ICAMHCRF work, and how we make improvements based on it. We also highlight that some of these improvements need to change the feature extraction as well.

In Chapter 5, we conduct experiments to prove that even though the video we are processing is low-quality, we can still get similar accuracy compared to current state of art work. We also evaluate our method on a newer data set, showing that our method can be adapted to other types of videos.

In Chapter 6, we conclude our work and discuss future work for this research.



# Chapter 2

## Background

Video Event Recognition with low-quality videos presents a great challenge, since the technical assumptions made by previous work no longer hold. In this chapter, we present some backgrounds about Video Event Recognition, and also some technical assumptions made by the previous work. In the last part of this chapter, we will also introduce some of existing tools and frameworks we use.

### 2.1 Problem Formulation

Defined from an application level, video events are semantic moments in a video. A moment is a scene in the video that has semantic meanings related to video contents. Taking sports videos as an example, the video events would be the movements within the game events: such as a player making a shot; or a team that makes a goal.

In technical terms, video consists of scenes. Each scene in the video will have a video event to describe its content. This video event is pre-selected from a pool of possible video events. Video Event Recognition system gives each scene a video event, selecting from a pool of predefined video events. For example, in golf sports video, a video event could be one of long drive, putting, and non-related (others). Table 2.1 gives a short example of a video and there are three kinds of video events in it. In this way, with the video event for each scene, we can tell the semantics of each scene.

In the following section, we will discuss Video Event Recognition system in more detail. Firstly, what is the input to Video Event Recognition system. Secondly, how does

		
long drive	putting	non-related

Table 2.1: Example Video Events in Golf Videos

the system works internally.

### 2.1.1 Pre-processing

Video Event Recognition recognizes at a scene level. As a prerequisite, a full video will be, first, cut into scenes. This is usually referred as the pre-processing stage in Video Event Recognition [20]. This process does not involve any video semantic recognition. It simply cuts the videos into scenes by looking at the transition between frames. If the transition is large enough, then it indicates that this might be a new scene. Previous works have presented several methods [20, 21, 22, 23] to do this automatically. A scene is the basic granularity for video events. Each scene will be recognized into one type of video event. Video Event Recognition system is going to recognize these scenes by looking at its content.

### 2.1.2 Problem Statement

As previous section mentions, the input of Video Event Recognition is a video scene. A scene consists a lot of frames. To begin, let's denote each frame  $i$  in the scene as  $x_i$ , and the scene itself as  $X = \{x_1, x_2, \dots\}$ . In this thesis, we generally use a vector of all pixels in the frame to represent frame  $x_i$ , as defined in detail in section 2.3.1. However in the problem statement, without loss of generality, frame  $x_i$  could be considered as a symbol representing the image in the frame. The task of Video Event Recognition is to give a video event, which is denoted as  $y$ , to scene  $X$ . We pre-select a set of all possible video events of the video as set  $\mathcal{Y}$ , which  $y \in \mathcal{Y}$ .

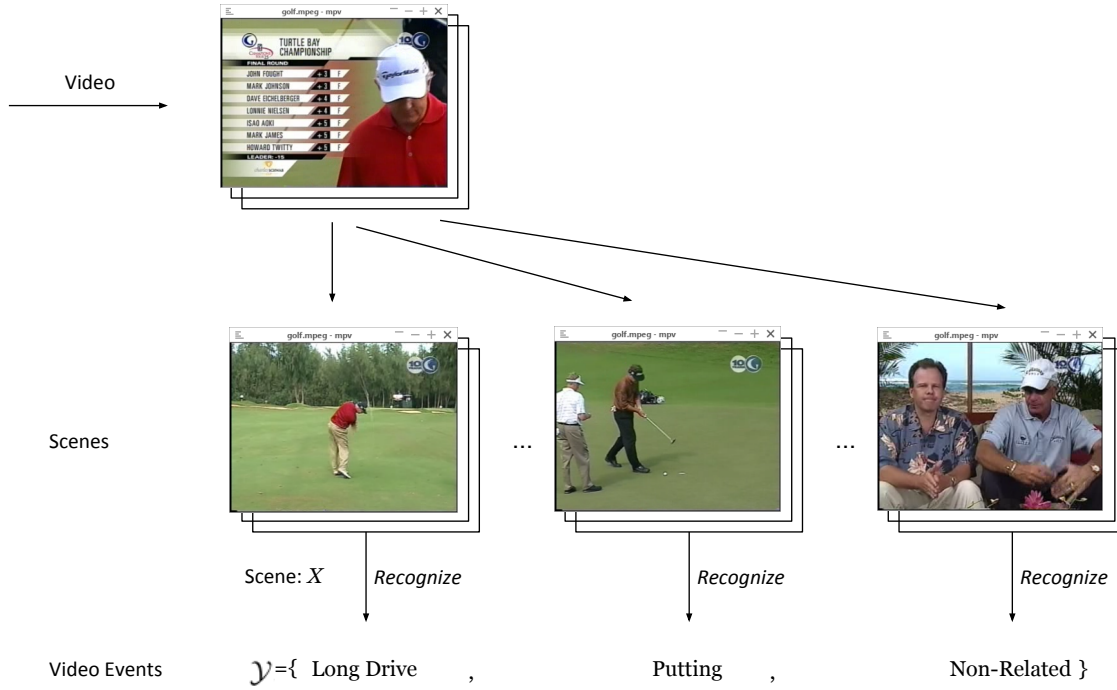


Figure 2.1: Example of Video Event Recognition Problem Statement

In order to give a video event for scene  $X$ , current state of art approach is to calculate conditional probability: for each  $y \in \mathcal{Y}$ , we calculate  $P(y|X)$ . This means, under the condition of such scene appearing, the probability of this scene being video event  $y$ . Therefore, the Video Event Recognition system is to find  $y$  with the maximum  $P(y|X)$  given scene  $X$ .

Figure 2.1 gives an example of Video Event Recognition. A video is first cut into scenes, and each scene will describe one type of content. The task for Video Event Recognition is, given each scene, we need to recognize the video event from a set of preselected events  $\mathcal{Y}$ . In this figure,  $\mathcal{Y} = \{ \text{Long Drive}, \text{Putting}, \text{Non-Related} \}$ .

## 2.2 General Framework by Previous Work

In this section, we show some technical details of previous work. According to the problem statement we mentioned in Section 2.1.2, we now show that how previous work solve the problem – how to model  $P(y|X)$  and how to find the  $y$  such that  $P(y|X)$  is optimal.

Current state of art work [4] presents a framework for Video Event Recognition. It consists of two stages: feature extraction and sequence summary model. Feature extraction stage is to identify the semantic of each video frame,  $x_i$ ; sequence summary model is to provide a way to model the conditional probability  $P(y|X)$  given a sequence of frame, scene  $X$ .

The first step is feature extraction. Feature extraction is to transform the video frame from pixel space into a higher semantic level. We denote frame  $i$  in a scene as  $x_i$ , and  $y$  as a type of video events, and  $y \in \mathcal{Y}$ , where  $\mathcal{Y}$  is a set of all possible video events. The objective of extracting features from frame  $x_i$  is to find a set of feature functions ( $K$  of them in total)  $f_k(x_i, y)$  ( $k = 0, 1, 2, \dots, K - 1$ ), which represent how likely that frame  $x_i$  is video event  $y$ .

With these feature functions generated from the first step, the second step summarizing a sequence follows. We can use feature functions to describe a frame in the scene. These descriptions, as a sequence of frames in a scene, also forms a sequence. Each feature function describes how close it is between each frame and each video event. A sequence of these features need a summary. Therefore we define, by using feature functions  $f_k(x_i, y)$  for all  $x_i$  and  $y$ , we need to model the conditional probability  $P(y|X)$ , where  $X = \{x_1, x_2, \dots\}$  is a scene or a sequence of frames.

## 2.3 Technical Details in State of Art Work

This section shows some technical details of current state of art work. Current state of art work [4] follows the 2 step process framework: feature extraction and sequence summary model.

### 2.3.1 Feature Extraction

#### Independent Component Analysis

In the above problem statements, video frame  $x_i$  is an image. Image consists of pixels, but pixel space data level is not high enough to express video semantics. Therefore, we need a transformation from pixels level into a higher level data expression. The transformations here are called feature extraction.

Feature extraction for video frame is usually conducted under some transformations. In current state of art work [4], pixels are first transformed into a 2D color histogram matrix  $H_i$ .

$$H_i[r][g] = \frac{\#pixels \text{ with color } (r, g, \_)}{total \# \text{ pixels}} \text{ where } 0 \leq r, g < 256$$

So the size of matrix  $H_i$  is  $256 \times 256$ . Any further transformations are based on this color histogram matrix  $H_i$ .

Next step, previous work shows that the pattern in those color histogram still follows locality, but exhibits non-Gaussian characteristics. This lead researchers to decompose the low level histogram matrix  $H_i$  to get high level representation. Current state of art work [4] performs ICA [15] (Independent Component Analysis) decomposition.

Suppose video frame  $x_i$  is decomposed to the  $k$  components  $C_k$ , we write:

$$H_i = M_k s_k + \mu_k, \forall k \text{ where } x_i \in C_k \text{ } 0 \leq k < K$$

This means, whenever a frame was decomposed into a set of components, it will be expressed with:  $M_k$  the mixture matrix;  $s_k$  the coefficient to this mixture; and  $\mu_k$ , the error to this mixture. Each of these parameters will be independent to each other. Given this, consider the following probability:

$$P(x_i|y) = \sum_k P(x_i|yC_k)P(C_k|y)$$

This probability means given the video event  $y$  we assume to put on the whole scene, what is the probability for this frame  $x_i$ ? To calculate this, since  $x_i$  is fully decomposed, we can sum up the conditional probability in all components. Notice that all parts before the sum are independent from ICA's perspective. Inspired from this, researchers come up with  $K$  feature functions. For each frame  $x_i$ , feature function which represents  $x_i$  at a higher level is:

$$f_{i,k}(y, x_i) = \log(P(x_i|yC_k)P(C_k|y)) = \log(s_k) - \log(|M_k|) + \log(P(C_k|y))$$

The feature function consists of two parts. First part is  $P(x_i|yC_k)$ , which is related

to the frame content and can be represented through ICA into  $\frac{s_k}{|M_k|}$ . The second part is the static weight of function  $f_{i,k}$ ,  $P(C_k|y)$ , which is learned via a iterative process.

This is the first stage of Video Event Recognition system. Now the frame feature function itself represents video semantic. However, what we want is the video semantic over a sequence of these frames. How to calculate the final conditional probability  $P(y|X)$  over a sequence frame,  $X$ . This will lead us to the next stage: sequence summary model.

### 2.3.2 Sequence Summary Model

Section 2.2, mentions that the second step of the general frame work is to model  $P(y|X)$  for all video event  $y \in \mathcal{Y}$ . Current state of art work uses HCRF [24] (*Hidden state Conditional Random Field*) to model this.

#### Hidden States Conditional Random Field

HCRF is a regression model: it gives a generic equation on how the final probability would be, and the equation contains number of parameters. In the training process, an optimizer will find the best parameters with the lowest error rate. In this way, the equation for probability fits with the training data. HCRF models the probability:

$$P(y|X) = \frac{1}{Z(X)} \sum_{h \in \mathcal{H}} \exp(\psi(y, h, X))$$

, where  $Z(X)$  and  $\psi(y, h, X)$

$$Z(X) = \sum_{y \in \mathcal{Y}} \sum_{h \in \mathcal{H}} \exp(\psi(y, h, X))$$

$$\psi(y, h, X) = \sum_{i, x_i \in X} \left( \sum_{k=1}^K \lambda_k f_k(y, h_i, X) + \sum_{j=1}^J \tau_j g_j(y, h_{i-1}, h_i, X) \right)$$

$f$  and  $g$  are the feature functions. Their meanings are: for each frame  $i$ , the higher the function value are, the more likely that frame  $i$  with hidden state  $h_i$  is video event  $y$ .  $\lambda_k$  and  $\tau_j$  are the parameters of feature functions, and their values depend on the training data.  $X$  is independent variable, and  $P(y|X)$  is the dependent variable. Video Event Recognition system calculate  $P(y|X)$  for recognition purpose during testing. Meanwhile,

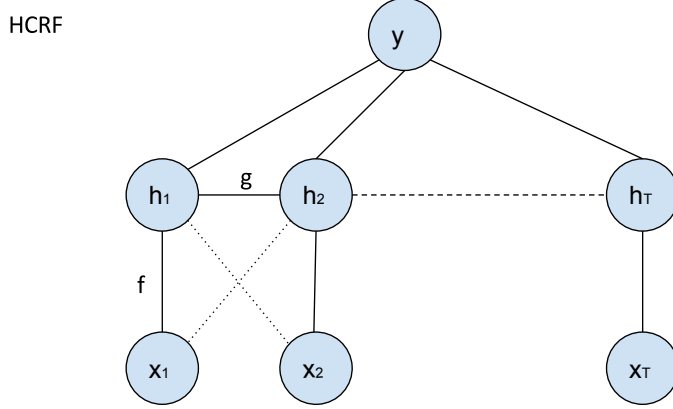


Figure 2.2: Graphical Representation of HCRF

we have to perform a regression to estimate parameter  $\lambda_k$  and  $\tau_j$ . This is usually referred as optimization phase in HCRF.

Figure 2.2 shows the graphical representation of HCRF. HCRF summarizes the frame sequence in video scene  $X$  into a video event  $y$ . HCRF model introduces hidden states  $h$  as intermediate parameters. Classical HCRF model has two kinds of feature functions  $f$  and  $g$ . The structure of HCRF enables HCRF to consider the correlations between video frames when making the summary.

HCRF performs maximum likelihood for optimization. During the training process, assume we know the video scene  $X$  is video event  $y$ . We calculate the likelihood  $\mathcal{L}$  of such input with the unknown parameters, and use  $\mathcal{L}$  as a target function to maximize. When  $\mathcal{L}$  is at the maximum value, we choose the parameters as our best parameters.

Current state of art work [4] uses a modified HCRF model in sequencing model. First, it removes the *state transition function*  $g$ . That is to say, feature function will not take previous hidden states into account. Second, it removes hidden states completely for simplicity. This speeds up the learning and training process, and had very limited effect under high-quality videos due to its high frame rate. Therefore, the model used in current state of art work is the following.

$$P(y|X) = \frac{1}{Z(X)} \exp(\psi(y, X))$$

, where  $Z(X)$  and  $\psi(y, h, X)$

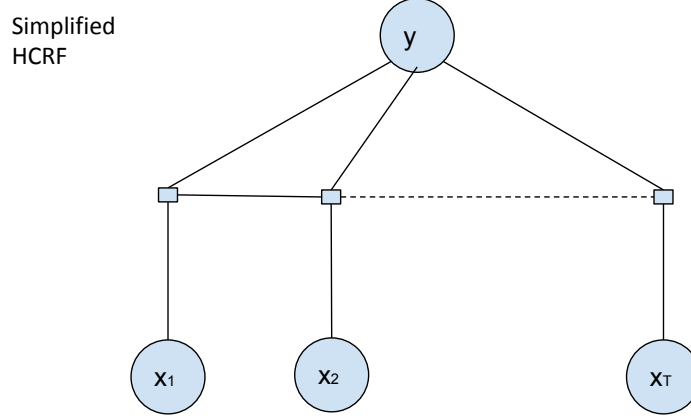


Figure 2.3: Graphical Representation of Simplified Model in Previous Work

$$Z(X) = \sum_{y \in \mathcal{Y}} \exp(\psi(y, X))$$

$$\psi(y, X) = \sum_{i, x_i \in X} \left( \sum_{k=1}^K \lambda_k f_k(x_i, y) \right)$$

Figure 2.3 shows the graphical representation of the simplified model used in current state of art work. First, state transition functions are removed. Second, hidden states are removed from the model. This results in a much simpler model.

It also claims that, for maximum number of feature functions per-frame,  $K$  is non-convex. To achieve the best result,  $K = 3, 4$  is the optimal value state. That is to say, this HCRF model will also have 3,4 parameters. The  $f$  is the feature function we mentioned above, which is calculated using ICA decomposition.

### 2.3.3 Summary and Limitations

Several current state of art works [3, 20, 4] already setup a framework and have pushed the accuracy of Video Event Recognition to usable states. Most of these works follow these two stages. Their differences mainly focus on different policies on choosing features or applying different sequencing models. Latest work [4] has showed accuracy of 73% with the ICA feature extraction and HCRF sequencing model on golf sports videos.

However, none of the previous work has ever targeted on low-quality videos. Low-



quality videos disable some of the technical assumptions that previous works have made before. Low-quality videos are often greyscale and low resolution, leading significant bias color histogram or no color histogram at all. Low-quality videos may not follow ICA pattern due to pixel distortion. Lower frame rate also affects the current sequence model.

## 2.4 Existing Tools in This Thesis

### 2.4.1 Sparse Representation and Sparse Coding

Sparse Representation and Sparse Coding is a model proposed in 1996 [19]. At first, it is invented to model the visual cortex signals recorded from the brains of monkeys. Previous works on feature extraction all work on manually selected features, like color histogram. Unlike them, Sparse Coding operates on raw pixels. For low-quality videos, most of them are greyscale, and that simplifies this technique, since pixel colors are scalar (greyscale), rather than 3-dimensional (RGB).

In this technique, Sparse Representation is a decomposition process, and Sparse Coding is an optimization learning process. The outcome of Sparse Coding process is a set of vectors called *base vectors*. After this optimization learning process, Sparse Representation can decompose the input images into these base vectors with coefficients.

#### Sparse Coding

This is the prerequisite process for Sparse Representation. In this process, the computer learns the feature extractor itself. The input to this process is training frames. Each frame is a greyscale image. We extract all the pixels from the greyscale image and put them in a vector  $a$ . For example, in one of our evaluation, the image is  $36 \times 24$ , then the length of vector  $a$  is 864.

Suppose that we have altogether  $N$  frames in a scene, and each vector has a length of  $M$ . We gather all those vectors into a  $M \times N$  matrix called  $A$ . We would like to decompose matrix  $A$  (which is all of training images) into a product of two matrices,  $B$  with size of  $M \times K$  and  $Z$  with size of  $K \times N$ , where  $K$  is a tunable parameter:

$$A = B \cdot Z + \epsilon \text{ s.t. } \min_{B, \alpha} \left\{ \frac{1}{2} \|\epsilon\|_F^2 + \|Z\|_1 \right\}$$

This decomposition process is a constraint based decomposition process. The target for optimization is to make  $0.5\|\epsilon\|_F^2 + \|Z\|_1$  as minimum as possible. With this constraint, we decompose  $A$  into  $B \cdot Z$ . We call matrix  $B$  the *base matrix*, and the column vectors in  $B$  the *base vectors*. We call matrix  $Z$  the *coefficient matrix*. If we look through image by image:  $A = \{a_1, a_2, \dots, a_N\}$  ( $a_i$  is a vector representing video frame). Then we have

$$a_i = \sum_{k=1}^K z_{k,i} b_k + \epsilon_i$$

This equation is viewed from each frame's perspective. Which is to say,  $a_i$  is linearly represented by a set of vector  $B = \{b_1, b_2, \dots, b_K\}$ . We save the base matrix  $B$  as the result of the learning process, and we gather all coefficients  $\{z_{1,i}, z_{2,i}, \dots, z_{K,i}\}$  into a vector, defined as feature vector with respect to base matrix  $B$ .

The feature learning process actually learns a set of base vectors from the training data. Noticing that the algorithm optimizes the error, and learns the base matrix itself. This is why it is usually referred as self-taught learning.

## Sparse Representation

Sparse Coding is the feature learning process. After the learning process, using the base matrix we learned, we can generate the features. Given any input image, we extract all of its greyscale pixels into a vector, denote as  $x$ . We want to linearly decompose  $x$  with respect to the base matrix.

$$x = \sum_{k=1}^K \alpha_k b_k + \epsilon_x \text{ s.t. } \min\{\|\epsilon_x\|_2 + \|\alpha\|_1\}$$

The constraint for vector decomposition is to minimize  $\|\epsilon_x\|_2 + \|\alpha\|_1$ . Similar to above, we gather the coefficients  $\alpha_k$  into a vector  $\alpha$  as the feature vector. Coefficients inside the feature vector indicates that, for frame  $x$ , the similarity towards certain base vectors. Notice that we are minimizing the  $\|\alpha\|_1$  for the coefficients, which indicates that frames have rather high tendencies towards some base vectors.

# Chapter 3

## Self-Taught Feature Extraction

One of the major challenges to recognize video events on low-quality videos is to find new technical assumptions to extract features. Previous works have pointed out some efficient ways to manually extract features from certain types of high-quality videos. However, these methods can hardly be applied to low-quality videos.

First, feature extraction method by previous work is a time-consuming manual process. In previous work, the feature extraction step is based on domain specific knowledge. To extract features, researchers from previous work must carefully examine all aspects of features in the video frame, including colors, textures, or even SIFT key points. Although this turns out to be an efficient method for high-quality sports videos at the moment, repeating such a process for low-quality videos is a time-consuming process.

Second, low-quality videos present a greater challenge since the total amount of information is lacking. This makes manually finding features even harder. Taking color features as an example, previous work extract features by looking at the color distribution of each frame. Here, however, in low-quality videos, the color might be distorted or even not existing at all.

In table 3.1, we present a contrast on video qualities between our work and previous work. We use the same video, but our quality is much lower. Table 3.1 shows a sample video frame from both two kinds of videos. Previous work’s video was  $352 \times 240$  with 24-bit True Color, while our goal is to recognize video with a resolution of  $36 \times 24$  only and greyscale color.

Therefore, we believe, for low-quality videos, we need a novel method to extract



	
A sample frame of high-quality videos that previous work uses	A sample frame of low-quality videos in our evaluation

Table 3.1: Comparison of Video Qualities between Previous Work and Our Goal

features. In this chapter, we will present our first major contribution, self-taught feature extraction.

### 3.1 The Need for Self-Taught Feature Extraction

Manually finding features to extract from low-quality videos is hard. However, before we decide to solve this problem, we would like to affirm the question if this problem is solvable and does this feature even exist in low-quality videos?

The answer to this question is yes! We manually inspect these low-quality videos and find out that recognizing video events for a human is quite easy. This suggests two things: firstly, even in these low-quality videos, there is still enough features existing. Secondly, human beings seem to learn extracting features without finding the features. Recognition is part of the human vision. Therefore, we ask the following questions:

Since finding features manually is hard, can computers learn the features on its own? How does human's vision system learn features from these low-quality video frames? Can we borrow the ideas from human vision system to make computers teach themselves on how to extract features from these low-quality videos?

Fortunately, neuroscientists have revealed the possible mechanism of human vision system [19, 18], named Sparse Representation/Sparse Coding. In recent years, as computers are getting faster, this mechanism has been adopted by many other systems [9, 25, 26, 27] to solve problems in vision recognition. Our approach towards Video

Event Recognition for low-quality videos can simulate this mechanism, just as the way that humans learn to see. The way that humans learn to see does not require any manual feature findings, and it does not make any technical assumptions like colors, distributions, etc. Therefore, we call this mechanism being self-taught [28]. Thus, for us, we can avoid the trouble of manually finding features from low-quality videos.

## 3.2 Definition of Feature Function

First of all, we need to formalize the first step, feature extraction. Feature extraction is to transform the video frame from pixel level into a higher semantic level. We denote frame  $i$  in a scene as  $x_i$ .  $x_i$  is a vector of all greyscale pixels in the frame  $i$ . For example, if the frame is  $36 \times 24$ , then the length of vector  $x_i$  is 864. We denote  $y$  as a type of video event, and  $y \in \mathcal{Y}$ , where  $\mathcal{Y}$  is a set of all possible video events.

The objective of extracting features from frame  $x_i$  is to find a set of feature functions ( $K$  of them in total)  $f_k(x_i, y)$  ( $k = 1, 2, \dots, K$ ), which represents likelihood of frame  $x_i$  being video event  $y$ .

We would also like to constrain the value of function  $f_k(x_i, y) \in [0, 1]$ . In the later section, we will show that this constraint does not affect the final accuracy, as this feature function will eventually be used to model a conditional probability.

## 3.3 General Process

In Section 2.4.1, we introduces the sparse coding and sparse representation model. Sparse Coding is an optimization learning process. It optimizes a cost function and generates a set of base vectors. Sparse Representation is a decomposition process: given a set of base vectors, it can decompose the input vector into base vectors with coefficients. Inspired from this, we would like to use Sparse Coding as a feature training process and Sparse Representation as a learning process.

This process is explained in figure 3.1. Training scenes, along with their video events, are processed first through the feature training process. The training process generates a set of base vectors as the product. This step is self-taught: the computer teaches itself to find what features to look for. The second step, when a testing frame needs a feature

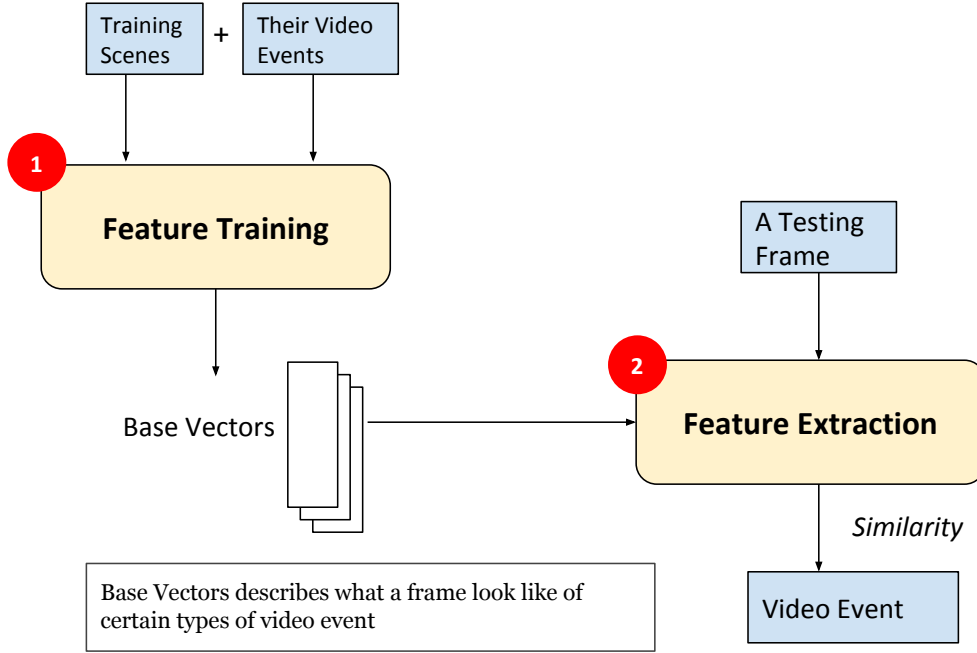


Figure 3.1: General Process of our Feature Extraction Method

function, it goes through a feature extraction process. The feature extraction process needs to read the base vectors generated by the feature learning process. The feature extraction finally outputs a feature function, which describes the similarity between a testing frame and any video events.

### 3.4 Feature Training

In the feature training process, we need to teach a computer how a type of video event looks. By using sparse coding technique, the computer will figure out what to learn from these video frames. Therefore, in this feature training process, we would like to tell the computer how each frame looks for each type of video events. To achieve this, we need labeled-data. In the sequence summary model section, we will need labeled-data for training as well.

Labeled-data in Video Event Recognition, also referred as the training scenes, is part of the testing video that has video events manually marked. The training scenes are usually part of testing videos as well. The purpose is to ensure the training is not biased.

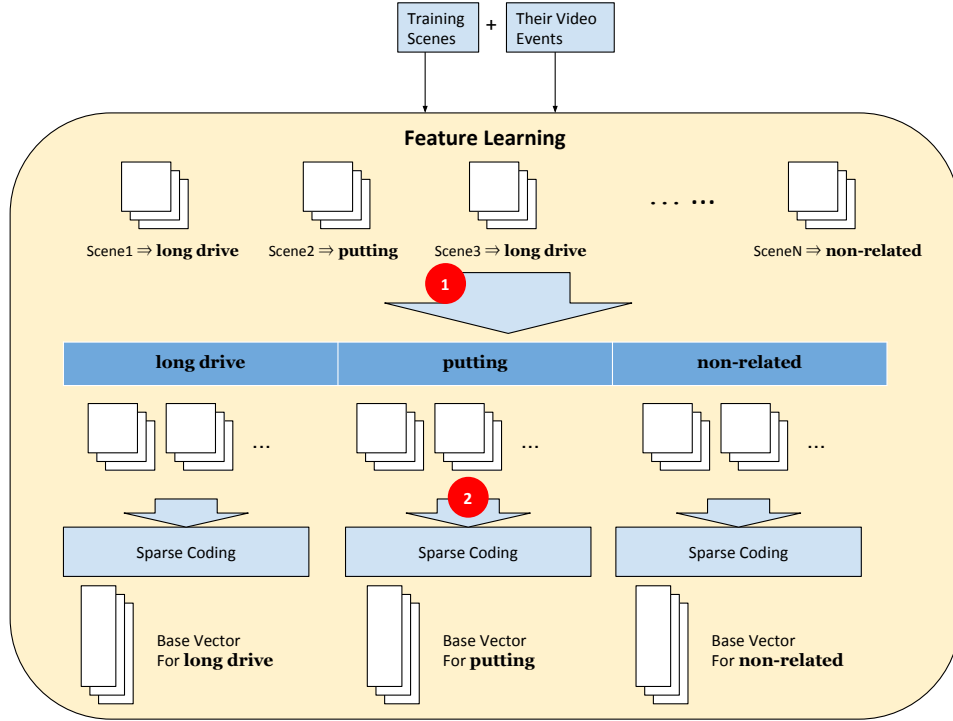


Figure 3.2: Feature Training Process

The labeled-data is a set of scenes that already knows to have video events. Let's denote the training data as the following set

$$TrainingData = \{(X_{t_1}, y_{t_1}), (X_{t_2}, y_{t_2}), (X_{t_3}, y_{t_3}) \dots\}$$

, where for each training scene  $X_{t_i}$ , the known video event of it is  $y_{t_i}$ . Each scene contains many frames, we call these frames as *training frames*, and we also call the training frames inside  $X_{t_i}$  belong to video event  $y_{t_i}$ .

The feature training process is to let the computer know: for each video event  $y$ , how do the frames belong to the video event  $y$  look. Therefore, the first step we would like to do is to classify the training frames by its known video event. The step 1 in figure 3.2, shows an example. We list a table that each column represents a video event and the scenes that belong to it. We name all the training frames in the following convention: for video event  $y$ , frame  $j$ ,  $x_j^y$  is the  $j$ th frame among all frames that belong to video event

$y$ .

The first step is the only step that needs for manual intervention: researchers have to at least tell the computer how the frames look for each type of video events. Then the computer could figure out what kind of features to look for to accomplish the recognition.

After this first step, the next step is the self-taught learning step. In this step, we apply the Sparse Coding technique.

Recalling the sparse coding feature training process. The sparse coding process will learn a base matrix itself from a set of training frames with the constraint that coefficients of them must be  $l_1$  minimized. Supposing for each video event  $y_j \in \mathcal{Y}$ , we gather all the video frames that belong to  $y_j$  as a matrix  $A^{y_j}$ , which is:

$$A^{y_j} = \{x_1^{y_j}, x_2^{y_j}, x_3^{y_j}, \dots\}$$

As defined above, each video frame  $x_i^{y_j}$  is a vector and length of which is the size of the video frame image. Therefore,  $A^{y_j}$  is a matrix and the height of which is the size of the video frames, the width of which is the number of training frames belonging to the video event  $y_j$ . Also, as defined above, for each video event  $y_j \in \mathcal{Y}$ , we will have such matrix. So there are  $|\mathcal{Y}|$  such matrices in total.

Next, for each of these matrices, we run a Sparse Coding process. Taking  $y_j \in \mathcal{Y}$  again as an example, according to Sparse Coding, we have

$$A^{y_j} = B^{y_j} \cdot Z^{y_j} + \epsilon^{y_j} \text{ s.t. } \min_{B, \alpha} \left\{ \frac{1}{2} \|\epsilon^{y_j}\|_F^2 + \|Z^{y_j}\|_1 \right\}$$

Similar to the definition in the Sparse Coding section, here,  $B^{y_j}$  is called the base matrix and the height of which is the same as  $A^{y_j}$  – the size of video frames. The width of which is a tunable parameter, which we call as the *degree of Sparse Coding*,  $K$ . As we have shown in the Sparse Coding section, another interpretation of the above equation would be

$$x_i^{y_j} = \sum_{k=1}^K z_{k,i}^{y_j} b_k^{y_j} + \epsilon_i^{y_j}$$

As you see, each frame that belongs to video event  $y_j$  is decomposed into  $K$  – the *degree of Sparse Coding* – base vectors with coefficients  $z$ . Sparse Coding generates the *base matrix*  $B^{y_j}$  or a set of *base vectors*  $b_k^{y_j}$  for each video event  $y_j$ . So, in total, there



are  $|\mathcal{Y}| \cdot K$  base vectors.

The physical meaning for these base matrices is simple. For each of these base matrices, it is selected by Sparse Coding, and it is a summary of all the frames that belong to video event  $y_j$  from the training scenes.

The step 2 in figure 3.2 shows this process. We take the frames from each column in the table that constructed by step 1, and then run Sparse Coding on each column. Each column represents a video event, and therefore, the base vectors generated from that column represent the video events.

The base vectors (or the base matrices) are the product of this feature training step. The feature extraction is going to use these base vectors to extract features. We will introduce the details in the next section.

## 3.5 Feature Extraction

The ultimate goal for feature extraction is to come up with a feature function. In this section, we will show how to generate such feature function.

In the previous section, the data that feature learning processed is the training data. The training data is labeled-data, and it has both training scenes and the known video event associated with it. In this section, we need to extract features from arbitrary video frames. Hence, we will work on the *testing video*. The testing video contains several scenes as well. We call these scenes the *testing scenes*, and the frames inside them as *testing frames*. Recalling the feature function definition is to find function  $f(x_i, y)$  to measure how likely that frame  $x_i$  in a scene belongs to video event  $y$ . In this section, we construct a set of feature functions  $f_k(x_i, y)$  to measure how likely is  $x_i$  belong to video event  $y$ , with any arbitrary testing frame  $x_i$ . This section constructs  $K$  feature functions in total, where  $K$  is a tunable parameter defined in the previous section, known as the *degree of Sparse Coding*.

We notice that the base vectors generated from feature learning process are the summary made by the computer from frames of a certain type of video events. Therefore, we would like to make our feature functions to measure how similar the testing frame is compared with the base vectors we learned. If we can measure this similarity as a real number between  $[0, 1)$ , then we can use this measurement as the feature functions.

Recalling that these base vectors are generated by Sparse Coding, which means any training frames can be sparsely decomposed into these base vectors. If the testing frames are similar to the training frames, then the testing frames can also be sparsely decomposed into these base vectors.

In the section 2.4.1, we give a brief introduction on sparse representation. Here we will show how we leverage the coefficients from Sparse Representation to construct our feature functions.

In Sparse Representation, we will decompose all the input vectors into base vectors and minimize the  $l_1$  norm of the coefficients. Therefore, the first thing we need to do is to select the base vectors for Sparse Representation. In the feature learning process, it generates  $|\mathcal{Y}| \cdot K$  base vectors. For each video event, it generates  $K$  vectors, and here we need to construct  $K$  feature functions. To construct a feature function  $f$ , we take one base vector from each video event. Therefore, for any testing frame  $x$ , we have:

$$x = \sum_{y_j \in \mathcal{Y}} \alpha_{y_j, k} b_k^{y_j} + \epsilon_x \text{ s.t. } \min\{\|\epsilon_x\|_2 + \|\alpha\|_1\}$$

, where  $k = 1, 2, 3, \dots, K$ .  $b_k^{y_j}$  is  $k$ th base vector that is summarized from video event  $y_j$ . That is to say: input testing frame  $x$  is decomposed into  $|\mathcal{Y}|$  parts with base vectors from **all possible video events**. Step 1 in figure 3.3 shows this process. As it shows the decomposition on input testing frame  $x$  is done with 3 base vectors as an example. Each base comes from all different video events. In total, we can have  $K$  of these decompositions.

In this way, we collect coefficients from this decomposition,  $\alpha_k = [\alpha_{1,k}, \alpha_{2,k}, \dots, \alpha_{|\mathcal{Y}|,k}]$ . We refer  $\alpha_k$  as a *feature vector*. Later in this section, we will generate feature functions from the feature vectors.

As the Sparse Representation is a linear decomposition process, the coefficient of that base vector represents the similarity between the testing frame and the base vector. The larger the coefficient for that base vector is, the similar the video frame to that base vector is. Intuitively, we can just use the coefficients as the feature functions, such as:

$$f_k(x_i, y) = \alpha_{y,k}$$

However, this has a major flaw, if the base vectors that generated from sparse coding

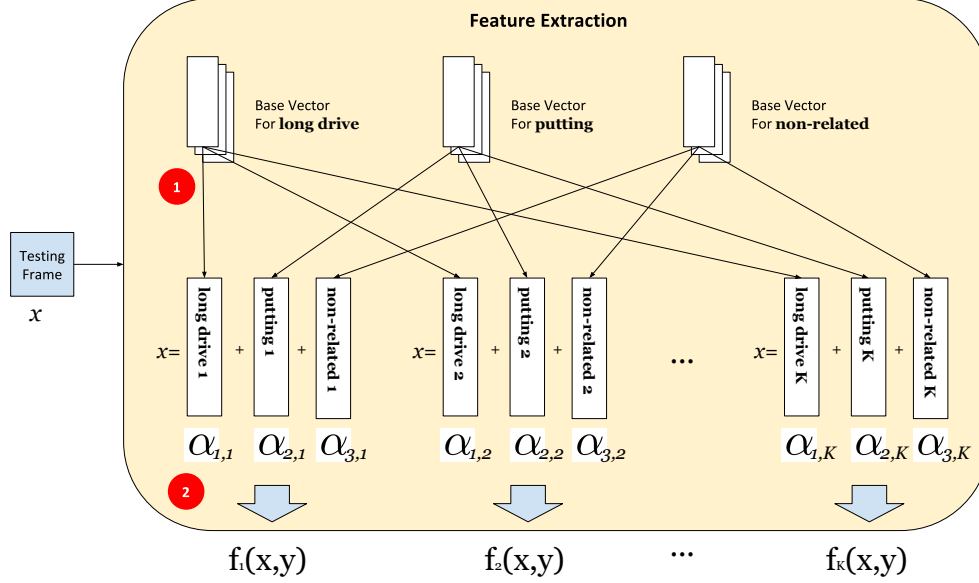


Figure 3.3: Feature Extraction Process

are not normalized. There exists such a case that  $x_i$  and  $y_0$  are similar, but since all elements in the  $b_k^{y_0}$  are very large,  $\alpha_{y,k}$  turning out to be very small. This corner case suggests us that the feature function should be constructed from metrics that's normalized.

Noticing that this is a linear decomposition, and also, the  $l_1$  norm of  $\alpha_k$  is minimized, which suggests that coefficients are highly discriminative between each other: most of them are very small and close to zero, while few of them are non-zero. Those non-zero coefficients indicates that the input testing frame is similar with these base vector. This phenomenal inspires us to use the error as a measurement of similarity.

Supposing we would like to know the similarity between the input testing frame and the video event  $y$ . We only need to pick out the coefficient for base vector  $b_k^y - \alpha_{k,y}$  and treat rests of base vectors as errors. Consider the following function:

$$f_k(x_i, y) = 1 - \frac{\|x_i - \alpha_{y,k} b_k^y\|_2}{\|x_i\|_2}$$

The  $\frac{\|x_i - \alpha_{y,k} b_k^y\|_2}{\|x_i\|_2}$  part in the feature function is an error function. The smaller the error is, the closer  $x_i$  is to video event  $y$ . Therefore, we use 1 to subtract that. Now the

error function is between  $[0, 1)$ . The larger it is, the more likely that testing frame  $x_i$  belongs to video event  $y$ . We complete describing this process as figure 3.3. There are three steps involved. As a testing frame  $x$  inputs the feature extraction phase, firstly, we rearrange the base vectors. Secondly, by doing Sparse Representation, we decompose  $x$ . Lastly, according to the coefficients from sparse representation, we construct our feature function  $f_k$ . In total, there are  $K$  of these feature functions for each frame.

## 3.6 Chapter Summary

In this chapter, we describe one of our major contributions of this thesis. To overcome the difficulties of finding features from low-quality videos, we solve this problem in a different way. We use self-taught learning tools to solve this problem: let the computer itself find out what it needs to learn. We first present the feature learning process. Through this process, we give computer “a direction to learn features”. We use Sparse Coding as a tool and construct our own strategy for learning features. The intermediate results, which are generated from this process, are referred as the base vectors. Base vectors are the summary of frames in a type of video event.

Next, we show that, by using the base vectors, which are generated from training scenes, we are able to construct  $K$  feature functions for later use. We call the coefficients, which are constructed from Sparse Representation, as feature vectors. We also make sure the feature functions are generated from normalized metrics to avoid corner cases.

Figure 3.3 shows a complete picture of the whole feature extraction process. The whole process is novel and effective, and it makes no pre-assumptions on the video qualities.

## Chapter 4

# Hidden States Sequence Summary Model

Previous chapter introduced feature functions that can transform the video frame from pixel space into higher level semantics. The high level semantics in Video Event Recognition system is the similarity between a testing frame and a video event. The more a testing frame looks like a video event, the larger the feature function value is.

However, this only covers the model of each frame. Recall the goal of Video Event Recognition is to recognize the video event of a scene, where a scene has multiple frames. Therefore, we need a summary model that can summarize these frames. As in previous sections, we are able to transform frames from pixel space into feature space. The next step is to summarize these features from these frames in a scene, and recognize the appropriate video event for this scene.

In section 2.2, we mention that previous work has already proposed a general framework for Video Event Recognition. Last chapter covers the first part of this general framework – feature extraction step. This chapter is going to cover the second part – the sequence summary model.

Previous works [3, 4] have proposed several sequence summary models. We mention some of them in section 2.3. The current state of art work [4] uses *HCRF* (*Hidden states Conditional Random Field*) model as sequence summary model and shows that this model has very good accuracy on Video Event Recognition. However, a major challenge showed up. Feature extraction for low-quality videos cannot produce very representative

features. Although in the last section, we use our novel feature extraction technique to improve the feature extraction process, this is still causing a trouble for accuracy. This trouble, in turns, affects the sequence summary model, demanding more accuracy, and adaptive sequence summary model compared to previous work.

Based on the sequence summary model which is used in the current state of art work [4], we improve its sequence summary model. By adding semantic hidden states to the model, we are able to build a solid, adaptive and accurate sequence summary model for Video Event Recognition.

## 4.1 Definition of Sequence Summary Model

First, we would like to formalize the problem statement of sequence summary model in this chapter. In Video Event Recognition, as the second step, sequence summary model collects all the features for all the frames inside the scene as a feature sequence and summarizes this feature sequence into a video event.

Similar to the definition in the feature extraction process, we need to separate between *training scenes* and *testing scenes*. Training scenes are *labeled-data*, for they contain *training frames*, and each training scene is known as belonging to a certain video event. Similar to denotation in the previous chapter, we denote the scene as  $X$ , and the  $i$  th testing frame in a testing scene as  $x_i$ . Given one video event  $y$ , for each frame  $x_i$ , we could generate  $K$  feature functions in total:  $f_1(x_i, y), f_2(x_i, y), \dots, f_K(x_i, y)$ . By using these sequences of features, we would like to measure a conditional probability:  $P(y|X)$ , which is given  $X$ , for any video event  $y$ , finding out the conditional probability. The higher this probability is, the more likely this testing scene  $X$  is recognized as video event  $y$ .

So the key question is how to come up with a model to model the condition probability  $P(y|X)$  from the sequence of these features. Previous state of art work uses a statistical learning model called *HCRF* (*Hidden States Conditional Random Field*). We cover some of the background about HCRF in section 2.3.2. It is a regression model. First it gives a generic distribution of this conditional probability  $P(y|X)$ . The generic distribution contains a lot of parameters. Then, through its training process, it optimizes an error function to estimate these parameters.

Training scenes are used for training the HCRF model, to generate a concise conditional probability  $P(y|X)$ . Testing scenes are testing data, and they contain testing frames. The goal is to recognize their video events, through calculating the conditional probability  $P(y|X)$  we trained.

## 4.2 Introducing Hidden States

The original HCRF model is proposed with hidden states, however, in the model of the current state of art work, it trims away the hidden states. Recalling that in section 2.3.2, we mention the model that is used in the current state of art work.

$$P(y|X) = \frac{1}{Z(X)} \exp(\psi(y, X))$$

, where  $Z(X)$  and  $\psi(y, X)$

$$Z(X) = \sum_{y \in \mathcal{Y}} \exp(\psi(y, X))$$

$$\psi(y, X) = \sum_{i, x_i \in X} \left( \sum_{k=1}^K \lambda_k f_k(x_i, y) \right)$$

In the model used in current state of art work,  $f_k(x_i, y)$  are the feature functions. Current state of art work uses ICA (*Independent Component Analysis*) to construct feature functions.  $\lambda_k$  are the regression parameters to HCRF model. They are the weights among feature functions. As described in section 2.3.2, these parameters will be estimated during training process in HCRF model. There are two intermediate functions  $Z$  and  $\psi$ .  $Z$  is a normalization factor, which make sure HCRF models a probability.  $\psi$  is a sum of all feature functions with their weights.

Unlike the original HCRF model, the current state of art model used in Video Event Recognition contains no hidden states. In other words, the model in previous work is a weaker form. It is likely due to two reasons. Firstly, to add hidden states into this model, we must have a clear definition of hidden states. Previous work seems to ignore the effort on this part. Secondly, for the scenario in previous works, since the videos are high-quality, the feature function can usually present very precise results. This in turn

shows that by adding hidden states to improve the sequence summary model might not be necessary.

In fact, in current state of art work, it suggesting  $K = 3, 4$  is to be able to achieve the optimal accuracy, while in a lot of HCRF applications [29, 24], usually end up with  $K$  being around  $10 \sim 20$ . This suggests that in the current state of art work, sequence summary model is not a bottleneck.

However, with the drop of the video quality in our applications, the need for a more accurate sequence summary model became more and more important. To solve this problem, we would like to realize the full potential of HCRF model, by adding hidden states into it. Firstly, like the approach of previous work, we would like to use our feature function as the state function in HCRF model. Also, similar to previous work, we currently do not have any state transition functions. As a result, we will just use feature functions as the state functions. Secondly, we would like to add hidden states to the existing model. Ideally, we would like to have:

$$P(y|X) = \frac{1}{Z(X)} \sum_{h \in \mathcal{H}} \exp(\psi(y, h, X))$$

, where  $Z(X)$  and  $\psi(y, h, X)$

$$Z(X) = \sum_{y \in \mathcal{Y}} \sum_{h \in \mathcal{H}} \exp(\psi(y, h, X))$$

$$\psi(y, h, X) = \sum_{i, x_i \in X} \sum_{k=1}^K \lambda_k f_k(x_i, h, y)$$

Compared to model from previous work, two parts of the model change. The first part is that we introduce hidden states to HCRF. Aside from the set of all possible video events  $\mathcal{Y}$ , similar to this, we introduce a new set of possible hidden states set  $\mathcal{H}$ . The second part is the feature functions now changing. It now takes in a new parameter:  $h$ .

This change improves the model accuracy in general. This model is more flexible as well. First, it allows feature functions being able to be less accurate, for the  $\sum_{h \in \mathcal{H}} \exp(\psi(y, h, X))$  part in  $P(y|X)$  will smooth the non-representative features. Second, it makes the regression model more powerful, as the regression contains more parameters.



In the next two sections, we will first introduce how to construct these hidden states, and explain their physical meanings. Then, we will show the modifications we need to improve the feature functions, and how to add a new parameter  $h$  to the feature functions.

## 4.3 Constructing Hidden States

The ideal model we have in mind is a HCRF model with hidden states. All possible hidden states is collected in a set  $\mathcal{H}$ . However, to construct such model, we need to setup the proper hidden states. In order to do this, first, we need to know, from a structural point of view, what are the hidden states. Once we understanding the structure, we are able to create meaningful hidden states.

### 4.3.1 Mathematical Property of Hidden States

First recalling our model for  $P(y|X)$  is the following.

$$P(y|X) = \frac{1}{Z(X)} \sum_{h \in \mathcal{H}} \exp(\psi(y, h, X))$$

We notice that  $P(y|X)$  is a sum over all possible  $h \in \mathcal{H}$ . Therefore if we consider  $P(y, h|X)$  as

$$P(y, h|X) = \frac{1}{Z(X)} \exp(\psi(y, h, X))$$

Then  $P(y|X)$  becomes clearer, because

$$P(y|X) = \sum_{h \in \mathcal{H}} P(y, h|X)$$

If we notice the definition of  $Z$  and  $\psi$ :

$$Z(X) = \sum_{y \in \mathcal{Y}} \sum_{h \in \mathcal{H}} \exp(\psi(y, h, X))$$

$$\psi(y, h, X) = \sum_{i, x_i \in X} \sum_{k=1}^K \lambda_k f_k(x_i, h, y)$$

We could find that variable  $y$  and  $h$  always turn up together in pair. Supposing we let  $\eta = (h, y) \in \mathcal{H} \times \mathcal{Y}$ , we can denote our model with hidden states as the following form.

$$P(y|X) = \sum_{h \in \mathcal{H}} P(\eta|X)$$

$$P(\eta|X) = \frac{1}{Z(X)} \sum_{h \in \mathcal{H}} \exp(\psi(\eta, X))$$

, where  $Z(X)$  and  $\psi(y, h, X)$

$$Z(X) = \sum_{\eta \in \mathcal{H} \times \mathcal{Y}} \exp(\psi(\eta, X))$$

$$\psi(\eta, X) = \sum_{i, x_i \in X} \sum_{k=1}^K \lambda_k f_k(x_i, \eta)$$

Comparing the  $P(\eta|X)$  with the sequence summary model in current state of art work, we find that this is exactly the same model. This implies the following facts of current state of art model: to recognize video event among  $|\mathcal{Y}|$  video events, adding  $|\mathcal{H}|$  hidden states to this model is equivalent to applying this model to recognize among  $|\mathcal{H} \times \mathcal{Y}|$  video events.

This mathematical fact of hidden states implies that the hidden states we proposed can be transformed into existing model with a different parameter. Existing model can be reused as a component in our hidden states model.

### 4.3.2 Physical Meanings of Hidden States

This mathematical property of hidden states suggests that hidden states in themselves have no difference to a video event. Adding  $|\mathcal{H}|$  hidden states only simply extends the number of all possible video events from  $|\mathcal{H} \times \mathcal{Y}|$ . Therefore, we can view hidden states as video sub-events: The video events inside a video event. The existence of hidden states indicates that all video events, are a process of change and contain several stages. These stages are called sub-events, and we define the sub-events relatively to the major video event as hidden states  $h \in \mathcal{H}$ . The idea is the unique point that hidden states can

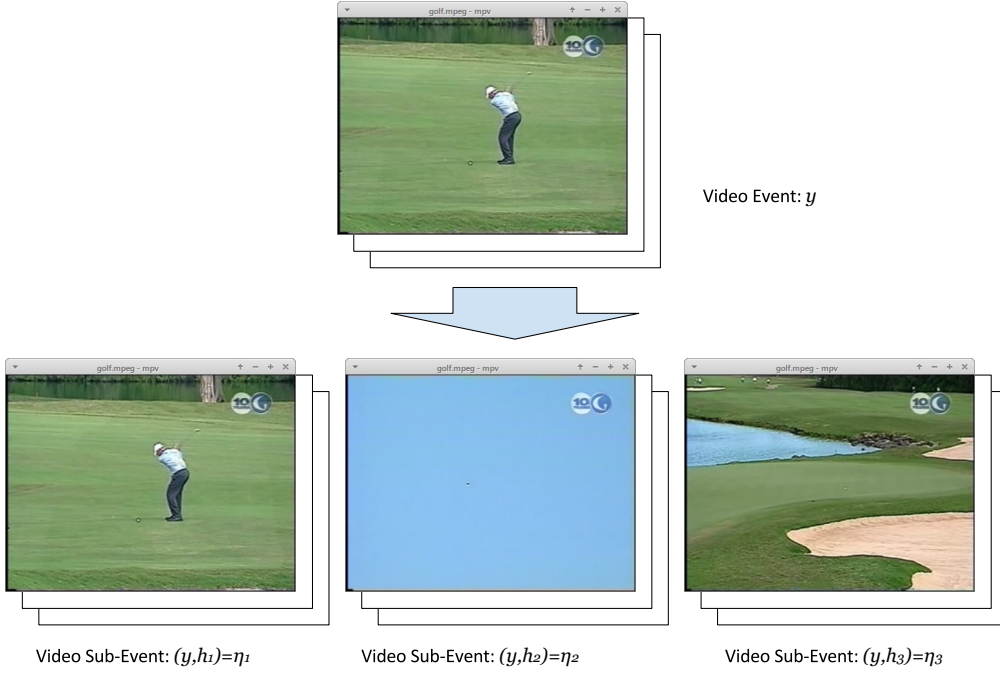


Figure 4.1: Example of Video Sub-Event

be leveraged to improve accuracy.

Figure 4.1 gives an example from real videos. In this figure we show an example that a video event consist of several sub-events ( $|\mathcal{H}| = 3$ ). This video event is a long drive video event. By using the mechanism we mention before, we are able to divide the video event into different video sub-events: 1. a player hit the ball; 2. the ball flying in the sky; 3.the ball hit the ground. The hidden states combined with the video events form a video sub-event, which belongs to set  $\mathcal{H} \times \mathcal{Y}$ .

## 4.4 Feature Extraction with Hidden States

In previous sections, we showed what the model with hidden states could be used for Video Event Recognition. In the original HCRF model [24], the hidden states are implicit. In both training and testing of HCRF model, the hidden states are summed away. This means that there is no need to modify the training or testing algorithm on the sequence summary model. In this section, we discuss about, by adding hidden states, how it will

affect our feature extraction step.

As we summarize in section 4.2, there are two parts of changes by introducing hidden states. The first is the sequence summary model itself. Second, it adds a new parameter to the feature function  $f$ . Now the feature functions have to be aware of hidden states.

In this section we modify our feature extraction step to make it aware of hidden states. This change involves in both feature training and feature function.

#### 4.4.1 Feature Training with Hidden States

Recalling in the feature training step. We collect all the training scenes and gather them as the labeled-data for training. The labeled-data, or the training scenes, can be considered as the following format:

$$TrainingData = \{(X_{t_1}, y_{t_1}), (X_{t_2}, y_{t_2}), (X_{t_3}, y_{t_3}) \dots\}$$

Considering the frames inside each training scene, frames from one training scene belonging to the same video event. The training data can be written as the following. You can view the training data as a set of tuples containing testing frame and its video event.

$$TrainingData = \{(x_{t_1}, y_{t_1}), (x_{t_2}, y_{t_2}), (x_{t_3}, y_{t_3}) \dots\}$$

We gather all the video frames that belong to video event  $y_j$  as a matrix  $A^{y_j}$ , which is:

$$A^{y_j} = \{x_1^{y_j}, x_2^{y_j}, x_3^{y_j}, \dots\}$$

Then apply Sparse Coding to learn a set of bases.

$$A^{y_j} = B^{y_j} \cdot Z^{y_j} + \epsilon^{y_j} \text{ s.t. } \min_{B, \alpha} \left\{ \frac{1}{2} \|\epsilon^{y_j}\|_F^2 + \|Z^{y_j}\|_1 \right\}$$

, where  $x_i^{y_j}$  means a training frame from any scene belonging to video event  $y_j$ .

This process currently are not aware of hidden states. That means that the base matrix or base vectors it generates are all belonging to major video event  $y_j$  without being aware of the video sub-events, or the hidden states. To solve this problem, we

---

**Algorithm 4.1** Video Sub-Event Selection During Feature Training

---

```

SubEvent(training_scene, major_video_event, number_hidden_states) {
    Frame last_frame = null
    PriorityQueue Q
    foreach Frame x in training_scene:
        double difference = L2Norm(PixelVector(x) - PixelVector(last_frame)
        )
        Q.add(tuple(difference, x))
        last_frame = x
    for i = 0 to number_hidden_states:
        difference, x = Q.pop()
        Print("sub-event detected at frame " x)
}

```

---

would like to have the training data include the information of hidden states.

$$TrainingData = \{(x_{t_1}, \eta_{t_1}), (x_{t_2}, \eta_{t_2}), (x_{t_3}, \eta_{t_3})\} = \{(x_{t_1}, y_{t_1}, h_{t_1}), (x_{t_2}, y_{t_2}, h_{t_2}), (x_{t_3}, y_{t_3}, h_{t_3}) \dots\}$$

, where  $\eta_{t_i} = (h_{t_i}, y_{t_i})$ ,  $y_{t_1}$  is the major video event for each frame  $x_{t_1}$ , and  $h_{t_1}$  is the minor video sub-event for frame  $x_{t_1}$ . We already have  $y_{t_1}$  from the training scenes, therefore, the frames. Next, we need to know the minor video sub-event for each frame.

Inspired from video boundary detection, we realize that we can cut the training scenes into sub-scenes. Since we already know the major video event for training scenes. If we apply a video shot boundary detection algorithm to it, the training scenes will break into several video sub-events.

Similar to the way that previous work applied in video boundary detection. We calculate the frame difference between each training frames in the training scene. Rather than cutting the training scenes into sub-scenes by a fixed threshold, we are able to sort these frame differences in ascending order. Next, we pick the top  $|\mathcal{H}|$  frame differences, and cut right at these timeline positions. As the pseudo-code in algorithm 4.1 shows.

In this way, each training frame with known video events are cut into sub-scenes. These sub-scenes could be used as training data for video sub-event. Therefore, we assign the hidden states  $h$  to each frames in the training video. In this way, we construct the training data in the form of:

$$TrainingData = \{(x_{t_1}, y_{t_1}, h_{t_1}), (x_{t_2}, y_{t_2}, h_{t_2}), (x_{t_3}, y_{t_3}, h_{t_3}) \dots\} = \{(x_{t_1}, \eta_{t_1}), (x_{t_2}, \eta_{t_2}), (x_{t_3}, \eta_{t_3})\}$$

Once we constructed this form, we are able to use the feature training method explained in section 3.4, by simply replacing variable  $y$  with variable  $\eta$ . First, we collect all the frames that belong to each  $\eta = (h, y) \in \mathcal{H} \times \mathcal{Y}$ , which means frames that belong to major video event  $y$  and minor video sub-event  $h$ . Then we run Sparse Coding to learn a set of base vectors. This will generate  $|\mathcal{H}| \cdot |\mathcal{Y}| \cdot K$  base vectors, where  $K$  is the degree of Sparse Coding and the length of each base vector is the size of video frame, since  $\eta \in \mathcal{H} \times \mathcal{Y}$ .

#### 4.4.2 Feature Function with Hidden States

In this section, we will show the detail feature functions with hidden states involved. Since we transform the model with the hidden states into an existing model with no hidden states with video events  $\eta \in \mathcal{H} \times \mathcal{Y}$ , the targeting feature functions could also transform from  $f_k(x_i, h, y)$  into feature functions  $f_k(x_i, \eta)$ , which are exactly the same form as the model in the previous work.

In the previous feature extraction process in section 2.3.1, we use the bases generated from feature learning process to sparsely decompose testing frames. By using the coefficients from the decomposition, we are able to construct the feature functions. Now with the hidden states, the number of base vectors grows from  $|\mathcal{Y}| \cdot K$  to  $|\mathcal{H}| \cdot |\mathcal{Y}| \cdot K$ , we can still run the Sparse Representation decomposition first. For any testing frame  $x$

$$\begin{aligned} x &= \sum_{\eta_j \in \mathcal{H} \times \mathcal{Y}} \alpha_{\eta_j, k} b_k^{\eta_j} + \epsilon_x \\ &= \sum_{h_j \in \mathcal{H}} \sum_{y_j \in \mathcal{Y}} \alpha_{h_j, y_j, k} b_k^{h_j y_j} + \epsilon_x \text{ s.t. } \min\{\|\epsilon_x\|_2 + \|\alpha\|_1\} \end{aligned}$$

Compared with the feature extraction in section 2.3.1, we only change the denotation of vector base  $b$  since there are  $|\mathcal{H}| \cdot |\mathcal{Y}| \cdot K$  of them now. Base vector  $b_k^{h_j y_j}$  means that it is the  $k$ th base vector that generated by Sparse Coding from frames which have major video event  $y_j$  and minor video sub-event  $h_j$ . The coefficients generated from this Sparse

Representation process are denoted as  $\alpha_{h_j, y_j, k}$ . These coefficients are gathered in a vector called  $\alpha$ .

Similarly, we construct the feature function as the following:

$$\begin{aligned} f_k(x_i, h, y) &= f(x_i, \eta) \\ &= 1 - \frac{\|x_i - \alpha_{\eta, k} b_k^\eta\|_2}{\|x_i\|_2} \\ &= 1 - \frac{\|x_i - \alpha_{h, y, k} b_k^{h, y}\|_2}{\|x_i\|_2} \end{aligned}$$

In this way, we construct the feature functions  $f_k(x, h, y)$  that can be used in our hidden states model.

## 4.5 Chapter Summary

In this chapter, we first introduce the importance of a hidden state sequence summary model for Video Event Recognition. The need for this model is initiated from the fact that the quality of these videos are dropping significantly. We first show that, by adding a hidden states  $h$ , we can make the model more flexible. Then, to find the physical meaning of the hidden states, we show the mathematical properties of such hidden states. Motivated by these properties, we construct the hidden states as video sub-events.

After introducing video sub-events, we realize that they also introduce new parameters for feature extraction. We then show that, by using the mathematical properties we mentioned before, the only thing is to make simple adjustment to the feature extraction process. This adjustment only introduces a boundary cutting on the training data to construct video sub-events, and this adjustment will result in a larger number of base vectors. This shows that the mathematical properties we find is both simple and useful.

In summary, we introduce the hidden states into the current state of an art model as a major contribution of this thesis. Combined with the feature extraction mentioned in chapter 3, we improve both the model accuracy and flexibility. These improvements together make Video Event Recognition on low-quality videos possible.





# Chapter 5

## Experimental Results

### 5.1 Implementation

We implement our technique in C++. We use `armadillo` library[30] for matrix computations, and `mlpack` library[31] for Sparse Representation and Sparse Coding. `mlpack` uses an algorithm that Lee proposed[32] in 2007 for Sparse Representation and Sparse Coding. We use `CRFSuite` library to implement HCRF[24]. `CRFSuites`[33] provides basic CRF[29] functionality and uses L-BFGS[34] algorithm to optimize the regression parameters.

Videos are manually pre-processed into scenes. In our implementation, we store the boundary between scenes as a file. Then, we extract frames from the video into a set of JPEG image files.

To take advantage of multiple CPUs, the internal implementation for feature extraction, feature function calculation are highly parallelized using OpenMP. Our implementation has two modes: the training mode and the testing mode. The training mode includes both feature training and sequence summary model training. First, the training process loads frames from the training scenes with their video events, runs the feature learning process, and generates base vectors to a file. Second, using these base vectors, we run feature extraction on the training frames to construct and train the sequence summary model.

The testing mode of our program reads all the frames in the testing scene. In each testing scene, we read the base vectors and use these base vectors to extract features.

Using these features, we apply the sequence summary model to recognize video events.

As the performance is not a major aspect that this thesis targets, we only perform a very quick measurement. Our implementation is fast enough, and memory allocation becomes the bottleneck. It takes around 8 hours to train and test a 1-hour golf video in total. After we solve the memory allocation bottleneck by using `tcalloc` library from Google, then the time to train and test the same video drops to 4-6 hours.

## 5.2 Golf Video Benchmark

To compare our work with the current state of art work. We reuse the same video as previous work used. We further process the video to lower the quality of the video. Therefore, the semantic content of the video is the same. However, our algorithm processes the lower quality video.

### 5.2.1 Video Data Description

The video is the recording of Turtle Bay Golf Championship in 2001. The total length is 58 minutes. The original video is a rather high-quality video. It's a TV broadcast video. The resolution is  $360 \times 240$ , and original fps is 30 frames per second, but for experimental purpose, both previous research and our work make some adjustments on this video. Details of the actual video used are described in table 5.1.

In table 5.1, we also include sample frames and their sizes. Our goal, processing low-quality videos, can result in size reduction on each frame for around 112 times. Combining with the saving on frame rate, we could reduce the total video size to 224 times. Imagining that Video Event Recognition is deployed in a cloud or surveillance application, this level of video quality could save 224 times of transmission bandwidth.

### 5.2.2 Overall Results

#### Accuracy

The golf video consists of 3 types of events, *putting*, *long-drive*, and *non-related*. The classification between these events depends on the players' action inside the video. There



Specification	Video Previous Work	Video in our evaluation
Frame Rate	24	12
Color	24 Bit Color	Grey Scale
Resolution	352×240	36×24
Sample Frame		
Size of Sample Frame	124 KB	1.1 KB

Table 5.1: Video Quality Degradation

are 207 events in total. We use 164 of them for training, and 43 of them for testing. Testing scenes do not include the training scenes.

	GMHMM[35]	ICAMHMM[3]	ICAMHCRF[4]	This Work
Video Quality	High	High	High	Low
Accuracy	56.93%	70.79%	73.28%	72.09%

Table 5.2: Accuracy Comparison

As table 5.2 shows, our solution can produce similar accuracy compared to previous work, but our work is tested on low-quality version of this video, which all previous works fail to work on <sup>1</sup>. The low-quality version of this video is only 2.5MB. It is 200 times smaller than the original high-quality version.<sup>2</sup>

In this experiment, we set the degree of Sparse Coding  $K$  to 4 and the number of hidden states  $|\mathcal{H}|$  to 3. We find  $K$  to 4 to be an experience number. Same as previous work, this parameter needs to be tuned on a case by case basis. In the later section, we will show that  $|\mathcal{H}| = 3$  is optimal for this video, as well as suggesting that for low-quality videos, we need hidden states, i.e. it is better to have  $|\mathcal{H}| > 1$ .

<sup>1</sup>At least, the greyscale video will lead all previous work to generate constant features for all different frames.

<sup>2</sup>The original golf video is already compressed in MPEG format.

	long drive	putting	non-related
long drive	<b>2</b>	1	1
putting	0	<b>28</b>	5
non-related	5	0	<b>1</b>

Table 5.3: Confusion Matrix

### 5.2.3 Confusion Matrix

Behind the accuracy rate, we perform a more detailed analysis. Here, we construct a *confusion matrix*. A *confusion matrix* is a table like table 5.3. Columns and rows are video events in the video. For a testing scene, the column means the correct video event, while the row means the answer our algorithm gives. The number in the cell means how many times of such testing scenes occur. Taking table 5.3 as an example: in cell 1,1, its value is 2, meaning there are two testing scenes, that supposed to be long drive, and our program gives the correct answer.

Table 5.3 shows the confusion matrix we performed using this video. We find this confusion matrix looks highly similar to previous work. However, there is one difference. It is that non-related video event is easy to confuse with the others. We believe that this phenomenon is due to the low resolution and colorless properties of low-quality videos: its data pattern tends to confuse naturally.

### 5.2.4 Hidden States Results

In this section, we measure our improvement to the sequence summary model. As we mentioned above, the number of hidden states  $|\mathcal{H}|$  needs to be tuned on a case by case basis. For the video in this experiment, we show that  $|\mathcal{H}| = 3$  is a rational choice of this parameter. Adding hidden states will boost accuracy compared to previous work’s model, which only had  $|\mathcal{H}| = 1$ .

In table 5.4, we show, for this video,  $|\mathcal{H}| = 3$  can reach the highest accuracy. We find that adding hidden states can easily out-perform the sequence summary model without hidden states, which is the first row in table 5.4.

Size of Hidden States	Accuracy
$ \mathcal{H}  = 1$	62.8%
$ \mathcal{H}  = 2$	67.4%
$ \mathcal{H}  = 3$	72.1%
$ \mathcal{H}  = 4$	69.7%

Table 5.4: Accuracy with respect to number of hidden states

### 5.2.5 Summary of Golf Video Benchmark

In this section, we reuse the same video content that previous work uses to compare with previous work. While producing similar accuracy, our algorithm can process the low-quality version of the video, and thus, being more robust than previous works.

## 5.3 Table Tennis Teaching Video

To show that our method can also process other types of videos, we evaluate our method on another video. Similar to above, we further process the video into low-quality video for evaluation.

### 5.3.1 Video Data Description

The video is a table tennis teaching video. It teaches the audience how to play table tennis. The total length is 1 hour 13 minutes. The original video is a medium quality video. It is TV broadcast quality, but some parts of the video are taken in the early 90s. The resolution is  $480 \times 360$ , and the original fps is 30 frames per second. We further reduce the quality of the video into an even lower quality. See table 5.5 for further technical specifications.

In table 5.5, we include the sample frames and their sizes. For this video, each frame is around 160 times smaller than the original video. Combining with the savings in frame rate, low-quality version of the video could save 480 times of total transmission bandwidth.


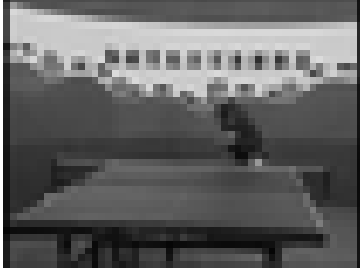
Specification	Original Video	Further Degradation in Our Experiments
Frame Rate	30	8
Color	24 Bit Color	Greyscale
Resolution	$480 \times 360$	$60 \times 40$
Sample Frame		
Size of Sample Frame	337.3 KB	2.1 KB

Table 5.5: Video Quality Degradation for Table Tennis Video

### 5.3.2 Overall Results

#### Accuracy

The table tennis teaching video consists of 2 types of general events: *training* and *match*. The training event means that the coach in the video is demonstrating some table tennis techniques. The match event is an example of the introduced technique. It takes partial video clips of the real matches as examples. However, among all the match events, some match events are different from each other dramatically. This is because: in this video, the example video clips are from several different matches. Each of these matches differs from each other drastically: color backgrounds of these matches are completely different; the playing styles of these matches are different as well. Therefore, we define the teaching video consisting of 5 types of events: *training*, *match-ding*, *match-kong*, *match-persson*, *match-prean*. The match events are separated from each other by different matches. Names of the events are named after the players in the match. In table 5.6, we list all the video events of this video, and also show a sample frame for each video event.

We manually label 67 video events from this video. Similar to above, we use 45 of them for training and 22 of them for testing. Testing scenes do not include the training scenes.

Similar to above tuning stage, we tune the following parameters:

- We use two of hidden states:  $|\mathcal{H}| = 2$






Event	Sample Frame	Description
<i>training</i>		The coach is demonstrating a table tennis technique
<i>match-ding</i>		Example match between Wang, Tao and Ding, Song.
<i>match-kong</i>		Example match between Wang, Liqin and Kong, Linghui.
<i>match-persson</i>		Example match between Wang, Tao and Jörgen Persson.
<i>match-prean</i>		Example match between Wang, Liqin and Carl Prean.

Table 5.6: Events of Table Tennis Teaching Video

	training	match-ding	match-kong	match-persson	match-prean
training	<b>13</b>	0	2	0	1
match-ding	0	<b>1</b>	1	1	0
match-kong	0	0	<b>2</b>	0	0
match-persson	0	1	0	<b>0</b>	0
match-prean	0	0	0	0	<b>0</b>

Table 5.7: Confusion Matrix for Table Tennis Teaching Video

- We set degree of Sparse Coding to 4:  $|K| = 4$

With these parameters, our overall accuracy is 72%.

### Confusion Matrix

Similar to before, we analyze the confusion matrix for table tennis video as well. Table 5.7 is the confusion matrix:

As we mention above, the video consists of two video events: match and training. Since some of the training scenes are vastly different from each other, we divide them into 4 match events. From the confusion matrix we find: our technique can recognize the difference between training and match events quite accurately. Given 13 training events, we never falsely recognize any of them into any match events. However, for match events, we find some matches are easy to confuse with other match events using our method. In the confusion matrix, we find *match-ding* and *match-persson* constantly mixing up in two cases. We also find that, for some matches, it is difficult to distinguish them between match and training. For example, *match-kong* constantly mixes up with the *training* event.

Overall, the false answers by our method are mostly between two different match events. There are only 3 cases that we mis-recognized between a match event and a training event. If we were only to recognize between the training and the match event, then, our overall accuracy could reach around 86%.

### 5.3.3 Summary for Table Tennis Teaching Video

In this section, we use an alternative video for testing. Previous work has not tested this video, and we are the first researchers testing our method on teaching videos. We show



that our method could also work well on this kind of videos.

We find that our technique can accurately detect the difference between training and example match events, although we have difficulties recognizing exactly which match it is for an example match. We show the overall accuracy is 72% amongst 5 video events, but if we were to recognize just between the training and the match events, then the accuracy is around 86%. This suggests that our method can do a very good job in recognizing the major differences in the video events.



# Chapter 6

## Conclusion and Future Work

We contribute a novel solution for Video Event Recognition on low-quality videos. This enables Video Event Recognition to work on a wider set of videos. This thesis contributes a novel feature extraction step and a novel improvement to existing sequence summary model.

By using our novel feature extraction method, we are able to let a computer learn the features on their own from these very low-quality videos (as low as  $36 \times 24$  pixels). We carefully construct the feature functions so that the feature functions produce normalized features. We improve the sequence summary model used in previous work. By adding in hidden states to previous model, we are able to get similar accuracy on a much lower quality videos as compared to previous work.

In our experiments, we find that even for very low-quality videos, we are able to get similar accuracy compared to previous work, which uses a high-quality video that is 100 times clearer. This thesis has achieved its overall goal of providing similar accuracy while eliminating the need for high-quality video. Through detail analysis, we find that low-quality video presents a greater potential challenge, which is that non-related video events are easier to confuse with each other and other types of video events. We are the first researchers on this topic to make such an observation and discovery. We believe further research could lead to a better understanding of such phenomenon in the future. This thesis not only opens Video Event Recognition to a newer data set, but also, more importantly, opens a new door for potential applications. For example, if we consider that Video Event Recognition deployed as a cloud service, the total network transmission could

be drastically reduced due to the fact that we can operate on very low-quality videos. Data transmission can be reduced from hundreds of megabytes to merely a couple of megabytes. Our Video Event Recognition could also be built into a surveillance system that captures videos far away from the target.

The work presented by this thesis is extensible in three directions.

The first thing is to realize these potential applications. In the process of realizing these applications, we will find new constraints in real-life environments. Taking mobile vision as an example, accuracy is less of a concern compared to energy and battery issues, since video processing takes too much of the computational resources. Spending extra energy to optimize for a little accuracy, those improvements might not be worthwhile in that case. This work is similar with the above situations, as we also assume that video quality could be a limiting factor in the real-life environments. We believe, with the new constraints added in, the way we make the trade-off will shift, and the technique to solve these constraints will be different. To discover and solve these constraints, similar work will need to be conducted in the future.

The second direction is to extend our algorithms. So far we have been using Sparse Coding and Representation to extract features, and we have been using HCRF model for sequence summary. Both of these steps have plenty of room for further improvements. This thesis is a pioneering work to solve the low-quality issues of Video Event Recognition. The frameworks and models are not optimized to their limits yet. Optimizing the current model is one way. For example: in the feature extraction step, we could consider using a more powerful model like autoencoder [36], which is an improvement based on sparse coding. In sequence summary models, we could also add state transition functions to represent the temporal locality. Leveraging these existing models and pushing them into a limit could be one way to explore this direction in the future.

Another future direction is to question the generic framework of Video Event Recognition proposed by previous work. It is not quite sure if this framework is optimal or even generic. Facing new challenges like video quality, it might be worthwhile to invent new frameworks and models. Especially, combined the two-step framework into a one-step model is another way to explore this direction.

The third direction is to apply this work to other topics. Almost all of tools and models used by us and existing works have been applied in other topics, such as speech and natural language recognition. The work done in this thesis is aimed to solve Video

Event Recognition only. We showed that, our work has high tolerance to video quality. This suggests, our work is fairly insensitive to background errors. While in topics like speech and natural language recognition, one of the biggest challenges is the background errors. Applying the work done in this thesis might be one direction to solve these challenges in other areas.

Therefore we conclude that, this thesis reaches its goal to recognized video events from low-quality videos, by proposing a novel method. The goal of this thesis has great potential for new applications in Video Event Recognition. The method itself, not only new but also has plenty of room for improvements in future research. The work presented in this thesis also has potential applications in other areas.



# Bibliography

- [1] Y. Feng, X. Wu, H. Wang, and J. Liu, “Multi-group adaptation for event recognition from videos,” in *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pp. 3915–3920, Aug 2014.
- [2] V.-T. Vu, F. Bremond, G. Davini, M. Thonnat, Q.-C. Pham, N. Allezard, P. Sayd, J.-L. Rouas, S. Ambellouis, and A. Flancquart, “Audio-video event recognition system for public transport security,” in *Crime and Security, 2006. The Institution of Engineering and Technology Conference on*, pp. 414–419, June 2006.
- [3] J. Zhou and X.-P. Zhang, “An ica mixture hidden markov model for video content analysis,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, pp. 1576–1586, Nov 2008.
- [4] X. Wang and X.-P. Zhang, “An ica mixture hidden conditional random field model for video event classification,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 23, pp. 46–59, Jan 2013.
- [5] P. Mingtao, W. Yafei, and Z. Meng, “Video events recognition by scene and group context,” *Communications, China*, vol. 10, pp. 165–171, Nov 2013.
- [6] N. Ben Aoun, M. Mejdoub, and C. Ben Amar, “Bag of sub-graphs for video event recognition,” in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 1547–1551, May 2014.
- [7] R. LiKamWa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl, “Energy characterization and optimization of image sensing toward continuous mobile vision,” in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys ’13, (New York, NY, USA), pp. 69–82, ACM, 2013.

- [8] S.-D. Wei and S.-H. Lai, “Robust face recognition under lighting variations,” in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 1, pp. 354–357 Vol.1, Aug 2004.
- [9] J. Wright, A. Yang, A. Ganesh, S. Sastry, and Y. Ma, “Robust face recognition via sparse representation,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, pp. 210–227, Feb 2009.
- [10] P. Viola and M. Jones, “Robust real-time face detection,” in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2, pp. 747–747, 2001.
- [11] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “Maui: Making smartphones last longer with code offload,” in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys ’10*, (New York, NY, USA), pp. 49–62, ACM, 2010.
- [12] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, “Comet: Code offload by migrating execution transparently,” in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, (Hollywood, CA), pp. 93–106, USENIX, 2012.
- [13] I. Zhang, A. Szekeres, D. Van Aken, I. Ackerman, S. D. Gribble, A. Krishnamurthy, and H. M. Levy, “Customizable and extensible deployment for mobile/cloud applications,” in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, OSDI’14*, (Berkeley, CA, USA), pp. 97–112, USENIX Association, 2014.
- [14] J. Li, K. Bu, X. Liu, and B. Xiao, “Enda: Embracing network inconsistency for dynamic application offloading in mobile cloud computing,” in *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing, MCC ’13*, (New York, NY, USA), pp. 39–44, ACM, 2013.
- [15] A. Hyvärinen and E. Oja, “Independent component analysis: Algorithms and applications,” *Neural Netw.*, vol. 13, pp. 411–430, May 2000.
- [16] B. A. Olshausen and D. J. Field, “Sparse coding of sensory inputs,” 2004.



- [17] E. P. Simoncelli and B. Olshausen, “Natural image statistics and neural representation,” *Annual Review of Neuroscience*, vol. 24, pp. 1193–1216, 2001.
- [18] B. A. Olshausen and D. J. Field, “Sparse coding with an overcomplete basis set: a strategy employed by v1?,” *Vision Res*, vol. 37, pp. 3311–25, 1997.
- [19] B. A. Olshausen and D. J. Field, “Emergence of simple-cell receptive field properties by learning a sparse code for natural images,” *Nature*, vol. 381, pp. 607–609, 1996.
- [20] J. Zhou and X.-P. Zhang, “Video shot boundary detection using independent component analysis,” in *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, vol. 2, pp. 541–544, March 2005.
- [21] H. Zhang, A. Kankanhalli, and S. W. Smoliar, “Automatic partitioning of full-motion video,” *Multimedia Syst.*, vol. 1, pp. 10–28, Jan. 1993.
- [22] R. Zabih, J. Miller, and K. Mai, “A feature-based algorithm for detecting and classifying scene breaks,” in *Proceedings of the Third ACM International Conference on Multimedia*, MULTIMEDIA '95, (New York, NY, USA), pp. 189–200, ACM, 1995.
- [23] A. Hanjalic, “Shot-boundary detection: unraveled and resolved?,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 12, pp. 90–105, Feb 2002.
- [24] A. Quattoni, S. Wang, L. Morency, M. Collins, and T. Darrell, “Hidden conditional random fields,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, pp. 1848–1852, Oct 2007.
- [25] S. Dai, Y. Zhan, Q. Mao, and S. Zhang, “A video semantic analysis method based on kernel discriminative sparse representation and weighted knn,” in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CP-SCoM), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pp. 879–886, Aug 2013.
- [26] J. Zhang, D. Zhao, and W. Gao, “Group-based sparse representation for image restoration,” *Image Processing, IEEE Transactions on*, vol. 23, pp. 3336–3351, Aug 2014.

- [27] X. Gao, N. Wang, D. Tao, and X. Li, “Face sketch – photo synthesis and retrieval using sparse representation,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, pp. 1213–1226, Aug 2012.
- [28] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, “Self-taught learning: Transfer learning from unlabeled data,” in *Proceedings of the 24th International Conference on Machine Learning, ICML ’07*, (New York, NY, USA), pp. 759–766, ACM, 2007.
- [29] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” in *Proceedings of the Eighteenth International Conference on Machine Learning, ICML ’01*, (San Francisco, CA, USA), pp. 282–289, Morgan Kaufmann Publishers Inc., 2001.
- [30] C. Sanderson, “Armadillo: C++ linear algebra library.” <http://arma.sourceforge.net/contact.html>.
- [31] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray, “MLPACK: A scalable C++ machine learning library,” *Journal of Machine Learning Research*, vol. 14, pp. 801–805, 2013.
- [32] H. Lee, A. Battle, R. Raina, and A. Y. Ng, “Efficient sparse coding algorithms,” in *Advances in Neural Information Processing Systems 19* (B. Schölkopf, J. Platt, and T. Hoffman, eds.), pp. 801–808, Cambridge, MA: MIT Press, 2007.
- [33] N. Okazaki, “Crfsuite: A fast implementation of conditional random fields.” <http://www.chokkan.org/software/crfsuite/>.
- [34] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, “A limited memory algorithm for bound constrained optimization,” *SIAM J. Sci. Comput.*, vol. 16, pp. 1190–1208, Sept. 1995.
- [35] L. Xie, S.-F. Chang, A. Divakaran, and H. Sun, “Structure analysis of soccer video with hidden markov models,” in *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*, vol. 4, pp. IV–4096–IV–4099, May 2002.
- [36] Y. Bengio, “Learning deep architectures for ai,” *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.