SIGN LANGUAGE RECOGNITION IN A LARGE SCALE SIGN DATABASE

by

PAT JANGYODSUK

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2016

To my family

who set the example and who made me who I am.

ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Vassilis Athitsos for constantly motivating and encouraging me, and also for his invaluable advice during the course of my doctoral studies. I wish to thank my academic advisors Dr. Farhad Kamangar , Dr. Gian Luca Mariottini and Dr. Gautam Das for their interest in my research and for taking time to serve in my dissertation committee.

I would also like to extend my appreciation to National Science Foundation for providing financial support for my doctoral studies. I wish to thank Zhong Zhang, Chirstopher Conly, Soheil Shafiee and Pavlos Doliotis, my fellow Ph.D students, who helped me develop research ideas and results through our discussion and many collaboration works we have done together.

I am grateful to all the teachers who taught me during the years I spent in school, first in Thailand and in the Unites States.

Finally, I would like to express my deep gratitude to my wife who have encouraged and inspired me to pursue graduate studies. I am extremely fortunate to be so blessed. I am also extremely grateful to family for their sacrifice, encouragement and patience. I also thank several of my friends who have helped me throughout my career.

May 5, 2016

ABSTRACT

SIGN LANGUAGE RECOGNITION IN A LARGE SCALE SIGN DATABASE

PAT JANGYODSUK, Ph.D.

The University of Texas at Arlington, 2016

Supervising Professor: Vassilis Athitsos

Recognizing a sign in a sign language video is one of the well known challenging problems in computer vision community. The difficulty arises from many factors including inconsistent sign performing, noisy background, difference in image transformation between training and testing set such as scale, rotation and illumination. One of the most difficult problems, however, is capturing core information features. In most cases, hands are considered the dominant features since sign language usually involve hands movement and shapes.

Having a large scale of a sign database also create another issue, expensive look-up time. As with majority of machine learning application, sign language recognition generally uses one-vs-all approach, where we compute the compatibility score between the given query and every class model and label the query with the class with the highest score. With large number of classes, this results in very inefficient look-up time. As such, efficient indexing is a requirement for the application.

In this dissertation, a sign language recognition application in a large scale system is proposed. The contributions are a random forest hands detector and a fast retrieval indexing method based on hashing. The random forest hands detector

is an extension work of Shotton et al [1] to support RGB videos. The main goal is to label hands pixels whether it is hand pixel or not. Since the focus is on sign language videos, the random offset introduced in Shotton et al [1] has now been extend to 3D space where the third dimension is time, resulting in incremental of features information. The difference between the proposed work and the original work [1] is that i) our work use RGB images as input making the detection accuracy harder due to the fact that depth information is not available and background segmentation is more difficult ii) The propose approach will use 3D offset space. Thus, utilizing time domain information which should result in better accuracy than using only 2D space offset.

The proposed indexing method is based on the concept of filter and refine approach, where candidate signs are first filtered through hash table. Then, the nearest neighbors are found among these candidates. The filtering step is fast since it involves only calculating the hash function of a given query sign. The bottleneck is in refine step where the actual expensive distance/ classification is computed between the query and all objects in candidate set. The contribution is how to define hash functions such that neighbors objects would likely fall into the same hash bucket while minimizing number of objects fallen into the same bucket to reduce the candidate set size. The proposed approach, Distance Based Hashing (DBH), adapt basic geometry properties and machine learning concept to learn such functions.

The experiment is conducted on American Sign Language dataset (ASL) containing 1,113 unique signs from 3 signers making a total of 3,339 videos of signs. It will be done in user independent scenarios where signers used in training set will never appear in testing set.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

ABSTRACT v

LIST OF ILLUSTRATIONS xi

LIST OF TABLES xiii

Chapter Page

1. Introduction 1

2. Preliminary Results - Sign Language Recognition 8

2.1 Sign Language Recognition based on hand trajectories and shapes . . 8

2.2 Sign Language Recognition Overview 9

2.3 Methods 10

2.3.1 RGB-D Calibration Tool 12

2.4 Experiment 14

2.4.1 Dataset 14

2.4.2 Implementation 15

2.5 Results 17

2.5.1 RGB-D Alignment 17

2.5.2 Sign Recognition 17

2.6 Future Works 19

3. Preliminary Results - Distance Based Hashing 20

3.1 Motivation 20

3.2 Locality Sensitive Hashing 22

3.3 Distance-Based Hashing 23

3.3.1 A Distance-Based Family of Hash Functions 24
3.3.2 Differences between LSH and DBH 26
3.3.3 Statistical Analysis of DBH 27
3.3.4 Finding Optimal (k, l) Parameters 31
3.4 Choosing pivot objects and line projections 32
3.4.1 Optimizing line projections selection 33
3.4.2 Optimizing Pivot Objects selection 35
3.4.3 Time Complexity 37
3.5 Additional Optimizations 37
3.5.1 Applying DBH Hierarchically 37
3.5.2 Reducing the Hashing Cost 40
3.6 Applying DBH in Sign Language Recognition Application 41
3.7 Experiments 42
3.7.1 Datasets 42
3.7.2 Implementation Details 46
3.7.3 Results 48
3.8 Conclusions and Future Works 49
4. Preliminary Results - Model Based Search 53
4.1 Sign Recognition Indexing 53
4.2 Model based Search 53
4.3 Motivation 54
4.4 Using JointBoost in Model-Based Search 56
4.5 A Joint Embedding of JointBoost Classifiers and Patterns 58
4.6 Using the Embedding for Efficient Model-Based Search 61
4.7 Applying Model Based Search on Sign Language Recognition 63
4.8 Experiments 65

4.8.1 Dataset . . . 65
4.8.2 JointBoost Implementation . . . 65
4.8.3 Indexing Implementation . . . 66
4.8.4 Baseline Methods . . . 66
4.8.5 Measuring Precision and Recall . . . 67
4.8.6 Results . . . 68
4.9 Conclusions and Future Works . . . 70
5. Hands Detection . . . 72
5.1 Related Work . . . 72
5.2 Background . . . 75
5.2.1 Decision Tree . . . 75
5.2.2 Random Forest . . . 78
5.2.3 Body Part Labeling using Random Forest . . . 79
5.3 Method . . . 81
6. Experiments and Results . . . 85
6.1 Experiments Setup . . . 85
6.2 DataSets . . . 85
6.2.1 American Sign Language Dataset . . . 85
6.2.2 TV Footage Dataset . . . 87
6.2.3 Implementation on ASL dataset . . . 89
6.3 Implementation on TV Footage dataset . . . 91
6.3.1 Comparison Methods . . . 91
6.3.2 Quantitative Measures . . . 94
6.4 Results . . . 95
6.4.1 Pixel-level classification . . . 95
6.4.2 Comparisons with state-of-the-art methods . . . 97

6.4.3 Compare 3D Offset Space with 2D Offset Space 103
6.5 Discussion and Future Works 106
7. Discussion and Conclusions 109
REFERENCES 112

LIST OF ILLUSTRATIONS

Figure		Page
1.1	Sign language images samples	3
2.1	Hand Shapes Visualization	12
2.2	Similar shapes retrieval ranking	13
2.3	Alignment Annotation Tool	14
2.4	ASL Annotated Images	15
2.5	RGB-D Alignment Visualization	17
2.6	DTW Signs Recognition Results	18
3.1	Example of Line Projection Selection	33
3.2	DBH Calculation Matrix Visualization	35
3.3	Example of a normalized Unipen digit.	43
3.4	Example images from the MNIST dataset	44
3.5	The 20 handshapes used in the ASL handshape dataset.	45
3.6	Examples of different appearance of a fixed 3D hand shape	45
3.7	DBH Results	51
3.8	Line Projections and Pivot Objects Selection Results	52
4.1	JointBoost - Precision vs Recall	68
5.1	Articulated hand shapes	73
5.2	Example of a Decision Tree	76
5.3	Example of Tree Inference	76
5.4	Example of a Random Forest	84
6.1	Annotated sample images	86

6.2 TV footage dataset [2] 88
6.3 Hands segment for training examples 89
6.4 Training flowchart 92
6.5 Testing flowchart 93
6.6 Pixel-level results visualization on ASL 95
6.7 ROC Curves on pixel level classification 96
6.8 1-handed signs detection results 98
6.9 2-handed signs detection results 99
6.10 1-handed signs detection results on individual datasets 101
6.11 2-handed signs detection results on individual datasets 102
6.12 Pixel-level classification visualization on TV footage dataset [2] 104
6.13 Results on TV Footage dataset [2] 105
6.14 Results on 3D Offset vs 2D Offset 106

LIST OF TABLES

Table Page

2.1 Signs Retrieval Accuracy 19

4.1 JointBoost speed-up factor 69

6.1 ASL Dataset Statistic 86

CHAPTER 1

Introduction

Sign language recognition or gesture recognition in general, has came to the public attention recently due to the popularity of motion based video games. With the advent of Kinect, the depth information has been made available making tremendous difference in term of perceived information, resulting in overall better accuracy. The importance of the application includes automatic sign dictionary database, gestured based application and ultimately, automatic real-time sign language translator. However, as an ongoing research works, there are many problems regarding the application, namely, features extraction, recognition and indexing.

In term of similarity to other applications, sign language recognition is usually compared to action recognition problems since both are classification problem given a motion video as a query. Both problems consists of extracting motion features from videos and typically, use a time series model to classify the action. However, sign language recognition remains as the more challenging the problem in the author's opinion, as followings.

1. The variation between classes in sign language application is typically smaller than that of action recognition. For examples, actions includes eating or jumping have large distinct difference in term of motion direction whereas the difference in sign, as shown in figure 1.1 can be minimal in term of hands shape variations.
2. The number of classes in sign language recognition is larger by some margin comparing to action recognition application in general. For examples, the American

Sign Language dataset (ASL) consists of 1,113 signs whereas the typical number of actions is mostly limited to a factor of 10 classes. If we do a random guess classifier on ASL dataset, we will only get 0.089% accuracy. As such, classifying in a sign language application is considered harder in general due to larger number of classes.

3. Number of training data per class in sign language is usually small due to the unavailability of data. Consequently, this results in a poor and overfitting parameters approximation for one-vs-all models including the popular Hidden Markov Model and Conditional Random Field.
4. In action recognition, most prominent features is motion velocity based features such as optical flow since the direction and volume of motion yield information regarding actions. However, this does not apply to sign language. Some signs are more focusing on hands trajectory where hand shapes are totally irrelevant whereas others focus solely on hands shapes and hands trajectory does not yield any discriminating information at all. With this in mind, there is no one true best features for all signs.

As usual with all of machine learning applications, the first step is extracting features. Usually, the process of extracting features start from detecting interest points, the definition of interest points is given as image windows where interesting features to the application are located. Here we have a lot of options, famous ones include SIFT [3], mo-SIFT [4], and STIP [5] interest points. However, the most prominent interest points in sign language recognition are hands and other body parts related to sign such as face and arms. The obvious reasons is, as a human, hand shape, hands movement and facial expression are what sign made up from. After detecting interest points, the next step is extracting features from the interest areas which can be hand location, shape or motion.

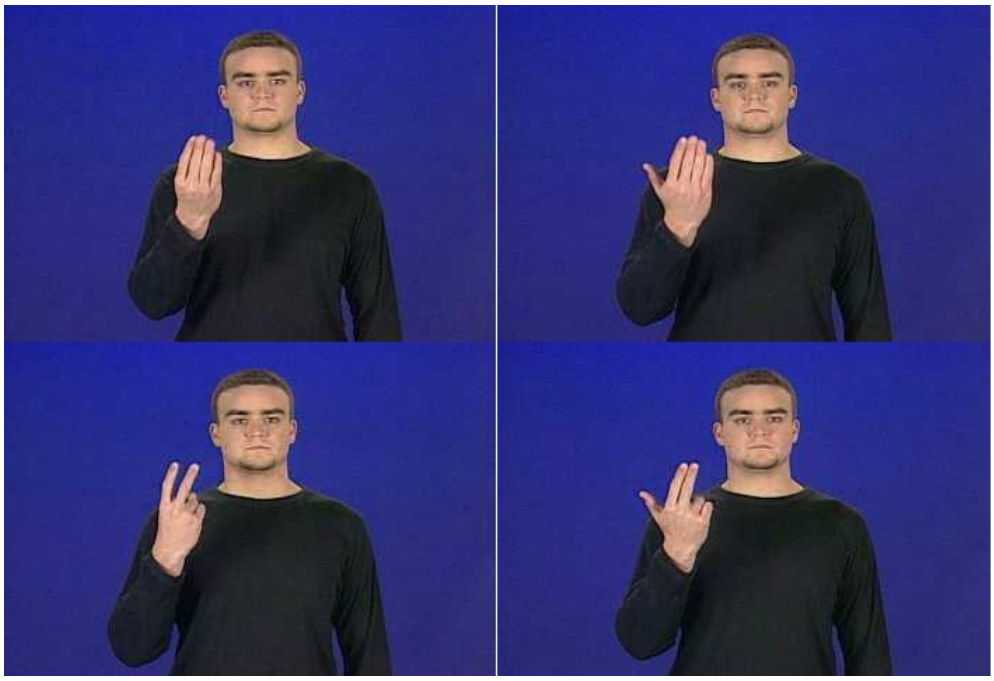

Figure 1.1: Sample images showing that even minor difference in hand shapes are interpreted as different signs. This makes extracting discriminating features in sign language recognition a significantly challenging problem.

Unfortunately, hands detection remains one of the well known hard problem in computer vision as follows. Basically, when one want to detect a certain object in an image, the most straightforward solution is using template matching method such as the famous Viola Jones detector [6]. In template matching approach, a trained template representing targeted objects shape is applied on the query image. The window where the compatible score (usually, this is classification score or normalized correlation) is high is considered the objects. To cope with image transformation, we'll just try a variety of different trained models. For example, view point difference is handled by training many models where each model is trained on different viewpoints. During testing, all the trained models will be applied to the given image.

That being said, however, the template matching approach cannot be applied to hands detection problem. With hands, they can be in any articulated shapes making

the using one model per viewpoint approach does not work since there will be too many models to train.

Then, there is a recognition problem. Once we extract features from a query video. We need to classify if the given sign fall into which class. essentially, the problem of recognizing gesture is a class of time series classification. Inspired by speech recognition, the most popular model is Hidden Markov Model and its variation [7, 8, 9, 10, 11]. Dynamic Time Warping, a time series matching algorithm, is also a popular choice [12, 13, 14, 15] due to the fact that it is a distance measurement and therefore, no training required making it a perfect choice for application where number of training is small including ours. In more recent work, Conditional Random Field (CRF) [16] and Hidden Conditional Random Field (H-CRF) [17] improves beyond Hidden Markov Model by removing 1-to-1 dependency between a hidden state and observation making the overall accuracy better. However, both CRF and H-CRF still requires large number of training in order to train a good model.

In our application, we use Dynamic Time Warping (DTW) as a recognition method since we only have 2 training examples per class. As such, methods that requires large training examples per class to estimate good model parameters such as Hidden Markov Model (HMM) and Conditional Random Field is not suitable for our dataset.

Finally, there are problem of indexing in large scale sign database. Generally, the number of classes in sign language application is large. With one-vs-all recognition approach such as Dynamic Time Warping (DTW), Hidden Markov Model (HMM) [18] and Conditional Random Field (CRF) [19]. One would need to compute compatible score, such as, classification score, distance or similarity measure between given query sign and every class model. The problem is that such computation is expensive and

it is even more pronounced when you need to do computation for all classes. To make the application runtime feasible, efficient indexing is a necessity.

Summarizing all steps on building a full sign language recognition application, firstly, hands locations, our interest regions, are detected with a hands detector. This is where the major contribution of this dissertation comes from. Next, we extract features around detected hands region. More information on which features to select will be explained later in the chapter 2. Then, we align the query sign with sign model from sign database using Dynamic Time Warping (DTW) algorithm, which is a time series alignment algorithm generating optimal alignment distance between 2 signs. Then, use one-vs-all approach to determine which sign to classify the query sign to by labeling the query to the class (sign) with minimum DTW distance between the query and that particular class model. However, there is a problem with this approach, particularly, the part of using one-vs-all approach. To compute Dynamic Time Warping distance, it requires $O(MND)$ where M is an average number of frames per a query sign, N is a number of an average frames per class model sign and D is number of features dimension. Roughly speaking, this is a cubic time algorithm. As such, the computational time is expensive. Using one-vs-all approach makes the total computation becomes $O(MNDP)$ where P is number of classes. In our dataset, we have a total of 1,113 signs but we have 2 class models per sign making $P = 2,226$, a significantly large number. Therefore, a good indexing method is a necessity.

In chapter 3, we introduce a new efficient way of finding nearest neighbors problem called Distance Based Hashing. The main idea of Distance Based Hashing is, given a distance matrix between pair of objects, construct a binary function approximating the original distance space between pair of objects. Then we build hash table using a series of binary function as a hash function. If the binary function is a good approximation on the original space, then, chances are, the query object will collide

on the same hash bucket as its nearest neighbors. Objects fallen into same buckets as the query object are considered nearest neighbor candidate Finally, we refine the result by finding true neighbors on the original space among these candidates.

How does Distance Based Hashing related to sign language recognition? As mentioned, sign language recognition using one-vs-all approach can be considered as the same problem as finding nearest neighbors where signs are objects, and distance metric is Dynamic Time Warping distance. As such, we can pre-compute a distance matrix for all pairs of signs based on DTW distance between pairs. Having a distance matrix, hash tables can be built as described previously. During recognition time, given a query sign, we compute a hash value of a query sign to find nearest signs candidates fallen into same hash buckets. From there, we find the true most similar sign by computing DTW distance between the query sign and these signs candidates. We classify the query as the sign with the smallest distance. The detail of Distance Based Hashing will be greatly discussed in chapter 3.

Let us take a look at an alternative approach to sign indexing problem. Usually, sign language recognition is one type of classification problem, where given an object, classify the object into class label. To this end, there are numerous approaches for speeding up multi-class classification process. However, given one sample of query sign might not provide enough information for accurate classification. If given multiple query samples means providing more information, we can train a query classifier based on these multiple query samples and use the trained classifier as query. In chapter 4, we propose an indexing method to this searching approach called model based search. The idea is, assuming that the classifier is a JointBoost classifier, then, we show that we can embed objects and classifiers onto the same Euclidean space. We also prove that the problem of classifying an object using JointBoost classifier is the same as computing Euclidean distance in embedded space between a classifier and

and an object. As such, classification task becomes finding nearest neighbors in the embedded space. Being a finding nearest neighbors problem in Euclidean space, we can simply adapt PCA to reduce the dimensional space for efficient neighbors search calculation.

In relation between model based search and sign language recognition, assuming that a user is willing to supply a system with a few query samples rather than a single one, we can train a classifier on these query samples. Then, we can apply model based search idea to speeding up sign classification. The detail of implementation will be described in more detail in chapter 4.

In this dissertation, the goal is building a sign language recognition in large scale system, where the major contribution lies within hands detection method. The following chapters is arranged as follow, first we discussed the previous results on sign language recognition, follows by explaining indexing scheme namely Distance Based Hashing and Model Based Search, and finally, give more detailed on the proposed method on hands detection.

CHAPTER 2

Preliminary Results - Sign Language Recognition

2.1 Sign Language Recognition based on hand trajectories and shapes

In the field of computer vision, sign language recognition remains one of the most challenging tasks. With the recent release of the Microsoft Kinect camera, depth-sensing technology has become widely available at an affordable price. The depth camera provides an additional dimension beyond that of RGB images so that a pixel becomes a point in 3D space instead of color intensity values. This greatly increases information found in features, leading to better recognition accuracy. The problem associated with RGB-D input, however, is pixel alignment between the depth and color images. Since the two sensors are separated by a physical distance, their perception of the scene is from a slightly different perspective, and there is not a one-to-one correspondence between pixels in the two images. In most cases, people use only the depth information, as properly calibrating the RGB and depth cameras is a non-trivial task. By discarding the RGB image, valuable information is lost that cannot necessarily be acquired from the depth image, such as hand shape, since the Kinect depth image is captured in low resolution. Therefore, object shape, like that of the hands, cannot be captured in detail. Here, we implemented a simple calibration tool that approximates 4 alignment parameters, including x-translation, y-translation, x-scale and y-scale, so that the researcher can align the two images and utilize both depth and color information.

We also improve sign language recognition rate over that of the work from [15]. In [15], the query sign was recognized using Dynamic Time Warping as a distance

measure between hand trajectories. They also compared hand shape in both the first and last frame of signs using the Euclidean distance between shape features described in the paper. While the shape features used in [15] are able to improve accuracy, they do not utilize gradient (edge) information that is the core feature in popular descriptors such as SIFT or HoG. We applied the same method, DTW, for hands trajectory comparison but use Histogram of Oriented Gradient (HoG) [20] to represent hand shape instead and achieve better results. Accuracy increases, on average, about 10%.

2.2 Sign Language Recognition Overview

The most common problems regarding sign language and gesture recognition, in general, are:

1. Image transformation between training and query data. Scale, rotation, affine and illumination, for example.
2. Noise in training and query data. While this problem is common in any machine learning application, it is exaggerated in computer vision applications, where the majority of information is unrelated to the task at hand.
3. Temporal segmentation of the gesture. When does the gesture start and stop?

The solutions to these problems lie in either features or recognition method.

While detecting features, a difficulty arises from the fact that, in a video, there is a lot of irrelevant information–for example, background, face, and cloth. As such, when extracting features, only body parts performing the gesture should be considered. This, however, is not a trivial task. Methods abound for hand detection, ranging from simple skin and motion models [21, 22] to shape-based template search [23]. Contextual search using graphical models has been popular in recent years, for example chain model [24] and skeleton tracking on Kinect depth images [25]. Using a

depth camera such as the Kinect eases some difficulty in computer vision application [26], as additional dimensional information is available. However, finding hands is still an ongoing research problem.

An alternative approach to searching for individual body parts is to extract features such as HoG [20] features from the whole frame. While this approach does not suffer from the difficulties found in body part or interest point extraction, it does capture noise and, thus, is not tolerance of image transformation.

In gesture recognition methods, the problem is viewed as one application of time series classification. Inspired by speech recognition, the most popular model is Hidden Markov Model (HMM) and its variations [7, 8, 9, 10, 11]. Dynamic Time Warping (DTW), a time series matching algorithm, is also a popular choice [12, 13, 14, 15] due to the fact that it is a distance measurement and therefore, no training is required, making it a perfect choice for application where the number of training examples is small, as is the case with ours. In more recent work, Conditional Random Field (CRF) [16] and Hidden Conditional Random Field (H-CRF) [17] improves upon HMM by removing the 1-to-1 dependency between a hidden state and observation, thus increasing overall accuracy. However, both CRF and H-CRF require a large number of training examples in order to learn a good model.

2.3 Methods

To recognize a sign, we use two kinds of features. The first one is hand trajectory. As in [15], the features we use for single-handed signs are:

1. Hand location with respect to face position

2. Unit hand motion velocity vector. Mathematically, given hand location $h(t)$, at frame t, this feature is

$$v(t) = \frac{h(t+1) - h(t-1)}{||h(t+1) - h(t-1)||}$$

For 2-handed signs, the features are:

1. Right hand location with respect to face position
2. Unit right hand motion velocity vector.
3. Left hand location with respect to face position
4. Unit left hand motion velocity vector.
5. Right hand position with respect to left hand, designated by $f(t)$
6. Unit motion vector of $f(t)$, given as

$$d(t) = \frac{f(t+1) - f(t-1)}{||f(t+1) - f(t-1)||}$$

We extract the trajectory feature described above from any given query video sign. This feature will be compared to those in the database training signs using DTW to match hand trajectories.

The second part of our algorithm is hand shape matching. Given a hand region on the first frame and the last frame of the sign, we extract features describing hand shape. The features used in the experiment are Histogram of Oriented Gradient (HoG) features. We chose HoG features for robustness against illumination changes and for recent success and popularity in many computer vision applications. Again, the HoG features will be matched with signs in the database using Euclidean distance as a distance metric. Note that the sign type of the query (whether it is one-handed or two-handed) must also be given so that each type is only matched against others of the same type.

According to some initial experiments, HoG features tend to give good results based on shape similarity as shown in figure .2.2

Figure 2.1: Hand shape sample images from the ASL data set. The color images are the annotated hand regions in the color frame. The grayscale images are the corresponding visualizations of extracted HoG features using inverse HoG [27].

To recognize a given sign, Q, we retrieve the top k sign candidates, $\mathbb{S} = \{S_1, S_2, ..., S_k\}$, using the matching method described above. It is considered correctly classified if

$$\exists S_i \in \mathbb{S}, C(S_i) = C(Q)$$

where $C(X)$ is a function returning the class of a given video X.

2.3.1 RGB-D Calibration Tool

As mentioned, we cannot apply the annotated bounding box from the depth image to the RGB image directly, due to mis-alignment of two images. Calibrating the Kinect camera is a non-trivial task. We propose a simple alignment annotation tool that approximates calibration parameters. These parameters are x-translation, y-translation, x-scale and y-scale. Note that the purpose of the tool is *NOT* to replace proper camera calibration. It is just for fast, rough and simple approximation of alignment parameters.

Figure 2.3 shows a screen shot of the alignment tool. To use the tool, the RGB frame and a segment extracted from the corresponding depth frame must be

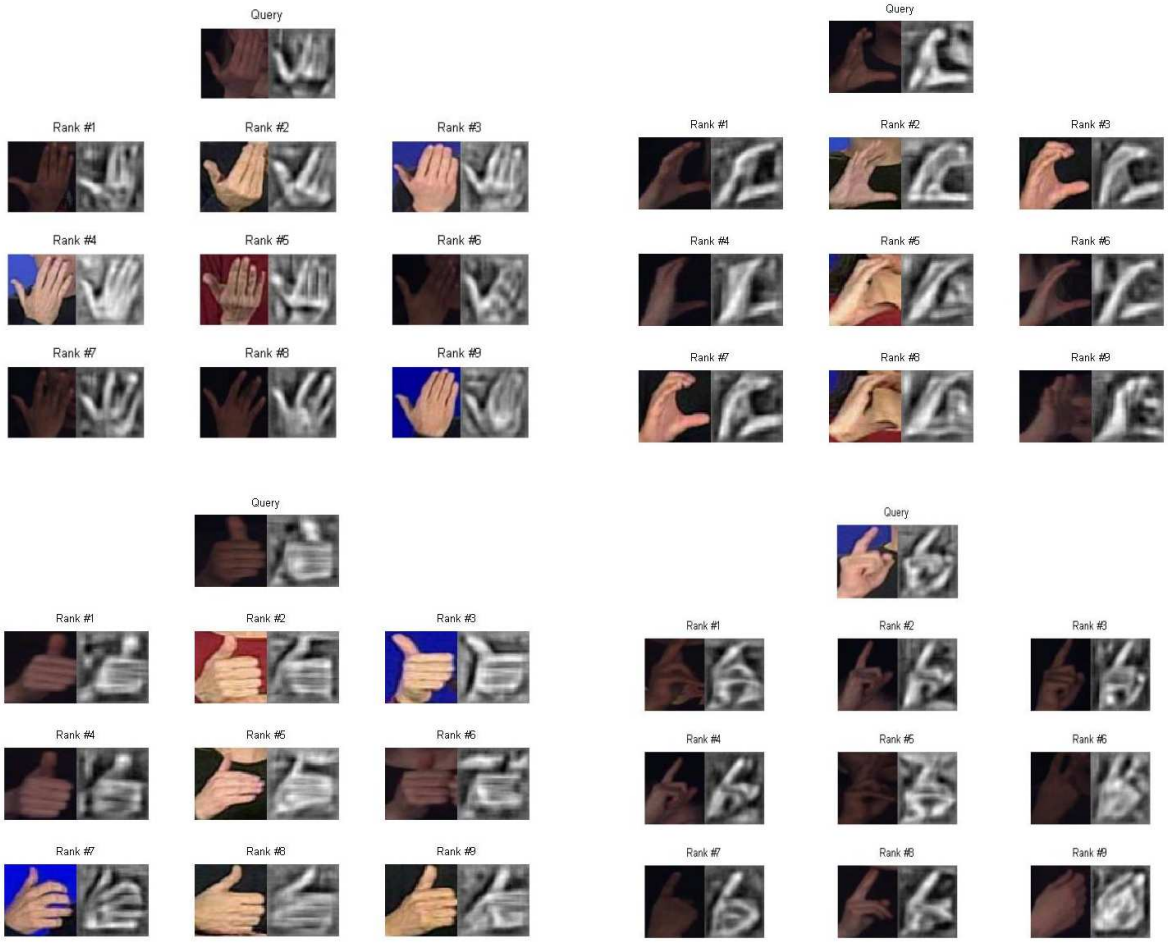

Figure 2.2: Similar shapes retrieval given a query as a HoG features representation. The corresponding right images are HoG visualization [27]

provided. In our case, we use body segmentation extracted from depth image. The annotator will align the segment to match that of RGB by adjusting translation and scale parameters such that it matches as they see fit. As can be seen in figure 6.1, the body segment extracted from depth image has been manually aligned to the RGB image. To learn the alignment parameters of a camera on a specific view, the

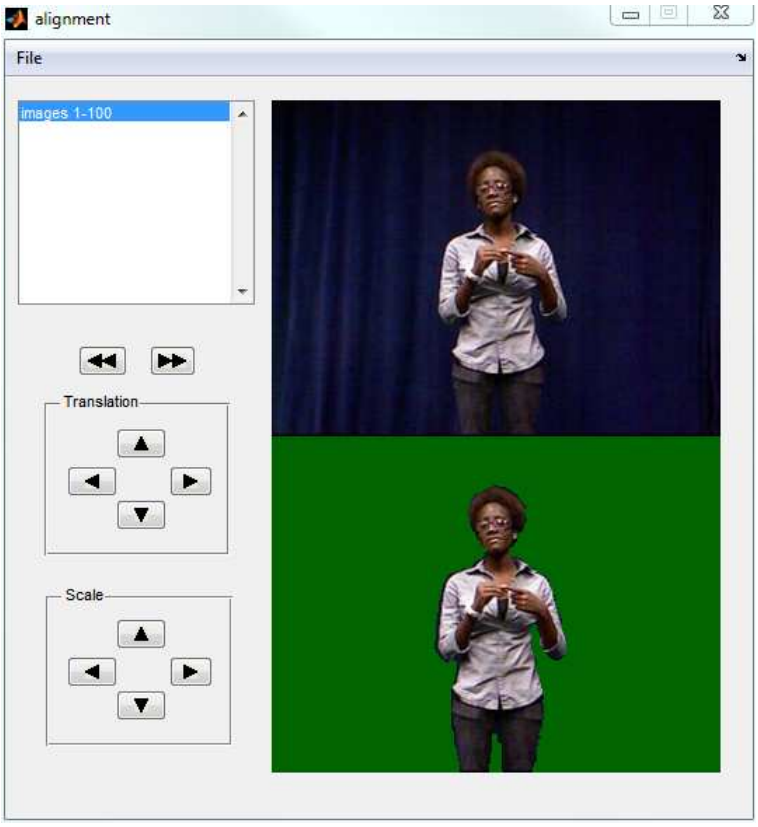

Figure 2.3: Sample screen shot from the alignment annotation tool. The bottom image is the working space where a human annotator adjusts the segment extracted from the corresponding depth frame to match the RGB frame. In the figure, this segment is the body segment. The top image serves as a reference image.

annotator will perform the manual alignment for a certain number of frames (50 in our experiment). The final parameters are the average values.

2.4 Experiment

2.4.1 Dataset

We conducted the experiment on an American Sign Language (ASL) data set consisting of 1,113 signs. There are 2 types of data.

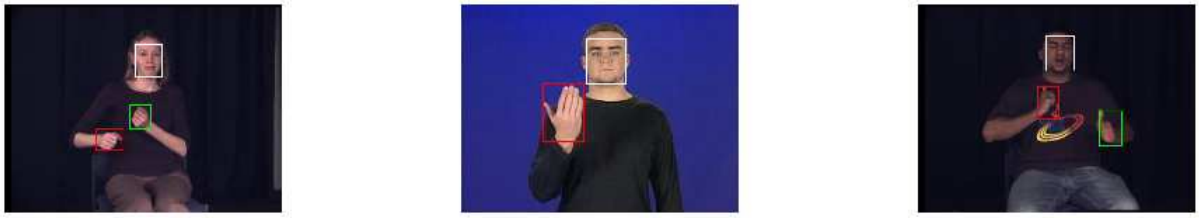

Figure 2.4: Sample images with annotated regions from the ASL data set. Each image is from a different signer. The rectangles bound various regions, including face and hands.

1. Videos from a data set captured by a standard RGB camera. No depth information is available. We have 3 signers for this type of data. Each signer performs 1,113 signs, making a total of 3,339 signs. This data set is called ASL data set.
2. Videos from a data set captured by a Kinect camera [28]. There are 2 signers for this data set for a total of 2,226 signs, called jku449.

Along with the sign videos, we also have bounding box annotations for the hands and face regions. The ASL data set videos have annotations for all 1,113 signs by all 3 signers, while the jku449 data set currently has annotations for 449 signs by one signer.

2.4.2 Implementation

As mentioned in section 2.3, sign recognition is done using DTW and shape matching based on HoG features. The feature used for DTW trajectory matching is the same that were used in [15]. In addition, we have made some minor improvements by standardizing features so that the mean value is 0 and the standard deviation is 1.

HoG features parameters were extracted using inverse HoG code [27] from MIT. With ASL data set, the hand regions were extracted using manually annotated bound-

ing boxes from the RGB images. The same cannot be done with jku449 data set since body part labeling was done on the depth images. If we extracted the shape from the depth images, we would not have accurate shape information due to the fact that depth images lack visual appearance information. As mentioned previously, we used our alignment annotation tool to learn estimated alignment parameters and applied the parameters to the depth image annotated bounding boxes. Ideally, the result is bounding boxes properly aligned with the hand regions in the RGB images.

The experiment was performed in a user-independent fashion. For ASL data set, we used signs from one signer as queries and compared them to signs from 2 other signers. The result for ASL data set was the average from all 3 signers. Since we only have annotation from one signer for jku449 data set, the query signs are compared to videos from ASL data set. To extract hand shape features for jku449 data set, we first applied the alignment parameters on the depth image annotated hand bounding boxes to get the hand region in the RGB image. Then, we extracted HoG features on the hand region from the RGB image. The quantitative measurement used was accuracy-top candidates retrieval plot. For each data set, we implement 3 methods to compare.

1. Hand trajectory matching with hand shape distance using HoG features as shape representation
2. Hand trajectory matching with hand shape distance using features in [15] as shape representation
3. Hand trajectory matching without hand shape distance.

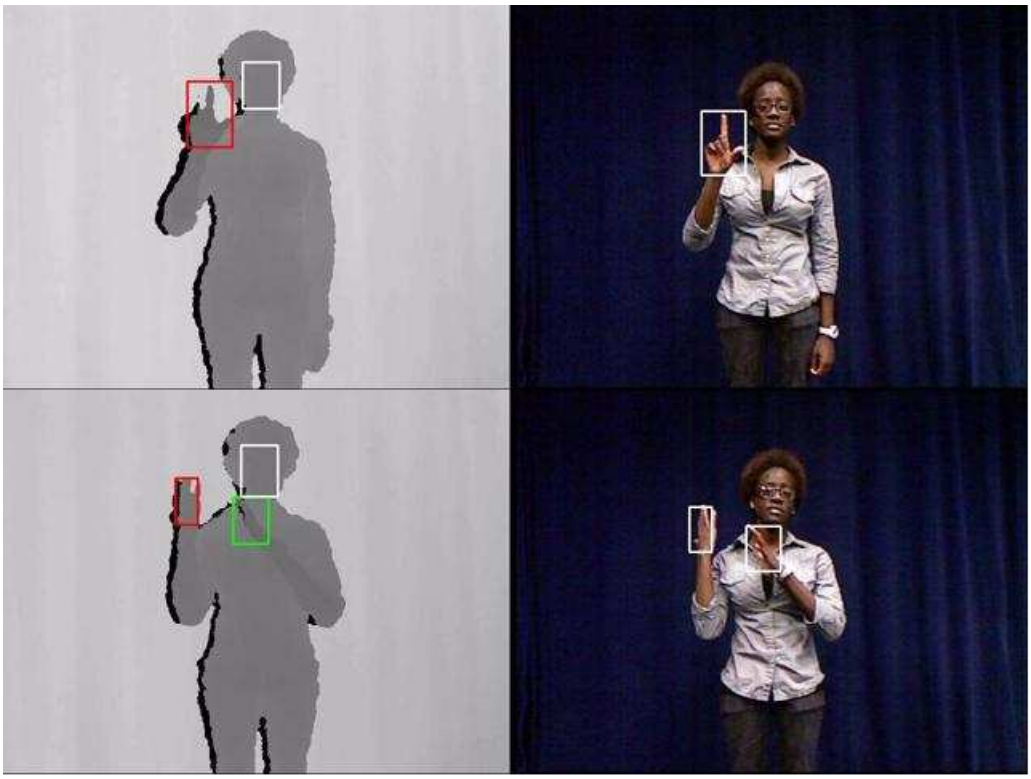

Figure 2.5: Visualization of RGB-D alignment. Left image is the manually annotated bounding box made on top of the depth frame. The right image is the bounding box after applying alignment parameters on top of corresponding RGB frame. It can be seen that the bounding box does encompass the majority of hand pixels

2.5 Results

2.5.1 RGB-D Alignment

Figure 2.5 shows examples of RGB-D alignment. It can be seen that the bounding box, while not perfect, captures the majority pixels belonging to the hands.

2.5.2 Sign Recognition

Figure 2.6 displays sign recognition accuracy. The x-axis represents the top retrieved candidate signs and y-axis is accuracy. The legend is in format 'Data set - hand shape features'. It can be seen that hand shape comparison does increase the accuracy by more than 10%. Using HoG for shape representation, the accuracy

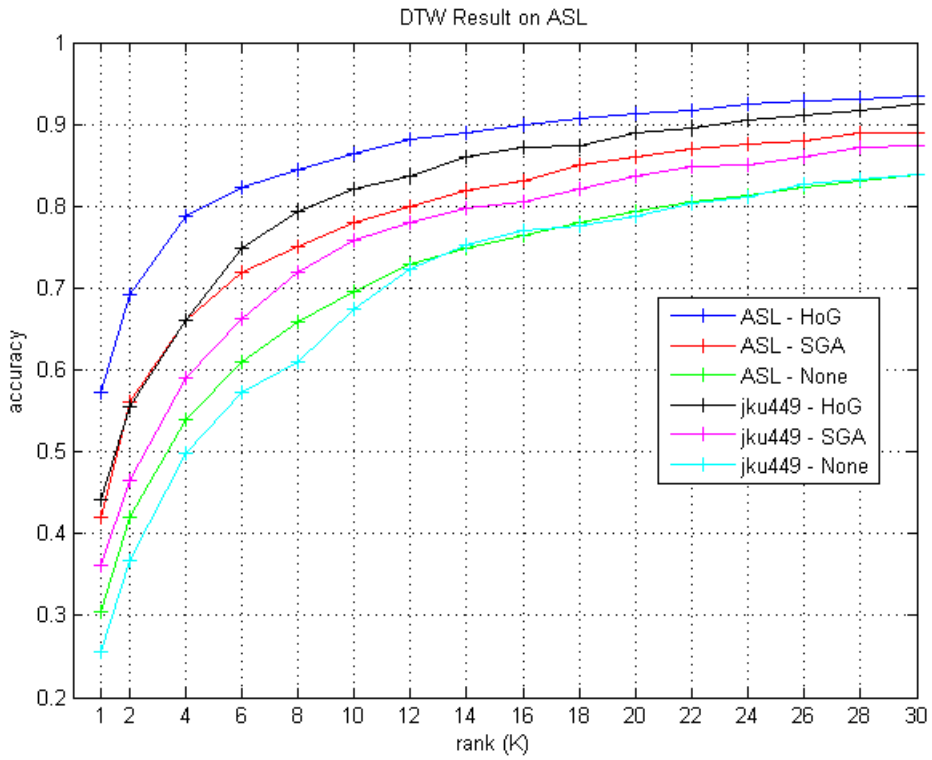

Figure 2.6: Sign recognition results for ASL and jku449 data set. The x-axis represents top retrieval rank and y-axis represents recognition accuracy. The legend is in the format 'Data set - Hand shape features'.

improves over using the shape presented in [15] by about 8%. At top 10 candidates retrieval, we achieved 86% accuracy compared to 78% in [15].

Accuracy on the jku449 data set is on average about 2-3% lower than that of ASL when using same method. At top 10 rank, the accuracy is 82% for jku449 data set. This is because the estimated calibration parameters, while proving to work well, are not perfect. Therefore, the extracted hand regions obtained from the color images are not always accurate. It can be seen that, without hand shape comparison, the results of ASL (green line) and jku449 (cyan line) are similar but begin to differ when hand shape is considered.

Top k	1	3	5	10	20	30
ASL - HoG	57.29%	73.94%	80.56%	86.43%	91.25%	93.38%
ASL - SGA	42%	61%	69%	78%	86%	89%
ASL - No shape	30.37%	48.05%	57.5%	69.63%	79.37%	83.95%
jku449 - HoG	44.18%	60.75%	70.45%	82.09%	88.96%	92.54%
jku449 - SGA	36.12%	52.84%	62.69%	75.82%	83.58%	87.46%
jku449 - None	25.67%	43.28%	53.58%	67.46%	78.81%	83.88%

Table 2.1: Sign retrieval accuracy in number. Top k refers to number of best matches

2.6 Future Works

There are a number of things left for future work. The simplest one is, using Kinect data for queries, extend the trajectory feature into 3D space. This, in theory, should give better accuracy due to the fact that more information is provided. Another idea would be to conduct a comprehensive experiment using other kinds of features or recognition methods. For instance, HMM and CRF for the recognition method or SIFT for hand shape features. Finally, we will work towards recognizing signs without user-provided information or annotations, such as hand bounding boxes, temporal segmentation and sign type.

CHAPTER 3

Preliminary Results - Distance Based Hashing

In this chapter, we described an efficient indexing scheme called Distance Based Hashing.

3.1 Motivation

In computer vision applications, answering a nearest neighbor query consists of identifying, for a given query object, the most similar database objects. Nearest neighbor retrieval is a common and indispensable operation in a wide variety of real systems. A few example applications are similar images search, handpose classification or nearest neighbor classification in general (e.g., [29, 30, 31]). Given ever-increasing database sizes, there is a need for efficient and scalable indexing methods, that can facilitate accurate and efficient nearest neighbor retrieval.

Locality Sensitive Hashing (LSH) [32, 33] is a framework for hash-based indexing, with appealing theoretical properties and empirical performance. LSH is an approximate technique; it does not guarantee finding the true nearest neighbor for 100% of the queries. At the same time, LSH provides a statistical guarantee of producing a correct result with high probability. Theoretically, for a database of n vectors of d dimensions, the time complexity of finding the nearest neighbor of an object using LSH is sublinear in n and only polynomial in d. The theoretical advantages of LSH have been also empirically demonstrated in several applications involving high-dimensional data [34, 32, 30, 35, 31].

A key requirement for applying LSH to a particular space and distance measure is to identify a family of *locality sensitive* functions, satisfying the properties specified in [32]. As a consequence, LSH is only applicable for specific spaces and distance measures where such families of functions have been identified, such as real vector spaces with L_p distance measures, bit vectors with the Hamming distance, or strings with a substitution-based distance measure (that does not allow insertions or deletions) [36, 37]. This is in contrast to distance-based indexing methods, that build indexing structures based only on distances between objects, and thus can be readily applied to any space and distance measure.

We introduce Distance-Based Hashing (DBH), a novel indexing method for efficient approximate nearest neighbor retrieval. Compared to LSH, DBH has several similarities but also some important differences. Overall , the main novelties of DBH are the following:

- DBH is a hash-based indexing method that is *distance-based*. Consequently, DBH can be applied in arbitrary (and not necessarily metric) spaces and distance measures, whereas LSH cannot.
- Indexing performance (in terms of retrieval accuracy and retrieval efficiency) is estimated and optimized using statistics obtained from sample data, whereas in LSH performance guarantees are obtained by using some known geometric properties of a specific space and distance measure. Dependence on known geometric properties is exactly what makes LSH not applicable in arbitrary spaces.

Experiments with several real-world data sets demonstrate that DBH provides very good trade-offs between retrieval accuracy and efficiency, and that DBH outperforms VP-trees, a well-known distance-based method for indexing arbitrary spaces. Furthermore, no known method exists for applying LSH on those data sets, and this

fact further demonstrates the need for a distance-based hashing scheme that DBH addresses.

3.2 Locality Sensitive Hashing

Let $\mathbb{X}$ be a space of objects, to which database and query objects belong. Let D be a distance measure defined on $\mathbb{X}$. In this work, we also use notation $(\mathbb{X}, D)$ to jointly specify the space and distance measure. Let $\mathcal{H}$ be a family of hash functions $h : \mathbb{X} \to \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers. As described in [32], $\mathcal{H}$ is called *locality sensitive* if there exist real numbers r_1, r_2, p_1, p_2 such that $r_1 < r_2$, $p_1 > p_2$, and for any $X_1, X_2 \in \mathbb{X}$:

$$D(X_1, X_2) < r_1 \Rightarrow \Pr_{h \in \mathcal{H}}(h(X_1) = h(X_2)) \geq p_1 \ . \tag{3.1}$$

$$D(X_1, X_2) > r_2 \Rightarrow \Pr_{h \in \mathcal{H}}(h(X_1) = h(X_2)) \leq p_2 \ . \tag{3.2}$$

Given a locality sensitive family $\mathcal{H}$, Locality Sensitive Hashing (LSH) indexing works as follows: first, we pick integers k and l. Then, we construct l hash functions $g_1, g_2, \ldots, g_l$, as concatenations of k functions chosen randomly from $\mathcal{H}$:

$$g_i(X) = (h_{i1}(X), h_{i2}(X), \ldots, h_{ik}(X)) \ . \tag{3.3}$$

Each database object X is stored in each of the l hash tables defined by the functions g_i. Given a query object $Q \in \mathbb{X}$, the retrieval process first identifies all database objects that fall in the same bucket as Q in at least one of the l hash tables, and then exact distances are measured between the query and those database objects.

As shown in [32], if k and l are chosen appropriately, then a near neighbor of Q is retrieved with high probability (note that LSH is *not* an exact indexing method, as it may produce the wrong result for some queries). The method can be applied both for near-neighbor retrieval (for range queries) and nearest-neighbor retrieval (for

similarity queries). In Euclidean space $\mathbb{R}^d$, the time complexity of retrieval using LSH is linear in the dimensionality d and sublinear in the number n of database objects [33].

Applying the LSH framework to a specific space and distance measure requires identifying a locality sensitive family $\mathcal{H}$. Such families have been identified for certain spaces, such as vector spaces with L_p metrics [33, 32], or strings with a substitution-based distance measure [36, 37]. An improvement that can drastically reduce the memory requirements of LSH in Euclidean spaces is described in [38].

3.3 Distance-Based Hashing

In this section we introduce Distance-Based Hashing (DBH), a method for applying hash-based indexing in arbitrary spaces and distance measures. In order to make our method applicable to arbitrary spaces, a key requirement is to use the distance measure as a black box. Therefore, the definition of the hash functions should only depend on distances between objects. To keep the method general, no additional assumptions are made about the distance measure. In particular, the distance measure is *not* assumed to have Euclidean or metric properties.

The first step in our formulation is to propose a family $\mathcal{H}$ of hash functions. These functions are indeed defined using only distances between objects, and thus they can be defined in arbitrary spaces. The second and final step is to introduce a framework for analyzing indexing performance and picking parameters. We shall see that the proposed family $\mathcal{H}$ of hash functions is *not* always locality sensitive (depending on the space and distance measure), and therefore our method cannot be analyzed using the LSH framework. Consequently, we introduce a different frame-

work, whereby indexing behavior is analyzed using statistical data collected from sample objects of $\mathbb{X}$.

3.3.1 A Distance-Based Family of Hash Functions

In existing literature, several methods have been proposed for defining functions that map an arbitrary space $(\mathbb{X}, D)$ into the real line $\mathbb{R}$. An example is the pseudo line projections proposed in [39]: given two arbitrary objects $X_1, X_2 \in \mathbb{X}$, we define a "line projection" function $F^{X_1,X_2} : \mathbb{X} \to \mathbb{R}$ as follows:

$$F^{X_1,X_2}(X) = \frac{D(X,X_1)^2 + D(X_1,X_2)^2 - D(X,X_2)^2}{2D(X_1,X_2)} \ . \tag{3.4}$$

If $(\mathbb{X}, D)$ is a Euclidean space, then $F^{X_1,X_2}(X)$ computes the projection of point X on the unique line defined by points X_1 and X_2. If $\mathbb{X}$ is a general non-Euclidean space, then $F^{X_1,X_2}(X)$ does not have a geometric interpretation. However, as long as a distance measure D is available, F^{X_1,X_2} can still be defined and provides a simple way to project $\mathbb{X}$ into $\mathbb{R}$.

We should note that the family of functions defined using Equation 3.4 is a very rich family. Any pair of objects defines a different function. Given a database $\mathbb{U}$ of n objects, we can define about $n^2/2$ unique functions by applying Equation 3.4 to pairs of objects from $\mathbb{U}$.

Functions defined using Equation 3.4 are real-valued, whereas hash functions need to be discrete-valued. We can easily obtain discrete-valued hash functions from F^{X_1,X_2} using thresholds $t_1, t_2 \in \mathbb{R}$:

$$F^{X_1,X_2}_{t_1,t_2}(X) = \begin{cases} 0 & \text{if } F^{X_1,X_2}(X) \in [t_1,t_2] \ . \\ 1 & \text{otherwise} \ . \end{cases} \tag{3.5}$$

In practice, t_1 and t_2 should be chosen so that $F^{X_1,X_2}_{t_1,t_2}(X)$ maps approximately half the objects in $\mathbb{X}$ to 0 and half to 1, so that we can build balanced hash tables.

We can formalize this notion by defining, for each pair $X_1, X_2 \in \mathbb{X}$, the set $\mathbb{V}(X_1, X_2)$ of intervals $[t_1, t_2]$ such that $F_{t_1,t_2}^{X_1,X_2}(X)$ splits the space in half:

$$\mathbb{V}(X_1, X_2) = \{[t_1, t_2] | \Pr_{X \in \mathbb{X}}(F_{t_1,t_2}^{X_1,X_2}(X) = 0) = 0.5\} \ . \tag{3.6}$$

Note that, in most cases, for every t there exists a t' such that F^{X_1,X_2} maps half the objects of $\mathbb{X}$ either to $[t, t']$ or to [t', t]. For a set of n objects, there are $n/2$ ways to split those objects into two equal-sized subsets (if n is even) based on the choice of $[t_1, t_2] \in \mathbb{V}(X_1, X_2)$. One of several alternatives is to choose an interval $[t_1, \infty]$ such that $F^{X_1,X_2}(X)$ is less than t_1 for half the objects $X \in \mathbb{X}$. Another alternative is to choose an interval $[t_1, t_2]$ such that, using F^{X_1,X_2}, one sixth of the objects in X are mapped to a value less than t_1 and two sixths of the objects are mapped to a value greater than t_2. The set $\mathbb{V}(X_1, X_2)$ includes intervals for all these possible ways to split X into two equal subsets.

Using the above definitions, we are now ready to define a family $\mathcal{H}_{\text{DBH}}$ of hash functions for an arbitrary space $(\mathbb{X}, D)$:

$$\mathcal{H}_{\text{DBH}} = \{F_{t_1,t_2}^{X_1,X_2} | X_1, X_2 \in \mathbb{X}, [t_1, t_2] \in \mathbb{V}(X_1, X_2)\} \ . \tag{3.7}$$

Selecting some binary hash functions h from $\mathcal{H}_{\text{DBH}}$ we can define k-bit hash functions g_i by applying Equation 3.3. This way, indexing and retrieval can be performed as in LSH, by:

- Choosing parameters k and l.
- Constructing l k-bit hash tables, and storing pointers to each database object at the appropriate l buckets.
- Comparing the query object with the database objects found in the l hash table buckets that the query is mapped to.

3.3.2 Differences between LSH and DBH

In the previous paragraphs we have defined a distance-based indexing scheme that uses hash functions. We call that method Distance-Based Hashing (DBH). What DBH has in common with LSH is the indexing structure: we define l hash tables using l hash functions g_i, and each g_i is a concatenation of k simple, discrete-valued (binary, in our case) functions $h \in \mathcal{H}_{\text{DBH}}$.

If the function family $\mathcal{H}_{\text{DBH}}$ were locality sensitive, then DBH would be a special case of LSH, and we would be able to use the LSH framework to optimally pick parameters k and l and provide guarantees of accuracy and efficiency. The main difference between DBH and LSH stems from the fact that we do not assume $\mathcal{H}_{\text{DBH}}$ to be locality sensitive. Whether $\mathcal{H}_{\text{DBH}}$ is actually locality sensitive or not depends on the underlying space and distance measure. Since we want to use DBH for indexing arbitrary spaces, we need to provide a method for analyzing performance without requiring $\mathcal{H}_{\text{DBH}}$ to be locality sensitive.

From an alternative perspective the difference between LSH and DBH is that applying LSH on a particular space requires knowledge of the geometry of that space. This knowledge is used to construct a family $\mathcal{H}$ of hash functions for that space and to prove that $\mathcal{H}$ is locality sensitive. If the goal is to design an indexing scheme for arbitrary spaces, then clearly no geometric information can be exploited, since arbitrary spaces have arbitrary geometries.

A simple example to illustrate that the family $\mathcal{H}_{\text{DBH}}$ defined in Section 3.3.1 is not always locality sensitive is the following: let us construct a finite space $(\mathbb{X}, D)$, by defining a distance matrix M, where entry $M_{i,j}$ is the distance $D(X_i, X_j)$ between the i-th and j-th object of $\mathbb{X}$. We set the diagonal entries $M_{i,i}$ to zero, we set all off-diagonal entries to random numbers from the interval $[1, 2]$, and we enforce that

M be symmetric. Under that construction, space $(\mathbb{X}, D)$ is metric, as it satisfies symmetry and the triangle inequality.

In such a scenario, for any two objects $X_i, X_j \in \mathbb{X}$, the probability $\Pr_{h \in \mathcal{H}_{\text{DBH}}}(h(X_i) = h(X_j))$ does not depend at all on the distance between X_i and X_j, and in practice $\Pr_{h \in \mathcal{H}_{\text{DBH}}}(h(X_i) = h(X_j))$ is expected to be very close to 0.5, especially as the size of $\mathbb{X}$ becomes larger. Consequently, regardless of our choice of r_1 and r_2, there is no reason for appropriate p_1, p_2 to exist so as to satisfy the locality sensitive conditions expressed in Equations 3.1 and 3.2.

More generally, the random manner in which we constructed matrix M violates the fundamental assumption of any distance-based indexing method: the assumption that knowing $D(X_i, X_j)$ and $D(X_j, X_k)$ provides useful information/constraints about $D(X_i, X_k)$. The reason that distance-based methods work in practice is that, in many metric and nonmetric spaces of interest, distances are indeed not random, and knowing distances between some pairs of objects we can obtain useful information about distances between other pairs of objects.

Based on the above discussion, designing a useful distance-based indexing method requires identifying and exploiting the information that distances between objects provide, when such information is indeed present. When geometric constraints (such as Euclidean properties and/or the triangle inequality) are not available, we can still exploit statistical information obtained from sample data, i.e., from known objects sampled from the space of interest. We now proceed to describe how to obtain and use such statistical information in the context of DBH.

3.3.3 Statistical Analysis of DBH

An important question in analyzing any indexing scheme is identifying the factors that determine indexing performance, i.e., the factors that determine:

- Retrieval accuracy: how often is the true nearest neighbor retrieved using this indexing scheme?
- Retrieval efficiency: how much time does nearest neighbor retrieval take? What fraction of the database is pruned by the indexing scheme?

We now proceed to perform this analysis for DBH.

As before, let $(\mathbb{X}, D)$ be the underlying space and distance measure. Let $\mathbb{U} \subset \mathbb{X}$ be a database of objects from $\mathbb{X}$. Let $\mathcal{H}_{\text{DBH}}$ be the family of binary hash functions defined in Equation 3.7. A key quantity for analyzing the behavior of DBH is the probability $C(X_1, X_2)$ of collision between any two objects of $\mathbb{X}$ over all binary hash functions in $\mathcal{H}_{\text{DBH}}$:

$$C(X_1, X_2) = \Pr_{h \in \mathcal{H}_{\text{DBH}}}(h(X_1) = h(X_2)) \ . \tag{3.8}$$

Given family $\mathcal{H}_{\text{DBH}}$ and the two objects X_1 and X_2, quantity $C(X_1, X_2)$ can be measured directly by applying all functions or a sample of functions $h \in \mathcal{H}_{\text{DBH}}$ to X_1 and X_2, if $\mathcal{H}_{\text{DBH}}$ is finite.

Suppose that we would like to take it to another detail step by computing, $C(X_1, X_2, L)$, the collision probability on a particular line projection, L, what will be a good way to do it. The simplest approach is to get the average value from all possible binary hash functions generated from the line projection. However, on a single line projection, the number of binary hashes generating from it could be massive depending on the choice of t_1 and t_2 and, thus, impractical to try all of them.

Alternatively, $C(X_1, X_2, L)$ can also be estimated in a different way. Instead of trying all binary hash functions generated from a line projection, we can use the fact that the choice of t_2 depends on t_1 (t_2 is selected such that $[t_1, t_2]$ covers 50% of line projection space). Therefore, the only choice actually made is t_1. Suppose that we project $X_1, X_2 \in \mathbb{X}$ to line projection L, if $t_1 \in [F^L(X_1), F^L(X_2)]$, then X_1

and X_2 are not collided. Making this to a discrete case, X_1 and X_2 are not collided if $t_1 \in [I(F^L(X_1)), I(F^L(X_2))]$ where $I(X)$ returns the sorted projection position of $X \in \mathbb{X}$ on L.

Based on the intuition above, we defined the collision probability over a line projection as:

$$C(X_1, X_2, L) = \frac{|\mathbb{X}| - 2|I(F^L(X_1) - I(F^L(X_2)|}{|\mathbb{X}|} \tag{3.9}$$

To sum up, we do the following to obtain $C(X_1, X_2, L)$.

- Project training objects onto a line projection
- Sort those objects by their projection value
- Each object is now have its own position index. Now, we can apply Equation 3.9 to any pair of objects to get the value.

The benefit of this function is we can estimate the collision probability on a particular line projection. Furthermore, the estimation is less prone to overfitting due to the choice of t_1 and t_2. Additionally, the collision probability of two objects X_1 and X_2 over a set of line projections $\mathbb{L}$, $C_{\mathbb{L}}(X_1, X_2)$ can be derived as

$$C_{\mathbb{L}}(X_1, X_2) = \int_{L \in \mathbb{L}} C(X_1, X_2, L) Pr(L) dL \tag{3.10}$$

Suppose that we have chosen parameters k and l, and that we construct l k-bit hash tables by choosing randomly, uniformly, and with replacement, kl functions from $\mathcal{H}_{\text{DBH}}$. The probability $C_k(X_1, X_2)$ of collision between two objects on a k-bit hash table is:

$$C_k(X_1, X_2) = C(X_1, X_2)^k \ . \tag{3.11}$$

Finally, the probability $C_{k,l}(X_1, X_2)$ that two objects collide in at least one of the l hash tables is:

$$C_{k,l}(X_1, X_2) = 1 - (1 - C(X_1, X_2)^k)^l \ . \tag{3.12}$$

Suppose that we have a database $\mathbb{U} \subset \mathbb{X}$ of finite size $n = |\mathbb{U}|$, and let $Q \in \mathbb{X}$ be a query object. We denote by $N(Q)$ the nearest neighbor of Q in $\mathbb{U}$. The probability that we will successfully retrieve $N(Q)$ using DBH is simply $C_{k,l}(Q, N(Q))$. The accuracy of DBH, i.e., the probability over all queries Q that we will retrieve the nearest neighbor $N(Q)$ is:

$$\text{Accuracy}_{k,l} = \int_{Q \in \mathbb{X}} C_{k,l}(Q, N(Q)) \Pr(Q) dQ \ , \tag{3.13}$$

where $\Pr(Q)$ is the probability density of Q being chosen as a query. This probability density is assumed to be uniform in the rest of this chapter.

Quantity $\text{Accuracy}_{k,l}$ can be easily estimated by:

1. sampling queries $Q \in \mathbb{X}$,
2. finding the nearest neighbors $N(Q)$ of those queries in the database $\mathbb{U}$,
3. estimating $C(Q, N(Q))$ for each sample Q by sampling from $\mathcal{H}_{\text{DBH}}$,
4. using the estimated $C(Q, N(Q))$, and applying Equations 3.11 and 3.12 to compute $C_{k,l}(Q, N(Q))$ for each sample Q, and
5. computing the average value of $C_{k,l}(Q, N(Q))$ over all sample queries Q.

Besides accuracy, the other important performance measure for DBH is efficiency. In particular, we want to know how many database objects we need to consider for each query using DBH. Naturally, in brute force search we need to consider every single database object. The expected number of database objects we need to consider for a query Q is denoted as $\text{LookupCost}(Q)$ and is simply the expected number of objects that fall in the same bucket with Q in at least one of the l hash tables. This quantity can be computed as:

$$\text{LookupCost}_{k,l}(Q) = \sum_{X \in \mathbb{U}} C_{k,l}(Q, X) \ . \tag{3.14}$$

For efficiency, an estimate for $\text{LookupCost}(Q)$ can be computed based on a sample of database objects, as opposed to computing $C_{k,l}(Q, X)$ for all database objects $X \in \mathbb{U}$.

An additional cost incurred by retrieval using DBH is the cost of computing the outputs $g_i(Q)$ of the l k-bit hash functions g_i. Overall, we need to apply kl binary hash functions $h \in \mathcal{H}_{\text{DBH}}$ on Q. Since each such function h is of the form specified in Equation 3.5, computing such an $h(Q)$ involves computing the distances $D(Q, X_1)$ and $D(Q, X_2)$ between the query and the two objects X_1 and X_2 used to define h. We denote by $\text{HashCost}_{k,l}$ the number of such distances we need to compute, in order to compute $h(Q)$ for all binary hash functions. Note that $\text{HashCost}_{k,l}$ is independent of the query Q, as $\text{HashCost}_{k,l}$ is simply the number of unique objects used as X_1 and X_2 in the definitions of the kl binary hash functions h. In the worst case, $\text{HashCost}_{k,l} = 2kl$, but in practice $\text{HashCost}_{k,l}$ can be smaller because the same object X can be used as X_1 or X_2 in the definitions of multiple binary hash functions h.

The total cost $\text{Cost}_{k,l}(Q)$ of processing a query is therefore the sum of the two separate costs:

$$\text{Cost}_{k,l}(Q) = \text{LookupCost}_{k,l}(Q) + \text{HashCost}_{k,l} \ . \tag{3.15}$$

Finally, the average query cost can be computed using sample queries, as was done for computing indexing accuracy. In particular:

$$\text{Cost}_{k,l} = \int_{Q \in \mathbb{X}} \text{Cost}_{k,l}(Q) \text{Pr}(Q) dQ \ . \tag{3.16}$$

In conclusion, the accuracy and efficiency of DBH, given parameters k and l, can be measured by sampling from the space of queries, sampling from the set of database objects, and sampling from the set $\mathcal{H}_{\text{DBH}}$ of binary hash functions.

3.3.4 Finding Optimal (k, l) Parameters

Given parameter k, clearly indexing accuracy increases and efficiency decreases as we increase l. Consequently, given a desired retrieval accuracy, and given k, we can

choose l by computing $\text{Accuracy}_{k,l}$ for $l = 1, 2, \ldots$ until we identify an l that yields the desired accuracy. Instead of successively measuring accuracy for each l, binary search can also be used, as a more efficient method for identifying the smallest l that yields the desired accuracy.

To find the optimal k we repeat the above process (of searching for an l given k) for different values $k = 1, 2, \ldots$. Different pairs of k, l that yield roughly the same indexing accuracy $\text{Accuracy}_{k,l}$ are likely to yield different costs $\text{Cost}_{k,l}$. Thus it is beneficial to choose the combination of k, l that, while achieving the desired accuracy, minimizes $\text{Cost}_{k,l}$. In practice, for a given accuracy, as we consider $k = 1, 2, \ldots$, efficiency typically improves up to a point and then it starts decreasing. Therefore, the optimal k can be identified as the last k for which efficiency improves.

In summary, given a desired retrieval accuracy rate, the optimal parameters k and l can be computed by searching over possible k and l and identifying the combination that, while yielding the desired accuracy, also maximizes efficiency. The accuracy and efficiency attained for each k, l pair is estimated as described in Section 3.3.3. Computing the optimal k and l is naturally done off-line, as a preprocessing step, and the costs of that computation have no bearing on the cost $\text{Cost}_{k,l}$ of the online retrieval stage.

3.4 Choosing pivot objects and line projections

Given a candidate set of line projection L, we wish to find an optimal set of line projections $\mathbb{L}, \mathbb{L} \subseteq \text{L}$. In our implementation, we first select a set of training objects as pivot objects. L is line projections formed by these objects.

In this section, we defined how to optimally select pivot objects and line projections. Line projections and pivot objects choice can be critical to retrieval performance. Figure 3.1 shows the difference when selecting good and bad line projections.

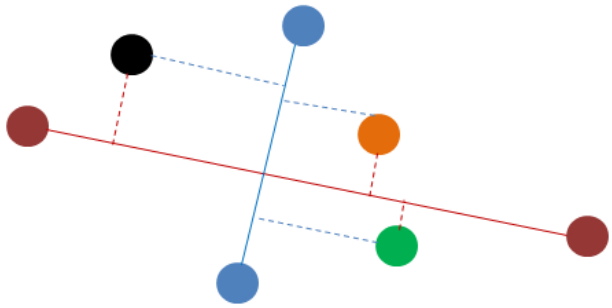

Figure 3.1: Example of good and bad selection of line projection. Orange and green objects are nearest neighbor of each other. Red line projection can preserve the original distance. Thus, projection distance of (orange, green) is the smallest of all creating good binary hash function. On other hand, blue line projection does not.

We could see that the blue line projection does not capture the original distance. For example, consider orange and green objects pair which are the nearest neighbor of each other, the projection distance of (orange, green) on blue line is larger than that of (orange, black). Thus, the projection distance does not reflect the original one. In contrast red line projection is much better as we can see that projection distance of (orange, green) is much lesser than that of (orange, black) and (green, black).

3.4.1 Optimizing line projections selection

Recall that a binary hash function is essentially a line projection applied $[t_1, t_2]$ threshold interval. Therefore, choosing good line projections will impact directly on both accuracy and efficiency. To solve this problem, the first thing to consider is how to measure the quality of a line projection. A good way to start is measuring the collision probability of a particular line projection, $C(X_1, X_2, L)$. From there, we can compute accuracy and cost given k and l parameters.

To select best line projections, we will use greedy strategy as follows. Given a number of selected line projections T, we shall run our algorithm T times. Each

time, one line projection is selected greedily and added to the set of already selected line projections from previous rounds $\mathbb{L}_{t-1}$.

At round t^{th}, we will choose a line projection such that it will compliment best with a set of already selected line projections from previous rounds $\mathbb{L}_{t-1}$. By compliment term, two values must be considered, accuracy and cost. We can derive those two values using Equation 3.13 and 3.16. The only change needed is replacing $C(X_1, X_2)$ to $C_{\mathbb{L}_t}(X_1, X_2)$ in Equation 3.12 to reflect the quality of $\mathbb{L}_t$ selected so far. As in optimizing (k, l), parameter k and desired accuracy are given, we will select a line projection such that it yields minimum cost while still passing the accuracy threshold. For each line projection candidates, we will find l parameter that gives minimum accuracy passing the threshold. Then, we calculate its cost. The one with minimum cost is selected at that particular round. Psuedo code of the algorithm is given in algorithm 1.

How about running time? Since most calculation is done using $C(X_1, X_2, L)$ in every single round, using pre-computed collision probability matrix should greatly save redundant computation. We compute a 3D array where rows and columns corresponds to training objects and depth coresponds to line projections. Entry (i, j, k) of the array corresponds to $C(X_i, X_j, L_k)$ In addition, we need another matrix storing $C_{\mathbb{L}_t}(X_i, X_j)$, current collision probability for all pairs at round t. The calculation of $C_{\mathbb{L}_{i+1}}$ can be done in linear time to the size of training set by including $C(X_i, X_j, L_{t^*})$ to the corresponding entry, where L_{t^*} is the selected line projection at that round. Figure 3.2 visualized these arrays. In addition, we can use sampling to reduce the size of line projection candidates in each round to further speed up the execution.

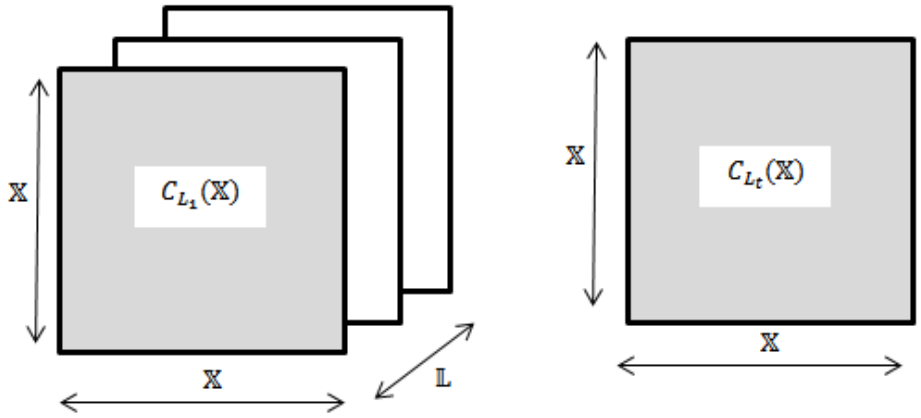

Figure 3.2: Visualization of calculation arrays. The left one is a 3D pre-computed array where row and column corresponds to training objects, depth corresponds to line projection candidates. Entry (i, j, k) is $C_{L_k}(X_i, X_j)$. The right matrix is the current collision probability matrix storing $C_{\mathbb{L}_t}(\mathbb{X})$ where $\mathbb{L}_t$ is the selected line projection so far. Rows and columns corresponds to training objects. This matrix will be updated in every round

3.4.2 Optimizing Pivot Objects selection

Similarly to optimizing line projections, same strategy is used to select pivot objects one by one greedily. However, the problem becomes more difficult as number of all line projections created from a set of pivot objects could be massive.

We denote pivot object candidate, $P \in \mathbb{P}$, $\mathbb{P} \subseteq \mathbb{U}$. To solve the problem, we will use the relax version of the algorithm. On the first round, one pair of pivot objects is selected randomly to avoid high computational cost from having to iterate through all line projections formed by all pairs of candidate pivot objects.

The pre-computed arrays are also different. One of the dimension is now represents each pivot object candidate instead of a line projection candidate. Let $\mathbb{P}_t = \{P_1, P_2, ...P_t\}$ denote the set of selected pivot objects selected so far at round t^{th}, $line(X_1, X_2)$ denotes line projection formed by object X_1 and X_2. At round t^{th}, the (i, j, k) entry of pre-computed array stores $C_{\mathbb{L}_{t,k}}(X_i, X_j)$ where $\mathbb{L}_{t,k} = \{line(P_k, P_1), line(P_k, P_2), ...line(P_k, P_t)\}$. In other words, this is the set of

Algorithm 1 Selecting line projections

Require: parameters k, desired accuracy $accuracy$

$\mathbb{L} \leftarrow \phi$

$C_{\mathbb{L}}(\mathbb{X}) \leftarrow 0$ ▷ initialize all entries to zero

for $t = 1 \rightarrow noDesiredCandidates$ **do**

  for $j = 1 \rightarrow noCandidates$ **do**

    $C_j \leftarrow C_{\mathbb{L} \cup L_j}(\mathbb{X})$

    $l \leftarrow getCompliment\_l(k, accuracy, C_j)$

    $Cost_j \leftarrow \int_{Q \in \mathbb{X}} \mathrm{Cost}_{k,l}(Q)\mathrm{Pr}(Q)dQ$

  end for

  $j^* \leftarrow argmin_j(Cost_j)$

  $C_{\mathbb{L}}(\mathbb{X}) \leftarrow C_{\mathbb{L} \cup L_j^*}(\mathbb{X})$ ▷ update collision matrix

  $\mathbb{L} \leftarrow \mathbb{L} \cup L_{j^*}$

end for

return $\mathbb{L}$

line projections formed by P_k to all selected pivot objects so far. For instances, at round 4^{th}, we have selected $\{X_3, X_6, X_7\}$ as pivot objects. Row 1^{st} of pre-computed matrix store collision probability of $\{line(X_1, X_3), line(X_1, X_6), line(X_1, X_7)\}$. If X_1 is selected at this round, $C_{\mathbb{P}_t}$ will include new 3 line projections corresponding to all line projections formed by the selected set. It should be mentioned that the pre-computed matrix must be updated at every round to include recently added line projection. This way, values in pre-computed array is ready to be used as it represents all line projections if it (pivot

object) is going to be chosen in the subsequent rounds. Thus, we can still update the current collision array in linear time. The psuedo-code is shown in algorithm 2.

3.4.3 Time Complexity

Time complexity of optimizing algorithm for both line projections and pivot objects is $O(TN^2)$, where T is number of desired rounds, N is number of candidates. The running time is quadratic to number of training objects. Thus, the size of training objects is crucial to training time. In previous section, we have mentioned that we use object sampling to reduce training time. With sampling, the complexity is reduced to $O(TD^2)$ where D is the size of sampling set. If $D << N$, the running time gap can be big.

It should be noted that all training objects are still utilized even if we used sampling by the following reason. Instead of sampling objects initially before the main loops, we re-sample objects in every optimization round. Since we sample different set of objects in each round, all training objects are used during the process.

3.5 Additional Optimizations

The previous section described a complete implementation of DBH. In this section we consider some practical methods for further improving performance. In particular, we describe a way to apply DBH in a hierarchical manner, using multiple pairs of (k, l) parameters, and we also describe a practical method for drastically reducing $\text{HashCost}_{k,l}$.

3.5.1 Applying DBH Hierarchically

The accuracy and efficiency of DBH for a particular query object Q essentially depends on the collision rate $C(Q, N(Q))$ between the query and its nearest neighbor,

Algorithm 2 Selecting pivot objects

Require: parameters k, desired accuracy *accuracy*

$(P_x, P_y) \leftarrow SelectPairRandomly()$

$C_{line(\mathbb{P})}(\mathbb{X}) \leftarrow C_{line(P_x,P_y)}(\mathbb{X})$

$\mathbb{P} \leftarrow \{P_x, P_y\}$

▷ Initialize pre-computed matrix

for $j = 1 \rightarrow noPivotCandidates$ **do**

$C_{L_j}(\mathbb{X}) \leftarrow C_{line(P_j,P_x) \cup line(P_j,P_y)}(\mathbb{X})$

end for

for $t = 3 \rightarrow noDesiredCandidates$ **do**

for $j = 1 \rightarrow noPivotCandidates$ **do**

$C_j \leftarrow C_{line(\mathbb{P}) \cup L_j}(\mathbb{X})$

$l \leftarrow getCompliment\_l(k, accuracy, C_j)$

$Cost_j \leftarrow \int_{Q \in \mathbb{X}} \mathrm{Cost}_{k,l}(Q) \mathrm{Pr}(Q) dQ$

end for

$t^* \leftarrow argmin_j(Cost_j)$

$C_{line(\mathbb{P})}(\mathbb{X}) \leftarrow C_{line(\mathbb{P}) \cup L_t^*}(\mathbb{X})$ ▷ update collision matrix

$\mathbb{P} \leftarrow \mathbb{P} \cup P_{t^*}$

▷ update pre-computed matrix

for $j = 1 \rightarrow noCandidates$ **do**

$C_{L_j}(\mathbb{X}) \leftarrow C_{L_j \cup line(P_j,P_{t^*})}(\mathbb{X})$

end for

end for

return $\mathbb{P}$

and the collision rates $C(Q, X)$ between Q and the rest of the database objects $X \in \mathbb{U}$. In an arbitrary space $\mathbb{X}$, without a priori knowledge of the geometry of that space, these collision rates can only be estimated statistically, and they can differ widely for different query objects.

The key motivation for designing a hierarchical version of DBH is the observation that, typically, different choices of k and l may be optimal for different query objects. Empirically, we have found that the optimal choice of k and l depends mainly on the distance $D(Q, N(Q))$. This correlation makes sense intuitively: the closer two objects are to each other the more likely it is that these objects are mapped to the same bit by a random binary hash function. Therefore, as $D(Q, N(Q))$ decreases, we expect the optimal parameters k and l for that query object to lead to increasingly fewer collisions for the same indexing accuracy.

Based on the above observations, a natural strategy is to create multiple DBH indexes, so that each index is optimized for a different set of queries and corresponds to a different choice of parameters k, l. In particular, we rank query objects Q according to $D(Q, N(Q))$, and we divide the space $\mathbb{X}$ of possible queries into disjoint subsets $\mathbb{X}_1, \mathbb{X}_2, \ldots, \mathbb{X}_s$, so that $\mathbb{X}_i$ contains queries ranked in the top $(i-1)/s$ to i/s percentiles according to $D(Q, N(Q))$. Then, given the database $\mathbb{U}$ and the desired accuracy rate, we choose optimal parameters k_i and l_i for each query set $\mathbb{X}_i$, and we create a DBH index structure for that query set. We denote by D_i the smallest value such that for all objects $Q \in \mathbb{X}_i$ it holds that $D(Q, N(Q)) \leq D_i$.

Naturally, at runtime, given a previously unseen query object Q, we cannot know what $\mathbb{X}_i$ Q belongs to, since we do not know $D(Q, N(Q))$. What we can do is perform nearest neighbor retrieval successively using the DBH indexes created for $\mathbb{X}_1, \mathbb{X}_2, \ldots$ If using the DBH index created for $\mathbb{X}_i$ we retrieve a database object X such that $D(Q, X) \leq D_i$, then we know that $D(Q, N(Q)) \leq D(Q, X) \leq D_i$. In

that case, the retrieval process does not proceed to the DBH index for $\mathbb{X}_{i+1}$, and the system simply returns the nearest neighbor found so far, using the DBH indexes for $\mathbb{X}_1, \ldots, \mathbb{X}_i$.

In practice, what we typically observe with this hierarchical scheme is this: the first DBH indexes, designed for queries with small $D(Q, N(Q))$, successfully retrieve (at the desired accuracy rate) the nearest neighbors for such queries, while achieving a lookup cost much lower than that of using a single global DBH index. For query objects Q with large $D(Q, N(Q))$, in addition to the lookup cost incurred while using the DBH index for that particular $D(Q, N(Q))$, the hierarchical process also incurs the lookup cost of using the previous DBH indexes as well. However, we expect this additional lookup cost to be small, since the previous DBH indexes typically lead to significantly fewer collisions for objects with large $D(Q, N(Q))$. So, overall, compared to using a global DBH index, the hierarchical scheme should significantly improve efficiency for queries with low $D(Q, N(Q))$, and only mildly decrease efficiency for queries with high $D(Q, N(Q))$.

3.5.2 Reducing the Hashing Cost

As described in Section 3.3.3, the hashing cost $\text{HashCost}_{k,l}$ is the number of unique objects used as X_1 and X_2 in the definitions of the kl binary functions needed to construct the DBH index. If those kl binary functions are picked randomly from the space of all possible such functions, then we expect $\text{HashCost}_{k,l}$ to be close to $2kl$. In practice, we can significantly reduce this cost, by changing the definition of $\mathcal{H}_{\text{DBH}}$.

In Section 3.3.1 we defined $\mathcal{H}_{\text{DBH}}$ to be the set of all possible functions $F^{X_1,X_2}_{t_1,t_2}$ defined using any $X_1, X_2 \in \mathbb{X}$. In practice, however, we can obtain a sufficiently large and rich family $\mathcal{H}_{\text{DBH}}$ using a relatively small subset $\mathbb{X}_{\text{small}} \subset \mathbb{X}$:

$$\mathcal{H}_{\text{DBH}} = \{F^{X_1,X_2}_{t_1,t_2} \mid X_1, X_2 \in \mathbb{X}_{\text{small}}, \\ [t_1, t_2] \in \mathbb{V}(X_1, X_2)\} \ . \tag{3.17}$$

If we use the above definition, the number of functions in $\mathcal{H}_{\text{DBH}}$ is at least equal to the number of unique pairs X_1, X_2 we can choose from $\mathbb{X}_{\text{small}}$, and is actually larger in practice, since in addition to choosing X_1, X_2 we can also choose an interval $[t_1, t_2]$. At any rate, the size of $\mathcal{H}_{\text{DBH}}$ is quadratic to the size of $\mathbb{X}_{\text{small}}$. At the same time, regardless of the choice of parameters k, l, the hashing cost $\text{HashCost}_{k,l}$ can never exceed the size of $\mathbb{X}_{\text{small}}$, since only elements of $\mathbb{X}_{\text{small}}$ are used to define functions in $\mathcal{H}_{\text{DBH}}$.

In practice, we have found that good results can be obtained with sets $\mathbb{X}_{\text{small}}$ containing as few as 50 or 100 elements. The significance of this is that, in practice, the hashing cost is bounded by a relatively small number. Furthermore, the hashing cost actually becomes increasingly negligible as the database becomes larger and the size of $\mathbb{X}_{\text{small}}$ remains fixed, since the lookup cost starts dominating the total cost of processing a query.

3.6 Applying DBH in Sign Language Recognition Application

If we view signs as objects in some space with a distance measurement, the problem of recognizing a sign is the same as finding nearest neighbors. In this analogy, the distance metric is DTW distance. Recognizing a sign is basically finding closest signs based on DTW distance. The requirement for DBH to work is the ability to compute distance between 2 objects where the distance can be in any arbitrary

space. This is a perfect match for sign language recognition since DTW distance is non-metric.

To implement the system, given training signs, $\mathbb{X} = \{X_1, X_2, X_3, ..., X_n\}$, we pre-compute distance matrix $\mathbf{D}$ where $\mathbf{D}_{ij}$ is DTW distance between X_i and X_j. Having a distance matrix, we proceed on building hash tables as discussed in previous sections, optimize pivot signs selection, line projection selection and k, l parameters optimization. As a results, DBH model is trained.

During sign look-up, given a query sign Q, compute hash value, $H(Q)$, depending on the trained parameters. $D(Q, X)$ is now a DTW distance. Then, look up for candidate objects (signs) through hash tables in filtering step. In refine step, use DTW as distance in the original space to locate true neighbors. We classify the query sign according to the class of its true neighbors.

3.7 Experiments

In the experiments we evaluate DBH by applying it to three different real-world data sets: the isolated digits benchmark (category 1a) of the UNIPEN Train-R01/V07 online handwriting database [40] with dynamic time warping [41] as the distance measure, the MNIST database of handwritten digits [42] with shape context matching [29] as the distance measure, and a database of hand images with the chamfer distance as the distance measure.

3.7.1 Datasets

Here we provide details about each of the datasets used in the experiments. We should specify in advance that, in all datasets and experiments, the set of queries used to measure performance (retrieval accuracy and retrieval efficiency) was completely disjoint from the database and from the set of sample queries used to pick optimal

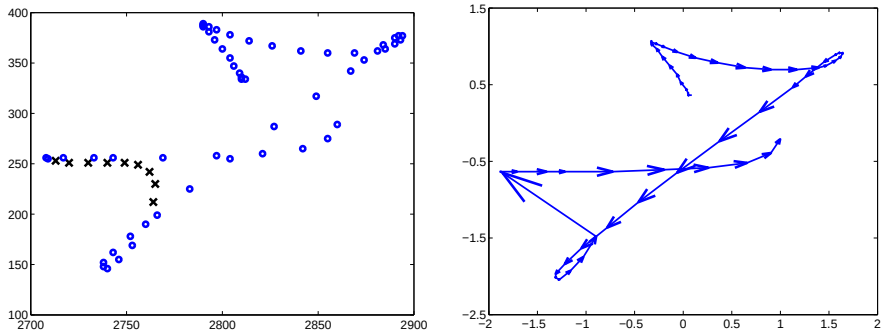

Figure 3.3: Left: Example of a "seven" in the UNIPEN data set. Circles denote "pen-down" locations, x's denote "pen-up" locations. Right: The same example, after preprocessing.

k and l parameters during DBH construction. Specifically, the set of queries used to measure performance was completely disjoint from the sample queries that were used, offline, in Equations 3.13 and 3.16 to estimate $\text{Accuracy}_{k,l}$ and $\text{Cost}_{k,l}$.

The UNIPEN data set. We use the isolated digits benchmark (category 1a) of the UNIPEN Train-R01/V07 online handwriting database [40], which consists of 15,953 digit examples (see Figure 3.3). The digits have been randomly and disjointly divided into training and test sets with a 2:1 ratio (or 10,630:5,323 examples). We use the training set as our database, and the test set as our set of queries. The target application for this dataset is automatic real-time recognition of the digit corresponding to each query. The distance measure D used is dynamic time warping [41]. On an AMD Athlon 2.0GHz processor, we can compute on average 890 DTW distances per second. Therefore, nearest neighbor classification using brute-force search takes about 12 seconds per query. The nearest neighbor error obtained using brute-force search is 2.05%.

The MNIST data set. The well-known MNIST dataset of handwritten digits [42] contains 60,000 training images, which we use as the database, and 10,000 test images, which we use as our set of queries. Each image is a 28x28 image displaying an

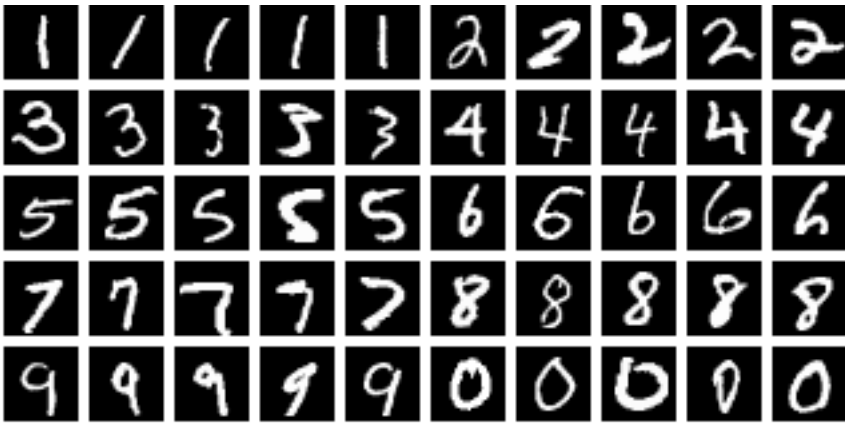

Figure 3.4: Example images from the MNIST dataset of handwritten digits.

isolated digit between 0 and 9. Example images are shown in Figure 3.4. The distance measure that we use in this dataset is shape context matching [29], which involves using the Hungarian algorithm to find optimal one-to-one correspondences between features in the two images. The time complexity of the Hungarian algorithm is cubic to the number of image features. As reported in [43], nearest neighbor classification using shape context matching yields an error rate of 0.54%. As can be seen on the MNIST web site (`http://yann.lecun.com/exdb/mnist/`), shape context matching outperforms in accuracy a large number of other methods that have been applied to the MNIST dataset.

Using our own heavily optimized C++ implementation of shape context matching, and running on an AMD Opteron 2.2GHz processor, we can compute on average 15 shape context distances per second. As a result, using brute force search to find the nearest neighbors of a query takes on average approximately 66 minutes when using the full database of 60,000 images.

The hand image data set. This dataset consists of a database of 80,640 synthetic images of hands, generated using the Poser 5 software [44], and a test set of 710 real images of hands, used as queries. Both the database images and the query

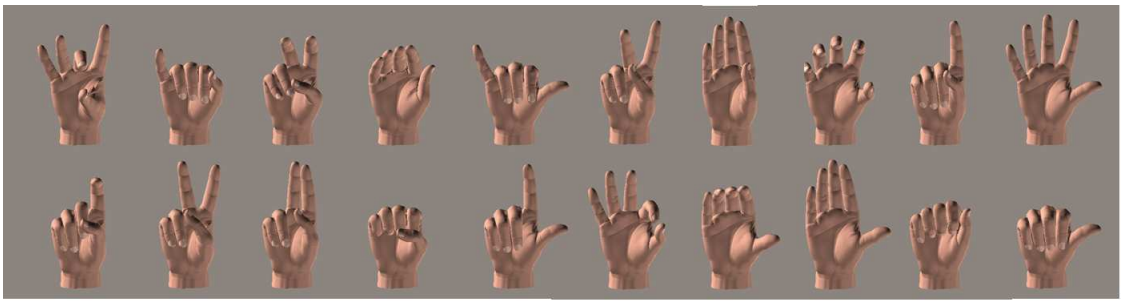

Figure 3.5: The 20 handshapes used in the ASL handshape dataset.

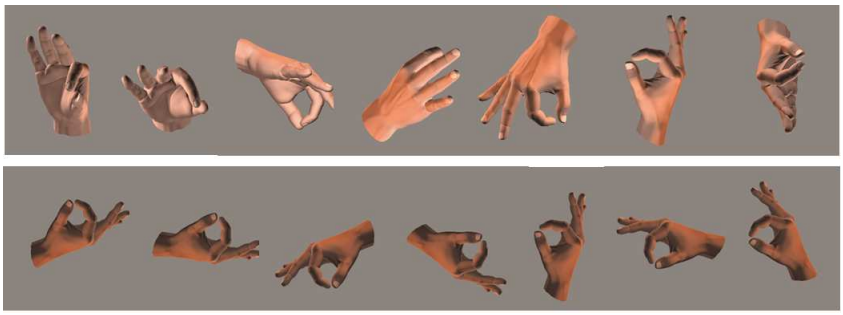

Figure 3.6: Examples of different appearance of a fixed 3D hand shape, corresponding to different 3D orientations of the shape.

images display the hand in one of 20 different 3D handshape configurations. Those configurations are shown in Figure 3.5. For each of the 20 different handshapes, the database contains 4,032 database images that correspond to different 3D orientations of the hand, for a total number of 80,640 images. Figure 3.6 displays example images of a single handshape in different 3D orientations.

The query images are obtained from video sequences of a native ASL signer, and hand locations were extracted from those sequences automatically using the method described in [45]. The distance measure that we use to compare images is the chamfer distance [46]. On an AMD Athlon processor running at 2.0GHz, we can compute on average 715 chamfer distances per second. Consequently, finding the nearest neighbors of each query using brute force search takes about 112 seconds.

3.7.2 Implementation Details

We have 5 versions of our method implemented as follows:

1. Regular DBH where (k, l) parameters are optimized
2. Hierarchical DBH
3. Optimizing pivot objects
4. Optimizing line projections
5. Optimizing pivot objects and line projections

For each data set we constructed a family $\mathcal{H}_{\text{DBH}}$ of binary hash functions as described in Section 3.5.2. Firstly, 100 pivot objects, $\mathbb{X}_{\text{small}}$, are selected. Note that the way we selected $\mathbb{X}_{\text{small}}$ depends on the algorithm. If pivot objects selection optimization is adapted, then the selection will be done as described in section 3.4.2. Then, we select pairs of objects $X_1, X_2 \in \mathbb{X}_{\text{small}}$ as line projections to create binary hash functions by applying Equation 3.5 and choosing randomly an interval $[t_1, t_2] \in \mathbb{V}(X_1, X_2)$. Again, the way we choose pair $X1, X2$ (line projection) is varied on algorithms. If no optimization process is defined, the choice is selected randomly.

If pivot objects optimization is applied, firstly, we selected 2,000 candidate objects randomly before reducing the number to 100 using the algorithm defined in Section 3.4.2. From this point, we have 4,950 line projections available. Similarly, if line projection optimization protocol is adapted. 2,000 line projection candidates are selected randomly from these 4,950 line projections before we optimally select 1,000 best line projections.

As mentioned in section 3.4.1, pre-computed array is needed to speed up the computation. In our implementation, we used two arrays, one is 2D and the other is 3D. The former one is accuracy matrix storing $C(X_i, N(X_i), L_j)$ whilist the latter one is the cost 3D matrix storing $C(X_i, X_j, L_k)$ with constrain that $X_j \notin N(X_i)$. However, due to memory limitation, we cannot put all training pairs in the cost

array. Therefore, for each training objects, we sampled some number of non-neighbor objects. The number currently used in our implemenation is 16 for each object making the cost matrix a size of $10,000 \times 2,000 \times 16$. As mentioned in section 3.4.1, we stored current collision values in two matrices to speedup the computation, where one is for accuracy and the other is for cost storing $C_{\mathbb{L}_t}(X_i, N(X_i))$ and $C_{\mathbb{L}_t}(X_i, X_j); X_j \notin N(X_i)$ respectively.

It is worth mentioning that we also did some minor tweaking to line projection selection algorithm using sampling. For each round, a certain number of training objects and line projections are chosen randomly. only objects in this set are used to compute the accuracy and cost of the current round. Note that the set is re-selected in every round. Therefore, this approach still utilize overall training set.

To estimate retrieval accuracy using Equation 3.13, we used 10,000 database objects as sample queries. To estimate the lookup cost using Equation 3.14 we used the same 10,000 database objects as both sample queries (Q in Equation 3.14) and sample database objects (X in Equation 3.14). The retrieval performance attained by each pair k, l of parameters was estimated by applying Equations 3.13 and 3.16, and thus the optimal k, l was identified for each desired retrieval accuracy rate.

Similarly, to estimate the accuracy and cost of line projections or pivot objects selected so far, same equations are used. However, as mentioned in Section 3.4.1, the collision probability calculation is a bit different as $C(X_1, X_2)$ must be replaced with $C_{\mathbb{L}}(X_1, X_2)$ to reflect quality of chosen set.

We should emphasize that Equations 3.13, 3.14 and 3.16 were only used in the offline stage to choose optimal k, l parameters. The accuracy and efficiency values shown in Figure 3.7 were measured experimentally using previously unseen queries, that were completely disjoint from the samples used to estimate the optimal k, l parameters.

For the hierarchical version of DBH, described in Section 3.5.1, we used $s = 5$ for all data sets, i.e., the hierarchical DBH index structure consisted of five separate DBH indexes, constructed using different choices for k and l.

3.7.3 Results

Figure 3.7 shows the results obtained on the three data sets for regular DBH, Hierarchical DBH and the original result from our previous work [47]. For each data set we plot retrieval time versus retrieval accuracy. Retrieval time is completely dominated by the number of distances we need to measure between the query object and database objects. The number of distances includes both the hashing cost and the lookup cost for each query. To convert the number of distances to actual retrieval time, one simply has to divide the number of distances by 890 distances/sec for UNIPEN, 15 distances/sec for MNIST, and 715 distances/sec for the hands data set. Retrieval accuracy is simply the fraction of query objects for which the true nearest neighbor was returned by the retrieval process.

In Figure 3.7, it can be seen that our regular DBH performed much better than the original work [47] in term of accuracy/cost trade-off. This is because, in the original work [47], t_1 is selected as median of overall projection values. Therefore, the binary hash family, $\mathcal{H}_{\text{DBH}}$, in the original work is more prone to overfitting because of fixed t_1 choice whereas t_1 is chosen randomly in our implementation. Hierarchical DBH generally gets better performance than regular DBH except for MNIST data set where the accuracy/cost ratio is roughly the same as regular DBH. Finally, DBH outperformed VP-Tree for all data sets.

Next, figure 3.8 compared accuracy/cost ratio of regular DBH and optimization methods. As seen in the figure, generally, the combination of line projection optimization and pivot objects optimization gives overall the best trade-offs between

efficiency and accuracy which make sense as the protocol utilize two optimization methods. Pivot objects optimization comes second with slightly better performance than line projection optimization protocol except in MNIST data set where both protocols give roughly the same performance. In sum, we concluded that the two new methods give better performance than regular DBH.

In conclusion, two new optimization methods, line projection and pivot objects selection, are proved to beat general DBH in term of trade-offs between retrieval accuracy and efficiency. In addition, we can further enhance the performance by using these two methods together. Again, we emphasize that all three data sets use non-metric distance measures, and no known method exists for applying LSH on those data sets

3.8 Conclusions and Future Works

We have presented DBH, a novel method for approximate nearest neighbor retrieval in arbitrary spaces. DBH is a hashing method, that creates multiple hash tables into which database objects and query objects are mapped. A key feature of DBH is that the formulation is applicable to arbitrary spaces and distance measures. DBH is inspired by LSH, and a primary goal in developing DBH has been to create a method that allows some of the key concepts and benefits of LSH to be applied in arbitrary spaces.

The key difference between DBH and LSH is that LSH can only be applied to spaces where locality sensitive families of hashing functions have been demonstrated to exist; in contrast, DBH uses a family of binary hashing functions that is distance-based, and thus can be constructed in any space. As DBH indexing performance cannot be analyzed using geometric properties, performance analysis and optimization is based on statistics collected from sample data. In experiments with three real-world,

non-metric data sets, DBH has yielded good trade-offs between retrieval accuracy and retrieval efficiency, and DBH has significantly outperformed VP-trees in all three data sets. Furthermore, no known method exists for applying LSH on those data sets, and this fact demonstrates the need for a distance-based hashing scheme that DBH addresses.

Future works on DBH includes experimenting on American Sign Language dataset. The goal of the experiment is to measure the speed up factor versus accuracy lost between brute force approach and applying DBH indexing. The experiment will be conducted using DTW as distance metric and in user-independent scenario such that test signer will never appear on training samples.

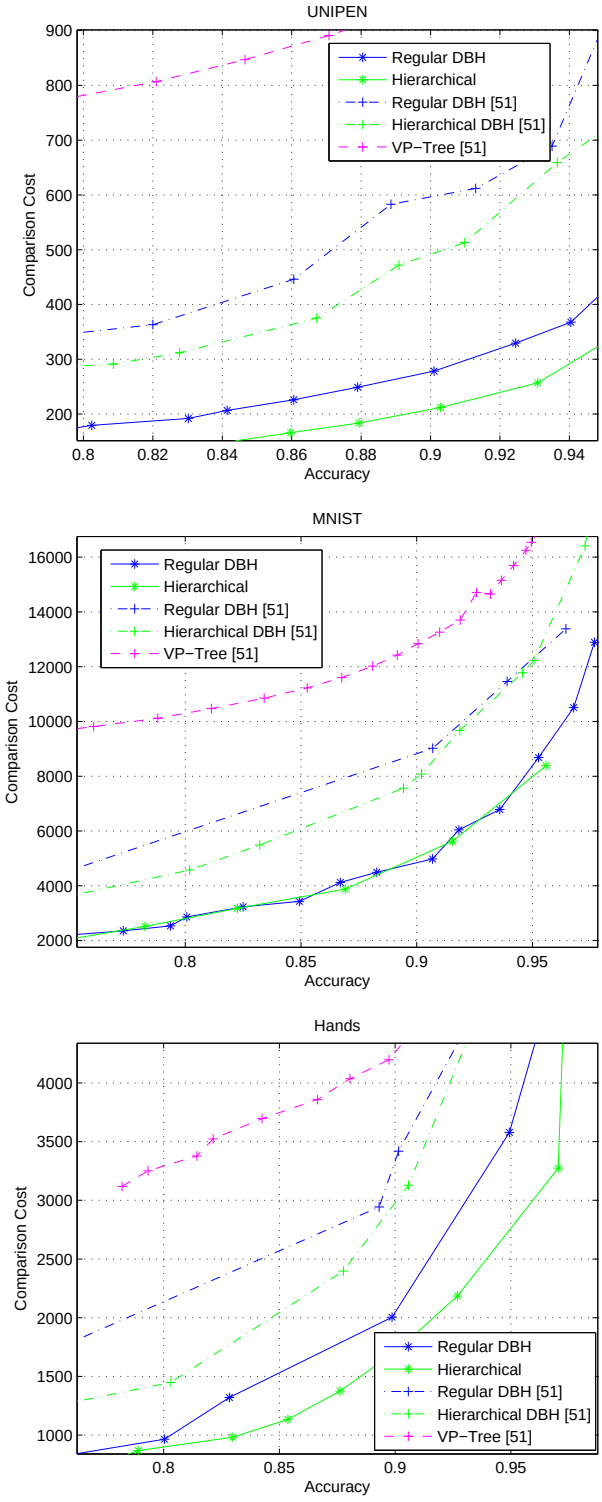

Figure 3.7: Results on three data sets comparing regular DBH, Hierarchical DBH and original result from previous work [47].

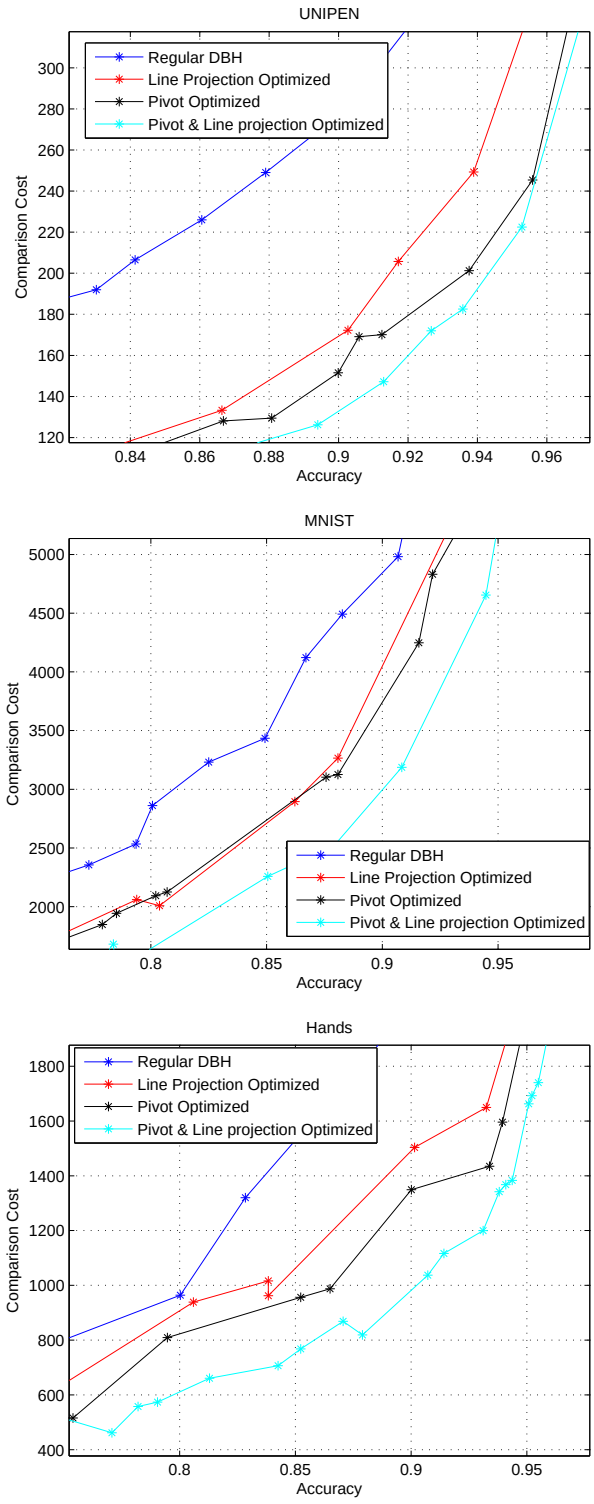

Figure 3.8: Results on 3 data sets comparing regular DBH with newly developed methods . The x-axis is retrieval accuracy. The y-axis is the average number of distances that need to be measured per query object.

CHAPTER 4

Preliminary Results - Model Based Search

4.1 Sign Recognition Indexing

Recognizing a sign is usually involving an expensive computational of distance/ similarity or classification process between trained model and query sign video. The expensive computation problem is even more pronounced during testing time, when, given a query and try to look for a similar sign in a large database of signs using one-vs-all approach. As such, indexing is one of the major prominent research area in computer vision application and data mining in general.

4.2 Model based Search

Large databases of patterns, such as hand shapes, faces, body poses, fingerprints, or gestures, are becoming increasingly widespread, thanks to advances in computer technology. Here, we focus on the problem of efficient search in such databases, when using model-based search. In model-based search, the user submits as a query a classifier, that has been trained to recognize the type of patterns that the user wants to retrieve. While model-based search can lead to good retrieval accuracy, the efficiency of model-based search can be inadequate if we need to apply the query classifier to every single database pattern. We implemented a method for improving the efficiency of model-based search. The method assumes that classifiers have been trained using JointBoost, and operates by defining an embedding, which maps both classifiers and database patterns into a common vector space. Using this embedding, the problem of finding the database patterns maximizing the response of the query

classifier is reduced to a nearest neighbor search problem in a vector space. This reduction allows the use of standard vector indexing method to speed up the search. In our experiments, we show that the proposed embedding, together with a simple PCA-based indexing scheme, significantly improve the efficiency of model-based search, as measured on a database of face images constructed from the public FRGC-2 dataset.

4.3 Motivation

Current technology has made it quite feasible to create large databases of patterns, such as faces [48], body poses [31, 1], fingerprints, or gestures [49]. Identifying relevant content in such databases can have a variety of real-world uses, such as identifying photographs of a missing person in surveillance camera recordings, or locating occurrences of a specific sign (or sequence of signs) in a database of sign language videos. One common way of searching for content of interest is search-by-example, also known as similarity-based search: the user submits as a query an example of the type of patterns they want to retrieve (e.g., a photograph of the person for which they want to locate additional photographs). Another way to do search is search-by-model (for which we also use the term model-based search), where the user submits as a query a classifier trained to recognize the specific patterns of interest (e.g., a classifier trained to recognize the person whose photographs the user wants to retrieve).

For search-by-example, a large number of similarity-based indexing methods have been proposed to make this type of search more efficient, and capable of scaling to large databases. However, for search-by-model, we are not aware of any method designed to speed up brute-force search, which involves applying the classifier submitted by the user to all database images. In this chapter, we implement such a method, for the specific case where classifiers are constructed using JointBoost [50], which is a variant of AdaBoost [51]. As shown in [50], JointBoost can improve classifica-

tion accuracy, compared to standard boosting methods, in multiclass problems with relatively few training examples per class, by forcing classifiers trained for different classes to share training information.

In such a large database of face images, a key operation is to retrieve images of a specific individual. In principle, a classifier can be constructed on the fly, by providing some training photographs of the person of interest and applying a standard machine learning method, such as boosting or support vector machines [52, 51, 53]. The simplest way to do model-based search would involve applying this classifier to every single face image in the database. This brute-force approach takes time linear to the size of the database and would have difficulty scaling to very large databases. For example, databases of face images compiled by a web search engine or a large-scale surveillance camera network can easily reach sizes of millions or billions of face images.

Support for large database sizes can be provided if, instead of search-by-model we use search-by-example, and submit, as a query, a single mugshot of the person of interest. In that case, hash-based indexing methods such as Locality Sensitive Hashing (LSH) [54] can be applied and have provably sublinear time complexity. However, we argue that model-based search has the potential, at least in some applications, to be far more accurate than similarity-based search, as it is quite plausible that a query classifier, trained to recognize objects of a specific class, contains more information than a single query pattern from that class. Thus, we believe it is of interest to study the problem of designing efficient alternatives to brute force for the search-by-model paradigm, and we propose such an efficient alternative in this paper.

The method uses an embedding formulation proposed in [55], which maps both database patterns X and JointBoost classifiers H to a common vector space. In [55], such an embedding was used for efficient classification of a single pattern X in a domain with a large number of classes. In our case, we have a different problem than

in [55]. Instead of doing multiclass recognition of a single pattern X, meaning that we look for the classifier H_y that maximizes $H_y(X)$ for a specific X, we do model-based search, meaning that we look for patterns X maximizing $H_y(X)$ for a specific classifier H_y. We adapt the basic ideas from [55] to define an embedding that is appropriate for this problem. The embedding we propose reduces model-based search to nearest-neighbor search in a vector space. This reduction allows us to use tools from the arsenal of similarity-based indexing methods to speed up model-based search. More specifically, in this paper we combine the proposed embedding with a simple vector indexing scheme, based on PCA, to improve the efficiency of model-based search.

In our experiments we use a database of 19,965 face images, from the public FRGC-2 dataset [48]. Compared to brute-force model-based search, the proposed method obtains speed-ups of over an order of magnitude, with relatively small losses in retrieval accuracy. While our case study is limited to a database of faces, the proposed formulation is general, and can be applied to speed up search-by-model in databases of different types of patterns.

4.4 Using JointBoost in Model-Based Search

Let $\mathbb{X}$ be a space of patterns, and $\mathbb{Y}$ be a finite set of class labels. Every pattern $X \in \mathbb{X}$ has a class label $L(X) \in \mathbb{Y}$. In JointBoost [50], for each class $y \in \mathbb{Y}$ a boosted classifier $H_y : \mathbb{X} \to \mathbb{R}$ is trained to discriminate between patterns of class y and all other patterns. Classifier H_y is of the following form:

$$H_y = \sum_{m=1}^{d} \alpha_{y,m} h_m + k_y \ , \tag{4.1}$$

where each h_m is a weak classifier with weight $\alpha_{y,m}$, and k_y is a class-specific constant that gives a way to encode a prior bias for each class y [50].

The key difference of JointBoost from other boosting methods is that all classifiers H_y share the same weak classifiers h_m. The only thing differentiating the different classifiers is the weights $\alpha_{y,m}$ assigned to each weak classifier h_m. Forcing strong classifiers to share the same weak classifiers was shown in [50] to improve classification accuracy when limited training data are available for each class. The intuition behind this behavior is that, in JointBoost, weak classifiers are chosen to jointly optimize performance in multiple one-vs.-all classifiers, and thus the choice of weak classifiers is supported by more data. In contrast, in a standard boosting approach, each one-vs.-all classifier H_y would be trained in isolation. When training each H_y in isolation, while weak classifiers chosen for each H_y could potentially be better suited for recognizing class y (thus increasing accuracy), limited training data for class y may lead to overfitting (thus decreasing accuracy).

Higher (more positive) responses $H_y(X)$ indicate higher confidence that the true class label $L(X)$ of pattern X is y. In the typical JointBoost application, which is multiclass recognition, to classify a pattern $X \in \mathbb{X}$, we evaluate $H_y(X)$ for all $y \in \mathbb{Y}$, and classify X as belonging to the class y for which $H_y(X)$ is maximized.

In our problem, which is model-based search, JointBoost can be used to train classifiers recognizing specific classes of patterns, such as specific persons if we are searching a database of face images. The user provides as a query a classifier H_y, trained to recognize objects of class y. In the simplest (but inefficient) approach, the system applies H_y to every single database pattern X, and ranks patterns in decreasing order of the response $H_y(X)$. The user then can inspect the top K results, where K is a number determined either by the user or by the system (the system can have a threshold such that only responses above that threshold are shown to the user).

The classical formulation of JointBoost assumes that all classifiers are trained at the same time. This may be problematic in search-by-model applications, where we may not know in advance all possible classes that the user may search for. However, it is easy to adapt JointBoost to a situation where some classifier H_y is trained later, after the main training has occurred. In that case, the pool of weak classifiers h_m can remain the same pool that was chosen in the main training. To train the new classifier, the system can simply compute optimized weights $\alpha_{y,m}$ for that new classifier. This allows new classifiers to be built as needed.

In principle, for our application, a non-technical user can simply provide as a query a set of training examples for class y. The system, in a manner transparent to the user, can then train a classifier H_y on the fly, using those training examples as positives, and a large pool of other examples as negatives. Then, the system can submit H_y as the query classifier, on behalf of the user.

4.5 A Joint Embedding of JointBoost Classifiers and Patterns

In this section we propose an embedding V that maps both database patterns and JointBoost classifiers to a common vector space, and more specifically to points on the surface of a hypersphere. Using this mapping, the search for the database patterns X that maximize the response $H_y(X)$ of the query classifier H_y is reduced to the problem of finding the nearest neighbors of $V(H_y)$ among the embeddings $V(X)$ of all database patterns. The embedding is an adaptation of (but not identical to) the embedding proposed in [55], and in the following description we borrow heavily from [55].

In particular, we will map both JointBoost classifiers and database patterns into a $(d+2)$-dimensional vector space, where d is the number of weak classifiers that are used to define the JointBoost classifiers. We denote by $V(X)$ and $V(H_y)$ respectively

the vectors corresponding to database pattern X and JointBoost classifier H_y. In defining this mapping, we will explicitly ensure that all resulting vectors have the same norm.

We begin by defining the vector $V(X)$ corresponding to each pattern X in space $\mathbb{X}$:

$$V(X) = (h_1(X), \ldots, h_d(X), 1, c_X) \ , \tag{4.2}$$

where h_m are the weak classifiers used in Equation 4.1, and c_X is a value calculated for each X, that ensures that all $V(X)$ have the same Euclidean norm.

Quantity c_X can be determined as follows: first, we need to identify what the maximum norm of any $V(X)$ would be if we set all c_X to zero, for all patterns X in our database $\mathbb{U}$:

$$N_{\max} = \sqrt{\max_{X \in \mathbb{U}}[(\sum_{m=1}^{d} h_m(X)^2) + 1]} \ . \tag{4.3}$$

Then, we define c_X as:

$$c_X = \sqrt{N_{\max}^2 - [(\sum_{m=1}^{d} h_m(X)^2) + 1]} \ . \tag{4.4}$$

By defining c_X this way, it can easily be verified that the Euclidean norm of every $V(X)$ is equal to $N_{\max}$.

Now we can define the vectors corresponding to JointBoost classifiers H_y. In particular, given a classifier H_y, we define an auxiliary vector $V_{\text{orig}}(H_y)$, and the vector of interest $V(H_y)$, as follows:

$$V_{\text{orig}}(H_y) = (\alpha_{y,1}, \ldots, \alpha_{y,d}, k_y, 0) \tag{4.5}$$

$$V(H_y) = \frac{N_{\max} V_{\text{orig}}(H_y)}{\|V_{\text{orig}}(H_y)\|} \tag{4.6}$$

In the above equations, $\alpha_{y,m}$ and k_y are the weights and class-bias terms used in Equation 4.1, and $\|V\|$ denotes the Euclidean norm of V.

Using these definitions, it is easy to verify that for any classifier H_y and pattern X it holds that:

$$H_y(X) = V_{\text{orig}}(H_y) \cdot V(X), \tag{4.7}$$

where $V_1 \cdot V_2$ denotes the dot product between vectors V_1 and V_2. We note that the $(d+2)$-th coordinate of $V(X)$, which is set to c_X, does not influence $V_{\text{orig}}(H_y) \cdot V(X)$, since the $(d+2)$-th coordinate of each $V_{\text{orig}}(H_y)$ is set to zero.

In model-based search, given a query classifier H_y, the system needs to return to the user the top K database patterns X that maximize $H_y(X)$ (for some given value of K). Equation 4.7 shows that finding the patterns X maximizing $H_y(X)$ is the same as finding the patterns X maximizing $V_{\text{orig}}(H_y) \cdot V(X)$. It readily follows that maximizing $V_{\text{orig}}(H_y) \cdot V(X)$ is the same as maximizing $V(H_y) \cdot V(X)$, since the dot products of $V(H_y)$ with each $V(X)$ are simply scaled versions of $V_{\text{orig}}(H_y) \cdot V(X)$, where the scaling value $N_{\max}/\|V_{\text{orig}}\|$ does not depend on X.

We will now take one additional step, to show that maximizing the dot product between $V(H_y)$ and $V(X)$ is the same as minimizing the Euclidean distance between $V(H_y)$ and $V(X)$. That can be easily shown, by using the fact that both $V(X)$ and $V(H_y)$ are vectors of norm $N_{\max}$, because the dot product and the Euclidean distance for vectors of norm $N_{\max}$ are related as follows:

$$\|V(X) - V(H_y)\|^2 = 2N_{\max}^2 - 2(V(X) \cdot V(H_y)) \ . \tag{4.8}$$

As shown in [55], the above equation can be easily derived as follows:

$$\begin{aligned} \|V(X) - V(H_y)\|^2 = & \quad (4.9) \\ = \; & (V(X) - V(H_y)) \cdot (V(X) - V(H_y)) \quad (4.10) \\ = \; & (V(X) \cdot V(X)) + (V(H_y) \cdot V(H_y)) \\ & -2(V(X) \cdot V(H_y)) \quad (4.11) \\ = \; & 2N_{\max}^2 - 2(V(X) \cdot V(H_y)) \; , \quad (4.12) \end{aligned}$$

using the fact that $V(X) \cdot V(X) = V(H_y) \cdot V(H_y) = N_{\max}^2$.

This result means that, given a query classifier H_y, finding the top K database patterns X that maximize $H_y(X)$ is reduced to finding the K nearest neighbor of $V(H_y)$ among all vectors $V(X)$ corresponding to database patterns X. The next section describes how to use that fact for speeding up model-based search.

The main difference of the embedding definition in this section from [55] stems from the fact that, in our problem, our goal is to find the patterns X maximizing the response $H_y(X)$ of a given classifier $H_y(X)$, as opposed to finding, in [55], the classifier H_y (among many classifiers) maximizing $H_y(X)$ for a given X. Due to the different goal in this paper, we have inserted value c_X as the value in the last dimension of $V(X)$ in Equation 4.2, and we have used value 0 for the last dimension of $V_{\text{orig}}(H_y)$, whereas the values used for those last dimensions are different (and, loosely speaking, switched) in [55].

4.6 Using the Embedding for Efficient Model-Based Search

So far we have established that, given a query classifier H_y, to find the top K database patterns X maximizing $H_y(X)$, it suffices to find the K nearest neighbors of $V(H_y)$ among all vectors $V(X)$ obtained from database patterns X. The importance

of this reduction is that it allows use of any of several vector indexing methods to speed up the search, such as, e.g., the methods in [56, 54, 57].

In our experiments, we have been able to obtain significant speed-ups via a simple approach based on principal component analysis (PCA) [58]. Since the set of vectors $V(X)$ is computed off-line, we can use those vectors for an additional off-line step, where PCA is used to identify the principal components of those vectors and the corresponding projection matrix Φ. Given a query classifier H_y, its vector $V(H_y)$ can be projected to $\Phi(V(H_y))$ online, and then $\Phi(V(H_y))$ can be compared to the projections $\Phi(V(X))$ of the vectors corresponding to database patterns X.

PCA can easily be used within a filter-and-refine retrieval framework [59], as follows:

- **Input:** A query classifier H_y, and its vector representation $V(H_y)$.
- **Filter step:** Compute the projection $\Phi(V(H_y))$ to the lower-dimensional space, and rank database objects X in increasing order of the distance between $\Phi(V(X))$ and $\Phi(V(H_y))$.
- **(optional) Refine step:** Rerank the p highest ranked patterns X (where p is a system parameter), in decreasing order of $H_y(X)$. Beyond the top p patterns, the rest of the ranking computed by the filter step is not changed. In our experiments, $p = 0$, and thus no refine step is performed.
- **Output:** Return the K highest ranking patterns to the user. The number K of results to be returned is not something that we address in this paper, this number can be determined by the user or by the system.

As long as $d' \ll d$ (where d' is the number of dimensions of $\Phi(V(X))$, and d is the number of weak classifiers), the filter step is significantly faster than simply applying H_y to all database patterns X. At the refine step (if we opt to perform that

step) we do apply H_y on some patterns X, but, typically p is much smaller than the number of all patterns in the database.

In our experiments, we set $p = 0$, meaning that we did not use a refine step at all, as we obtained sufficiently good results using just the rankings from the filter step.

4.7 Applying Model Based Search on Sign Language Recognition

Generally, in sign language recognition system, it starts with having a user perform a sign and submit it as a query to the system asking for the most similar sign to find the closest class label. This is known as search by example as a traditional way of searching. To use model based search with sign recognition, users must be willing to supply the system with multiple query samples. This is a requirement as model based search use JointBoost classifier as query.

Since model based search requires JointBoost classifier to be able to operate, we can no longer use time series model such as Hidden Markov Model, Conditional Random Field as classifiers. To apply JointBoost on time series data, we need to extract features from a sign video as a single fixed length features vector rather than variable length time series features. To this end, firstly, we extract time series features as variable length number as usual. This could be done by finding hands and extract hands regional features for each frame. Then, we use bag-of-features to obtain one single fixed length vector. The idea of bag-of-features comes from text processing community. Typically, a document is represented as counting of each word. Representing a dictionary of words with variable, D, the extracted document features vector has $|D|$ dimensions. Each feature is the counting of each word presenting in the document. However, the information is too sparse and might be noisy. As such, we would want to transform word features into topic features. In other words, we

cluster words into topics and transform from words counting to topics counting. This will results in more dense, compact, core capturing information features vector. The intuition of bag-of-words is the same as Principle Components Analysis (PCA) and histogram, that is, to capture core information and remove noise.

The analogy of bag-of-words to sign videos is, a video is a document, each frame feature is a word. We would like to cluster these frame features into topics and the final results is a fixed length vector where each feature is a count of each topic (cluster) appearance. Using bag-of-features representation is very popular in computer vision topic including object recognition where each word is SIFT descriptor. The downside of bag-of-features is it has lost spatio-temporal information.

Once we extract bag-of-features vectors from given query sign videos, we can start training JointBoost query classifier. However, JointBoost requires joint training among all classes. Apparently, training all classes classifiers takes too much time during look up process. The practical solution to this regard, is pre-computing all weak classifier, h, from signs database. Having all weak classifiers at hands, the training process of query classifier become AdaBoost training. For negative samples, we can randomly select some signs from database as negative samples.

The obvious problem of implementing Model based Search is user cooperation and query classifier training time. If either of these 2 problems cannot be addressed, the more straightforward solution is using the original work of [55]. In this case, we pre-compute JointBoost classifiers for every sign in the database using the method mentioned previously. During look up time, we extract a bag-of-features query vector from a given query sign video. Then, we use this feature vector to look up for the nearest classifier as described in [55].

4.8 Experiments

4.8.1 Dataset

As a case study, we conducted experiments on a large database of face images, that we constructed using the FRGC-2 public face dataset [48]. The dataset consists of 36,817 face images from 535 classes (i.e., 535 distinct persons). The image resolution that we used is 100×100. The actual database that we used contained 19,965 images. The remaining 16,852 images were used as training set, to train the JointBoost classifiers,

2,130 images from the training set were also used as a test set for the similarity-based search method described in Section 4.8.4, that we used as one of the baseline methods. We should note that, for that method, no training was needed. We ensured that each set (training, database, test) contains at least one sample from every class. Beyond that constraint, images were sampled randomly. Images were cropped to get only the facial region, and normalized to mean 0 and standard deviation 1 to remove influences of brightness and contrast.

4.8.2 JointBoost Implementation

We applied PCA [58] to the 16,852 training images, and we kept the top 200 components as input for JointBoost training. To avoid confusion, this PCA operation is NOT related to the PCA operation discussed in Section 4.6. Rather, this PCA operation is simply a feature extraction preprocessing step, before we apply JointBoost. In training JointBoost classifiers, the system formed weak classifiers h_m simply by choosing, for each h_m, a PCA dimension (out of the 200 dimensions we kept) and a threshold value. So, essentially each weak classifier was a decision stump.

JointBoost selected a total of 2,345 unique weak classifiers (thus, $d = 2,345$ in Equation 4.1).

4.8.3 Indexing Implementation

As described in Section 4.5, the number of dimensions of the vectors that we map classifiers and patterns to is $d + 2$, where d is the number of weak classifiers in Equation 4.1. Therefore, the total number of dimensions is 2,347.

As we discuss in Section 4.6, we use PCA as the basis of our indexing method. The PCA projection matrix was trained from the embeddings $V(X)$ obtained from all 19,965 database images. We used the indexing method described in Section 4.6, without a refine step. Thus, the only parameter we needed to set was the number d' of PCA dimensions to use for the filter step. We show results with d' values 100, 200, and 500.

4.8.4 Baseline Methods

In our experiments, we compare (based on accuracy and efficiency) the proposed method with the following baseline methods:

- **Brute-force model-based search.** Here, we simply apply classifier H_y to every single database pattern X. The goal of this paper has been to propose a significantly more efficient alternative for this baseline method, without sacrificing too much in accuracy. Thus, it is important to examine the accuracy and efficiency trade-offs that our method achieves compared to brute force.
- **Brute-force similarity-based search.** Since similarity-based search is a common alternative to model-based search, we evaluated the accuracy of a similarity-based search method that used the Euclidean distance to compare a query image to database images. For this baseline method, we used as queries the 2,130 images that we designated as test set (and which were not part of the database).

- **Truncated JointBoost classifiers.** If we want to speed up model-based search using JointBoost classifiers (or any other boosted classifiers, for that matter), a very simple approach is to simply choose fewer weak classifiers (thus, use a smaller d in Equation 4.1). This is what we use in this baseline method. This method trades accuracy for efficiency (larger d means higher accuracy and longer running time), and its speed is quite similar to our method, as long as the number of weak classifiers used in this baseline method is equal to the number of PCA dimensions that we use in our method.

We should note that, for our method, for brute-force model-based search, and for truncated JointBoost classifiers, as queries we used the 535 classifiers trained by JointBoost to recognize each of the 535 classes in the FRGC-2 dataset.

4.8.5 Measuring Precision and Recall

We use precision and recall as measures of accuracy. For a given number K of results presented to the user for a query, precision is the percentage out of the top K results that are actually correct, and recall is the fraction of correct results ranked as top K over the total number of correct results.

To compute precision and recall for a specific query, we first need to determine K, i.e., how many results to return for that query. We use the simple approach of using the same K for all queries. For each specific method, after fixing K, we computed the precision value and the recall value obtained for each query. We averaged those values over all queries, to obtain the precision value and the recall value for the entire set of queries, for that value of K. By varying K, we obviously obtain different precision/recall values for each method, ranging from recall 0 to recall 1. We plot these values to generate a traditional precision-vs.-recall curve, as shown on Figure 4.1.

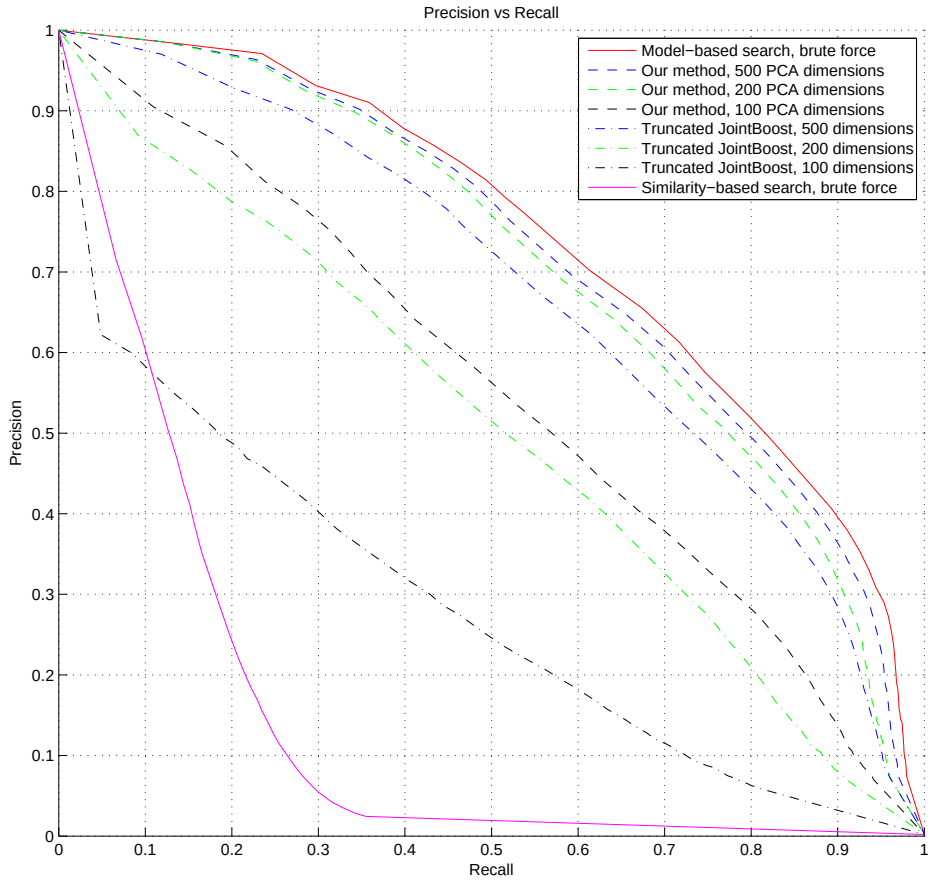

Figure 4.1: The precision vs. recall curves for our method with different numbers of PCA dimensions used, as well as for brute-force model-based search, brute-force similarity-based search, and three truncated JointBoost classifiers.

4.8.6 Results

Figure 4.1 shows precision versus recall plots for our method (with 100, 200, and 500 PCA dimensions used) as well as for the baseline methods. Table 4.1 shows the speed-up obtained by different methods, compared to brute-force model-based search (which, by definition, has a speed-up of 1).

As shown on Figure 4.1, model-based search clearly outperforms the similarity-based search alternative in terms of accuracy. For example, for a recall rate of 0.4,

Table 4.1: JointBoost speed-up factor

Method	Speed up factor
Model-based search, brute force	1
Our method, 500 PCA dimensions	6.317
Our method, 200 PCA dimensions	17.573
Our method, 100 PCA dimensions	31.491
Truncated JointBoost, 500 dimensions	6.415
Truncated JointBoost, 200 dimensions	18.458
Truncated JointBoost, 100 dimensions	31.690
Similarity-based search, brute-force	11.785

similarity-based search obtains a precision under 0.05, whereas model-based search obtains a precision over 0.85. This result shows that, in this dataset, model-based search using a generic machine learning method (JointBoost) does much better than similarity-based search using a generic similarity measure (Euclidean Distance). This result motivates the need for methods, such as the method proposed in this paper, to speed up model-based search. We should note that better results may be obtainable in similarity-based search as well, using more sophisticated similarity measures, e.g., [60, 61]. Still, we considered it important to establish that our model-based search implementation, at the very least, is much more accurate than a simple implementation of similarity-based search. We should also note that similarity-based search could probably be made even more efficient by using some indexing method, but given its very low accuracy we did not explore that direction.

Figure 4.1 and Table 4.1 also show that our method obtains significant speed-ups over brute-force search with rather small losses in accuracy. For example, our method when using 200 PCA dimensions obtains a precision vs. recall curve rather close to that of brute-force search, while attaining a speed-up of a factor of 17.6, which is more than an order of magnitude. We note that, when using only 100 PCA dimensions, the accuracy of our method deteriorates noticeably.

Finally, we observe in Figure 4.1 that our method obtains much better accuracy-vs.-efficiency trade-offs compared to using truncated JointBoost classifiers. In terms of speed-up factors, our method with d' PCA dimensions obtains, as expected, roughly the same speed-up as truncated JointBoost classifiers using only d' weak classifiers, as shown on Table 4.1. However, for the same d' of 100, 200, or 500, our method obtains significantly better precision-vs-recall curves. This result further highlights the benefits of the proposed method, by showing that our method works better than the simple ad hoc solution of improving efficiency using fewer weak classifiers.

In summary, model-based search was far more accurate than similarity-based search, which was, on the other hand, far more efficient. This result motivates the need for methods, such as our own, to improve the efficiency of model-based search. Our method, using the proposed embedding and 200 PCA dimensions, obtained accuracy rather similar to that of brute-force model-based search, with a speed-up factor of 17.6.

4.9 Conclusions and Future Works

We have proposed a novel indexing method for speeding up model-based search in databases of patterns. We have motivated our approach by showing that, in our case study, model-based search obtained much better accuracy than a simple similarity-based search implementation. Our proposed indexing method is based on an embedding that maps both classifiers and database patterns into a common vector space. Using that embedding, the task of finding the database patterns that maximize the response of the query classifier is reduced to finding nearest neighbors in a vector space. This reduction allows various general-purpose vector indexing methods to be used to speed up the search.

In our experiments on the public FRGC-2 dataset of face images, we have shown that the proposed embedding, combined with a rather simple PCA-based indexing scheme, provides significant speed-ups with only small losses in accuracy, compared to brute-force model-based search. In future work, we plan to explore more sophisticated vector indexing methods, to measure the extent to which they can further improve efficiency. Also, as the proposed method specifically targets cases where classifiers are trained via JointBoost, we are interested in exploring the problem of designing indexing methods for more general types of classifiers, including, for example, support vector machines.

For future works, we have a plan to apply the idea of [55] to sign language recognition. Explicitly, we will conduct an experiment on ASL dataset, where the implementation will be as explained in section 4.7. The reason we do not implement model based search is because of the limitation number of samples per class. The result will be focusing on speed up factor versus accuracy lost. Again, all experiment will be done in user independent scenarios where test signers will never appear on training set.

CHAPTER 5

Hands Detection

In this chapter, we discussed the related works on hands detection, ideas, details and implementation of the proposed hands detection method.

5.1 Related Work

The most basic idea on hands detection is using template matching method such as the famous Adaboost detector to detect hands shape directly. However, this approach does not satisfy the desired detection accuracy since hands shape can be in any arbitrary articulated shapes as shown in figure 5.1 making it impossible to capture all shapes using a single or fixed multiple number of templates.

Another approach is pixel based approach, where the goal is to label each pixel into body parts label including hands. Hands region are usually selected by choosing area where hands pixels are condensed. Generally, this is done by using Gaussian filter with variation of number of standard deviation values to represent different hands sizes. The most basic form of this approach is using skin and motion values [1]. Each pixel is given score as a linear combination of skin and motion score. To detect hands region, a fixed size of Gaussian filter is applied on the entire image to determine the region score. Region with highest score is selected as hands. While this approach works well with sign language videos due to the fact that the sign language always involve hands movement, it is failed in the case of static gesture where hands tend to remain idle. In addition, the approach has been criticized of relying too much on motion and unable to cope with noisy background.

Figure 5.1: Articulated hands shapes in signs. Detecting hands with template matching models will fail due to various articulated possible shapes.

One of the approaches that is currently very popular in computer vision research is using contextual information. The intuition is that, beside direct information such as hand shape or motion, contextual environment can also be used to infer the result as well. The most famous contextual model is a graphical model based on star structure made famous by the work of Felzenszwalb et al [62]. Generally, graphical models typically use some similarity/ distance functions to measure the compatibility between the local model and the observed region features. Then, these local models are connected together using a spring-like connector function. Graphic models are generally used for label the whole body parts including head, torso, upper arms, lower arms and hands, and thus, can be used as hands detector as well. There has been many works base on graphical models [62, 63, 64, 65, 66] on different kind of applications. However, the most well known ones are for body part labeling [62, 63, 64] and for direct template matching [65, 66].

Recently, Shotton et al [1] has proposed a pixel based classifier based on random forest model taking advantage of depth information to label body parts. The idea starts from selecting a pixel features which in this case, is depth intensity. Then select any random offset in 2D space. The tree features is a difference between selected offset pixel in pixel features. More explanation and background on this method is provided in the method section. The results show good accuracy on depth images and fast inference time due to the fact that it can be implemented on GPU. Unfortunately, the approach only applied to depth data. There has been many follow up works [67, 68, 69, 70, 71, 72] such as object recognition [68, 69]. Here, we are most interested with body parts labeling application. [73] proposed a body parts labeling on RGB images based on random forest model. Firstly, the signer is segmented from the background using Grabcut algorithm [74] and various methods. From these segmented region, the foreground and background distribution are built. They used foreground and background probability as pixel features. Finally, these pixel features are then applied on 2D random offset as an input to random decision forest. The main challenge in the approach is dynamic background as it is the television footage making it hard to use motion based features.

In this work, I propose a random forest hands detector based on [1]. The difference between our approach and [1] is

1. My approach will work on RGB videos whereas the original approach only works on depth images.
2. In the original work of Shotton et al [1], since the input is depth images, the authors only utilize random offsets in 2D space. In our case, the input is RGB videos, therefore, the random offsets is now in 3D space where the 3rd dimension is time frame utilizing all information.

5.2 Background

In this section, the background knowledge of the proposed method is explain thoroughly. In addition, Shotton et al [1] paper which is the inspiration of our method is also explained here.

5.2.1 Decision Tree

Decision tree is one of the most popular classification model in machine learning and data mining community. Given a data point $\mathbf{x} \in \mathbb{X}$, to classify $\mathbf{x}$, at each node in a decision tree, a binary decision is made whether to send $\mathbf{x}$ to left child or right child. The decision is usually a simple weak classifier such as threshold based decision. The process repeats until $\mathbf{x}$ reaches a leaf node. The leaf nodes of a tree containing a class label which is the final result of the classification.

Formally, given a data point $\mathbf{x}$, at node α, there is a binary decision function, $F_\alpha(\mathbf{x})$. $\mathbf{x}$ is sent to the following as described in conditions below

1. Left child of α if $F_\alpha(\mathbf{x}) = 0$
2. Right child of α if $F_\alpha(\mathbf{x}) = 1$

If a node, α is a leaf node, then it will also contain a class label $\mathbf{z}_\alpha \in \mathbb{Z}$ where $\mathbb{Z} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_m\}$ is class labels. If $\mathbf{x}$ reaches a leaf node α, then the predicted class label of $\mathbf{x}$, $z(\mathbf{x}) = \mathbf{z}_\phi$.

To train a tree, we have to figure out a decision function F_α. Given a training set, $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ...\mathbf{x}_n\}$ and a set of predefined classifiers $\{\theta_1, \theta_2, ..., \theta_k\}$, the goal here is to select a classifier θ^* such that it satifsy the optimization function.

The optimization function in decision tree is usually an information gain function given as

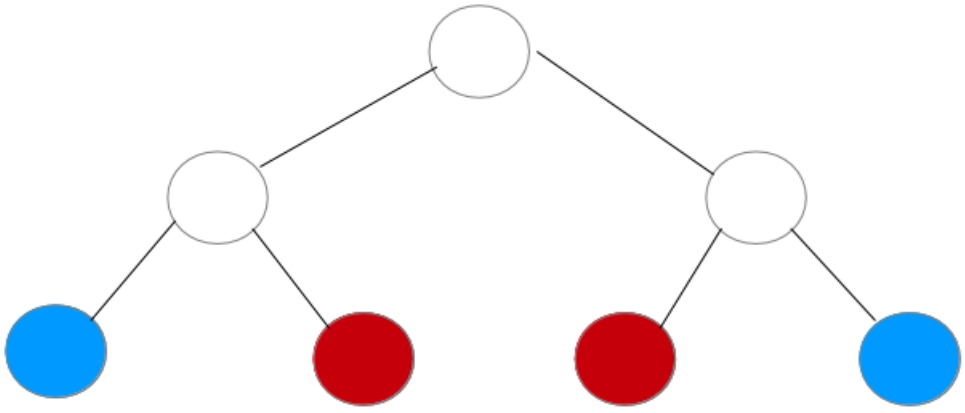

Figure 5.2: Sample decision tree. The colors at leave nodes indicate class labels. In this case, there are only 2 classes, red and blue

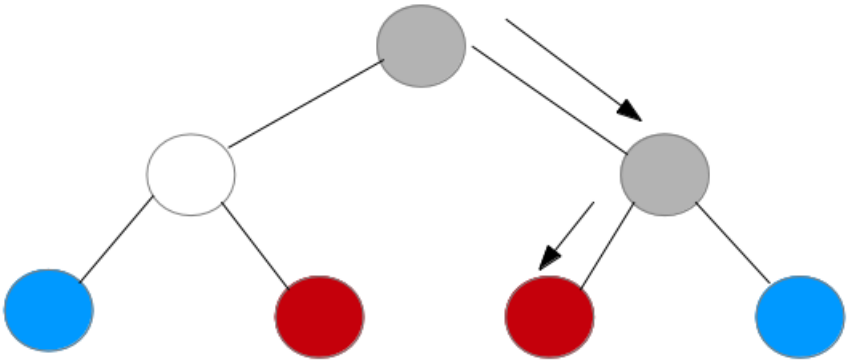

Figure 5.3: The same tree as Figure 5.2 illustrating the classfication process. Starting from the root node, the query data point is sent depending on the node binary decision. According to the arrow in the figure, the final predicted label is Red

$$G(\alpha) = H(\mathbf{X}) - \sum_{s \in \{l,r\}} \frac{|\mathbf{X}_s(\alpha)|}{|\mathbf{X}|} H(\mathbf{X}_s(\alpha)) \tag{5.1}$$

where $\mathbf{X}_l$ and $\mathbf{X}_r$ denote the set of samples partitioned to the left child node and right child node respectively. $H(\mathbf{X})$ is the entropy function.

The entropy function measures the purity or amount of information of a random variable is given as

$$H(\mathbf{X}) = -\sum_i P(\mathbf{z}_i) \log \left(P(\mathbf{z}_i)\right) \tag{5.2}$$

where $\mathbf{z}_i$ indicate an i^th class label.

The intuition of information gain function is maximizing purity differences between the current node and its children. By purity, it means that the distribution of class labels for given samples set in a node is biased towards one class and less uniform. Formally, the larger value of entropy function, $H(\mathbf{X})$, the more purity it is. That is, the function chooses the splitting parameters such that the data is splitted into maximum purity towards one class.

To implement a decision tree, we usually select a set of pre-defined weak classifiers, $\{\theta_1, \theta_2, ..., \theta_k\}$. Since we have to keep applying these weak classifiers to evaluate optimization function as such, they must be relatively cheap to compute. Usually, these are very simple classifiers such as selecting a feature and then apply threshold to make the result a binary value. In recurive fashion, we start from a root node and then keep splitting the training data to find the local optimized parameter in a given node. The splitting is stopped if any of the below conditions applied

1. $|\mathbf{X}| = 0$
2. $H(\mathbf{X}) = 0$ or $H(\mathbf{X}) = 1$
3. node depth $>$ maximum depth threshold. This terminating condition ensures that the algorithm will eventually be terminated.

The most obvious strong points of decision tree is, high dimensional data support. Usually a machine learning model try to maximize a parameter w^* where $|w^*|$ is propotional to number of features dimensions. As such it creates problems when the data is in high dimensions. With decision tree, however, the number of trained parameters depends on the depth of a tree. Plus, weak classifier responses computation are usually very simple and does not require the processing of a whole features dimensions.

Decision tree has one particular weakness, prone to overfitting. If the maximum depth threshold is set to high, then the trained tree will be very fit to the training

data. However, setting the threshold too small and the tree will not be good enough to capture the data complexity.

A solution regarding the problem is stop the splitting if $H(\mathbf{X}) <= a$ or $H(\mathbf{X}) >= b$ where a and b are lower and upper bound threshold. This will stop the tree from further expanding once it is considered pure enough. Another solution is called tree pruning which is the post processing once the completed tree is trained. The essential idea of tree pruning is reducing tree height leading to less overfitting. Lastly, we can use random forest.

5.2.2 Random Forest

The random forest classifier is essentially a collection of decision trees (hence the word forest). Given a class label, $\mathbf{z} \in \mathbb{Z}$, a data point, $\mathbf{x} \in \mathbb{X}$, and a decision tree $\mathbf{t}$, where its leaf nodes store a multinomial distribution $P_{\mathbf{t}}(z(\mathbf{x}) = \mathrm{z}_i|I)$. The final classification is the average of all decision tree results, given as:

$$P(z(\mathbf{x}) = \mathrm{z}_i|I) = \frac{1}{|\mathbb{T}|} \sum_{\mathbf{t} \in \mathbb{T}} P_{\mathbf{t}}(z(\mathbf{x}) = \mathrm{z}_i|I) \tag{5.3}$$

where $\mathbb{T}$ is the set of all decision trees and $|\mathbb{T}|$ is the number of all trees.

Random forest provides resistance to overfitting problem due to the fact that it averages the decision from all trees. Typically, each tree is trained with different datasets. Therefore, the final classification in random forest will not be biased towards one particular sub set since it will be weighted out.

Another benefit of random forest is scalability. Using random forest, we split data points into distinct sub sets. Each tree is trained based on different sets of data. As such, it is very friendly to distributed computing environment with big data as we can split the big training data to be as small as it fits the memory limitation. The

same scalable concept is also applied for testing phase. During testing, each data points can be easily processed individually since

- they are independent to one another
- memory requirement during testing is small

Having many benefits towards parallel computation and large scale data, random forest became one of the popular method in computer vision application. Each pixel is deemed as a data point and we can do classification on top of GPU since GPU is built mainly for per-pixel calculation speeding up the process to real-time level.

5.2.3 Body Part Labeling using Random Forest

In this section, we explain Shotton et al [1] work which is the inspiration for ours. In [1], the goal is to classify pixels into body part labels using a decision forest. In [1], given an image I, with pixel intensity $I(\mathbf{x})$ at position $\mathbf{x} = (x, y)$, the goal is to label position $\mathbf{x}$ with a body part label $\mathbf{z} \in \mathbb{Z}$ where $\mathbb{Z} = \{\mathrm{z}_1, \mathrm{z}_2, ..., \mathrm{z}_m\}$. Each z_i represents a body part label such as torso, head, upper arm, left arm etc. The classification task is done using a decision forest. A decision forest is essentially a collection of decision trees. Given a decision tree $\mathbf{t}$, its leaf nodes store a multinomial distribution $P_{\mathbf{t}}(z(\mathbf{x}) = \mathrm{z}_i|I)$. The final classification is the average of all decision tree results as given in Equation 5.3. The branching decision in a tree node is done using offset difference responses and then applying a threshold. According to [1], for each tree node with its associated parameters $\phi = (\theta, \tau)$, we compute

$$f_\theta(I, \mathbf{x}) = I(\mathbf{x} + \frac{\mathrm{u}}{I(\mathbf{x})}) - I(\mathbf{x} + \frac{\mathrm{v}}{I(\mathbf{x})}) \tag{5.4}$$

$\theta = (\mathrm{u}, \mathrm{v})$ parameters, where u, v are offsets in 2D space and, $I(\mathbf{x})$ is pixel intensity at location $\mathbf{x} = (x, y)$ which is depth intensity in [1], as input images are depth images. Intuitively, the equation describes pixel depth difference based on

chosen offsets. Parameter τ is a threshold, such that if $f_\theta(I, \mathbf{x}) < \tau$, then the query pixel is sent to the left child, otherwise, it is sent to the right child.

The intuition of Shotton et al [1] method is the usage of 2D random offset. The features is in very high dimensional space, as such, it can capture the spatial relationship of pixels in 2nd order. In other words, the features does not only capture relation between consecutive pixels but rather on further away ones. With the overwhelming number of data points and high dimensional features, random forest is suitable for the task of classification since it can handle both problems very well.

The drawback of this approach is, it requires the background to be clean. Different background between training set and test set results in different response of 2D offset leading to different features values which will eventually result in inaccurate pixel classification. Therefore, images with clean background or easily segmented background are required. Secondly, it requires some intuitive pixel-level features that does not change between gesture performers or can be easily normalized. Depth intensity as used in [1] is invariant between gesture performers. Plus, it also gives geometric interpretation. All of the above reasons make this method very well suited for Kinect captured images.

The body part labeling using random forest of Shotton et al [1], while works perfectly on depth images, it does not work on RGB images. To apply the idea to RGB space, we have to overcome many problems including:

- What is the proper pixel-level features to use? since depth intensity is not available.
- How to do scale normalization? or other kind of image transformation normalization between training and test set?
- How to segment the background?

  All three of the problems will be discussed in detail in the next section.

5.3 Method

In our work, the goal is to classify pixels as hand pixels or not. The method is based on decision forest. The main differences between our work and [1] are:

1. In [1], the input is a depth image, whereas in our setting it is an RGB video.
2. Since the input in [1] is a single image, they only utilize 2D offset difference. In our case, the input is a video, and therefore we can, and do, use 3D offset difference, where the third dimension is time.
3. Due to the limited amount of annotations, our class labels only consist of 2 classes, hands and non-hands.
4. We propose pixel-level features and scale normalization schemes that can be used with color videos, as the depth features and scale normalization used in [1] are not applicable here.

Formally, we denote a pixel intensity as $V(\mathbf{x})$ where V denotes a video as a frame sequence $V = \{I_1^V, I_2^V, ..., I_k^V\}$ where I_t^V is the t^{th} frame in video V, and $\mathbf{x}$ is a pixel location at $\mathbf{x} = (x, y, t)$.

Since our input is RGB video, we need to determine what pixel features we should use instead of the depth features of [1]. With depth images, the depth intensity difference of Equation 5.4 makes sense to use as a feature to infer body parts because depth changes capture geometry and also information regarding body scale. Unfortunately, that kind of information is not provided in RGB images. Since our input is not a single static image, but a video where motion occurs, this motion can be a valuable source of information for locating hands. We denote the motion score as:

$$m(V, \mathbf{x}) = m(V, x, y, t) = |I_{t+1}^V(x, y) - I_{t-1}^V(x, y)| \tag{5.5}$$

However, in some frames there is almost no motion. In this case, using motion scores provide little information as the majority of pixels have zero (or close to zero) motion values. To extract more information from motion, we use motion energy features calculated over multiple frames, as:

$$M_e(V, \mathbf{x}, s) = M_e(V, x, y, t, s) = \sum_{i=s}^{t} m(V, x, y, i) \tag{5.6}$$

where s is a parameter indicating the starting frame.

Finally, scale and time variation must be addressed. In [1], the offset is normalized by pixel depth intensity according to Equation 5.4. This makes sense because the scale of 2D offset depends on the current depth (the closer to the camera, the bigger the scale). In our case of RGB images, we use face size as a normalizing factor. The face size is defined to be the diagonal length of the detected face bounding box. Given a set of training videos, $\mathbb{V} = \{V_1, V_2, V_3, ...V_n\}$, and a function F returning a face size given a video V, $F(V)$, we compute a face size mean from the training set.

$$\mathbb{F} = \frac{\sum_{V_k \in \mathbb{V}} F(V_k)}{|\mathbb{V}|} \tag{5.7}$$

The scale normalizer for a given video Q ,is defined as

$$S(Q) = \frac{\mathbb{F}}{F(Q)} \tag{5.8}$$

In a nutshell, the normalizer value is the ratio between the mean of training face sizes and the face size in the input image. Note that $S(Q)$ is different for each tree since we use a different training set for each tree, so $\mathbb{F}$ is different for each tree.

The frame rate difference between the training and the input video is another issue to consider. To address this issue, we apply the same strategy as in scale

normalization. In this case, the frame rate is the value of interest. Let us denote a frame rate of a given video V as $R(V)$. We compute the average frame rate on a given training set, $\mathbb{V}$ as

$$\mathbb{R} = \frac{\sum_{V_k \in \mathbb{V}} R(V_k)}{|\mathbb{V}|} \tag{5.9}$$

The time scale normalizer, $T(Q)$, is defined as

$$T(Q) = \frac{\mathbb{R}}{R(Q)} \tag{5.10}$$

The intuition in Equation 5.10 is the same as in Equation 5.8. Combining scale and time normalization, we obtain the normalized offset vector, $N(\mathbf{x}, V)$, as

$$N(\mathbf{x}, V) = N(x, y, t, V) = [\frac{x}{S(V)}, \frac{y}{S(V)}, \frac{t}{T(V)}]^T \tag{5.11}$$

Using the motion energy score from Equation 5.6 and adapting the normalized offset from Equation 5.11, Equation 5.4 becomes

$$\begin{aligned} f_\theta(V, \mathbf{x}, s_1, s_2) \quad = \quad & M_e(V, \mathbf{x} + N(\mathrm{u}, V), \frac{s_1}{T(V)}) - \\ & M_e(V, \mathbf{x} + N(\mathrm{v}, V), \frac{s_2}{T(V)}) \end{aligned} \tag{5.12}$$

$M_e(V, \mathrm{x}, s)$ here is the motion energy response computed using Equation 5.6. Both u and v now are 3D space offsets (thus, 3D vectors) where the third dimension is time. We should note that, based on the above, the parameter θ, which specifies the information that is used at each decision node, is an 8-dimensional vector, consisting of 3D offsets u, v and scalars s_1, s_2.

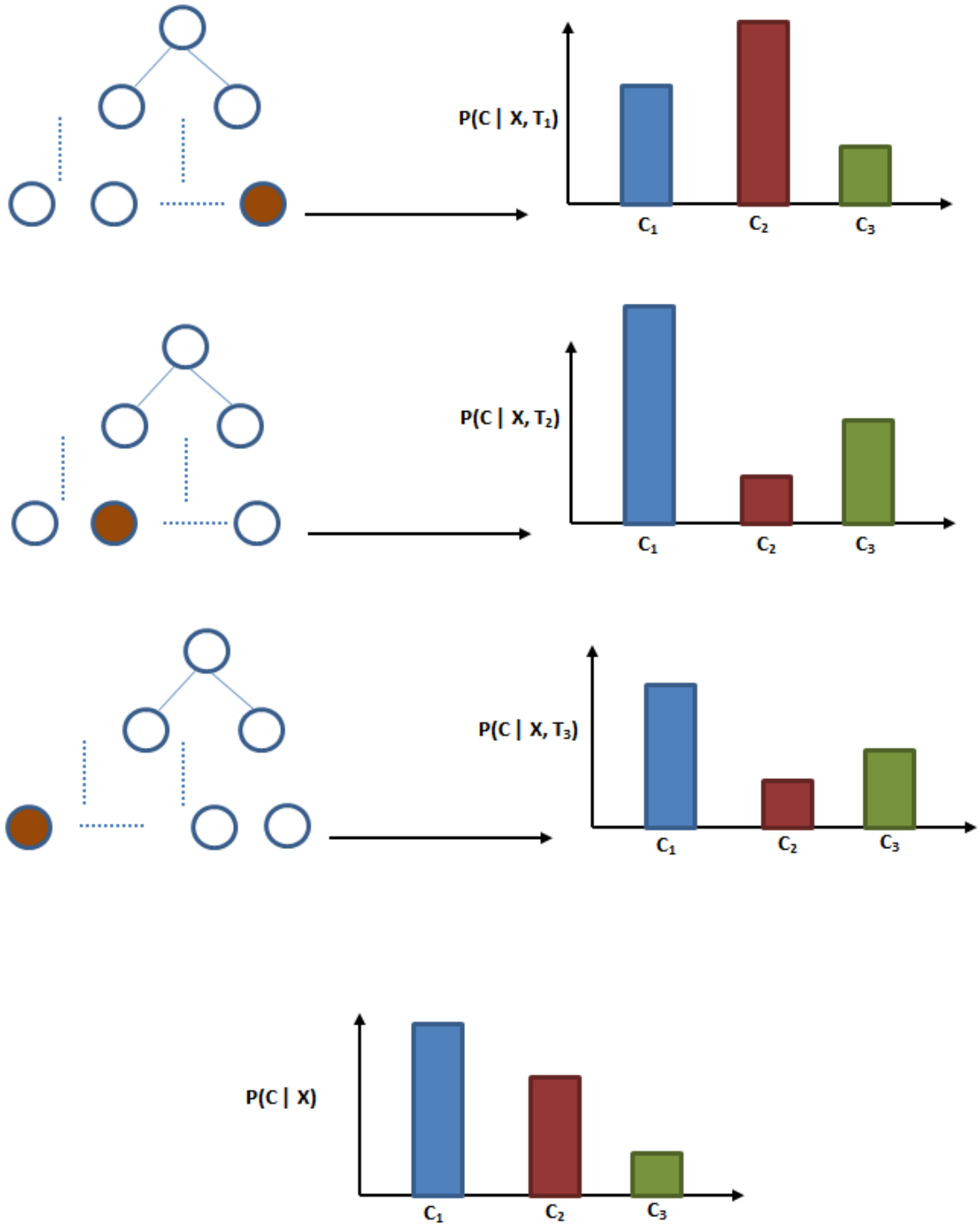

Figure 5.4: An example of inference in a random forest with 3 trees. The red node indicate the final decision leaf node containing a posterior probability, $P(C|X, T_i)$. There are 3 classes in this example. The final posterior probability, $P(C|X)$ is the average of all trees

CHAPTER 6

Experiments and Results

6.1 Experiments Setup

The experiment is conducted on multiple sign language datasets in a user independent scenarios where a test signer will never appear in training set. Training samples will include both left and right hands as we will train one single hands detector, not 2 separate detectors for each hand.

we will compare our method with one baseline method and 3 current state-of-the-art methods of hands detection. Namely, we used the simple skin and motion method of [75] as a baseline method, and we compared also with the state-of-the-art methods of multiple proposal hand detectors [23], Chain model [24] and Discriminatively Trained Deformable Part Models of Felzenszwalb et al [76]. The reasons for picking these methods is because they are all famous well known methods. The multi-proposal method [23] was selected as best industrial paper at BMVC'2011 conference. Chain model [24] was an oral presentation at CVPR'2010. Finally, Deformable Part Model by Felzenszwalb is very famous as it is considered as the state-of-the-art in term of windows based matching method.

6.2 DataSets

6.2.1 American Sign Language Dataset

The dataset consists of 1,113 signs from 3 different signers making a total of 3,339 videos of signs. The videos were recorded on RGB cameras on frontal view having signers sit in front of the cameras performing various signs. Faces and hands

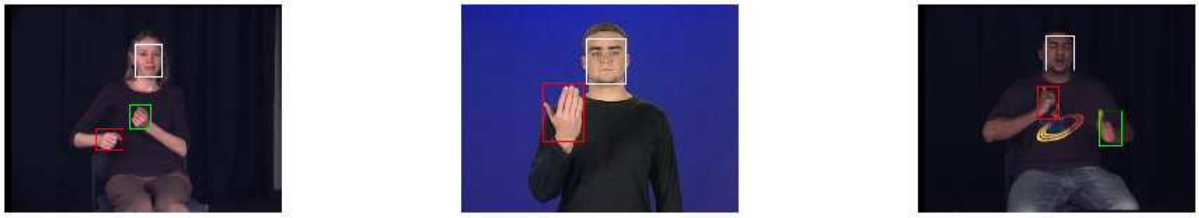

Figure 6.1: Sample images with annotated region from ASL dataset. Each image is a sample from different signer. Rectangle box displays annotated region including face and hands. From left to right, sample images from lb1113, gb1113 [77] and tb1113 datasets respectively.

locations ground truth are available based on human annotated labels. ASL dataset has its own difficulty such as

Table 6.1: ASL Dataset Statistic

Properties	lb1113	tb1113	gb1113
No. of signs	1,113	1,113	1,113
No. of 1-handed signs	389	389	389
No. of 2-handed signs	724	724	724
No. of signers	1	1	many
Frame size	640×480	640×480	352×240
Background color	Black	Black	Blue
No. of frames	38,806	35,812	26,827
No. of 1-handed frames	12,192	11,880	8,612
No. of 2-handed frames	26,614	23,932	18,215
No. of hands annotation	65,420	59,744	45,042

6.2.1.1 Data characteristic

1. Large number of classes - ASL dataset has a total of 1,113 classes. Using random guess, the accuracy is $\frac{1}{1113} = 0.00089$, less than 0.1% accuracy. As such, this makes it difficult to achieve high number of recognition accuracy.

2. Signer independent scenarios - In all our experiments, we used signer independent scenarios. With this kind of experiment set up, there is different in scale, speed, background and other image transformation between training set and test set. Therefore, the model must be able to handle this differences.
3. Unclear hand shapes - Some signers prefer to perform gesture on top of their faces. This, along with low resolution images, makes shape detection method such as template matching or deformable models do not work well.
4. Minor hand shapes difference between signs - Some signs are very hard to distinguish between one another due to very minor difference in hand shapes. To be able to correctly classify the sign, the classifier or features must be able to capture this minor variation. As such, some signs are very difficult to recognize correctly.
5. Few number of per class samples - Each class (sign) has only 3 videos. If we leave one out for testing, then there are only 2 videos left for training. Due to this fact, some probabilistic models that require separate training data for each class such as Hidden Markov Model does not work well on ASL.
6. Clean background - All videos in ASL dataset have static, non-moving, clean background. This makes it perfectly fit for random offset features.
7. The availability of annotations - ASL dataset comes with start and end frame annotation, along with hands and face location labels on all frames in all 3,339 videos, making it a huge training set for hands detection models.

6.2.2 TV Footage Dataset

The TV footage dataset [2] is a publicly available dataset consisting of 6,000 images captured from news broadcasting. However, among these 6,000 images, there are only 342 images with annotated body parts. the images has lots of background

Figure 6.2: Sample images from TV footage dataset [2].

noise since the dataset was recorded from television footage without any academic guidance. In our experiment, we randomly select 228 of annotated images as a training set and use the remaining 114 as a testing set. To implement our method, we follow the background segmentation of [73] to segment the signer from the noisy background. Then, we run hands detectors on segmented images. Note that all the competitors' results are all obtained on segmented images as well.

6.2.2.1 DataSet Characteristic

1. Noisy background. The data was captured in the wild from the news footage where a portion of the screen is the signer performing sign describing the news. Without proper segmentation, the random forest algorithm would certainly failed.
2. Same scale and frame rate. The data was captured from the same source. As such, signer scale and frame rate stays consistent across training and test set.
3. Small dataset. The BSR dataset consists of 6,000 TV footage images from a video sequence. However, only 342 of those are annotated with hands regions. As such, this dataset is considered small comparing to ASL dataset.

Figure 6.3: Hands segment for training examples

6.2.3 Implementation on ASL dataset

Since our random forests perform pixel-level classification, we need as training data annotations at the pixel level, to indicate if the given pixels are hand or not-hand pixels. To do that, we crop the annotated hand region, and then we apply the GrabCut algorithm using code from [74] to remove the background from the image, so that only hand pixels are left over. Figure 6.3 shows some samples of cropped hands.

To obtain the results, we used cross validation. That is, for each of the three sub datasets, we used that subset as a test set, where hands were detected using a model trained on the other two subsets. This gives us user independent results. When using a subset as the test set, we sample 100 testing videos out of the 1,113 videos from that subset. For all 3 sub datasets, this makes a total of 300 videos consisting of 8,392 frames as testing set. The reason for not using all videos for each test set is that, because of the long running time of the competitor methods [23, 24], we did not have time to apply those methods to all 3,339 videos.

At training time, due to the large number of available training data (as any pixel is a data point), we used sampling to reduce the number of training data. Firstly, for any given training video, 5 frames are selected randomly. Then, for each of the selected frames, the frame size is resized to 160×120. All extracted hand pixels are used as positive samples. Due to the large number of negative samples, we use more

sampling in order to reduce the number of negative samples. First, we filter the image using a skin detector on RGB color space [78]. Skin pixels are given more preference to be selected as negative samples as they are more likely to be misclassified. If n is the number of positive pixels, we select $2.5n$ negative samples.

To sum up, we do the following procedure for training

1. Use 2,226 videos from two sub datasets as training set, and 100 videos from the third sub dataset for testing.
2. For each training video, randomly select 5 frames.
3. For each frame, scale down the image to 120×160.
4. Apply skin detector based on RGB colorspace histogram [78]. Pixels whose skin score is below 0.1 are removed.
5. Use all hands region pixels extracted using GrabCut algorithm Toolbox [74] as positive samples
6. For negative samples, randomly select $2.5n$ pixels, where n is the number of positive pixels retrieved from that frame. Skin pixels are given more preference.
7. Split the number of training samples equally to each decision tree. The crucial factor here is that training samples from the same sign are supposed to be spread out to as many trees as possible.
8. Train each decision tree separately. Each tree has its own set of randomly selected parameters, θ, and is trained on a different training set. We use maximum information gain as shown in Equation 5.2 as a split parameter decision. The node splitting stops if it meets any of the stopping conditions described previously.

For hands bounding box detection, we use a 2D Gaussian filter with standard deviation parameter set to 5. The size of the bounding box is 21×21 pixels.

6.3 Implementation on TV Footage dataset

1. We begin with splitting 342 annotated images into training and test set, using 2/3 of the set which comes at 228 images as training set and the remaining 114 images as test set.
2. We adapt the background segmentation of [73] to segment the foreground from the background. This process is essential to random offset features as introducing motion noise in the background will severely damage the final accuracy. However, note that for every competitor's methods evaluation is also done through the segmented images as well.
3. We follow the same procedure as ASL implementation in term of training data selection. That is selecting all positive samples and sample on negative samples data depending on motion and skin scores. However, due to much smaller training size, the images are not scaled down.

6.3.1 Comparison Methods

We selected one simple method utilizing simple linear combination of motion and skin score [75] as a comparison based line system. This method compute the skin probability using naive Bayesian method along with motion score to compute the final hands pixel classification. The bounding box level detection is done through the use of area averaging through Gaussian filter. While the method sounds very simple, it has been proved to works quite well in ASL dataset.

Next we select Deformable Part Model [76] by Felzenszwalb et al as one of the comparison methods. The method is one of the most famous template matching model in the object recognition topic. The model is based on a graphical model utilizing hidden variables. The intuition of the idea is that, each interest object is composed of components. For examples, a bicycle is composed of tires, a handle etc.

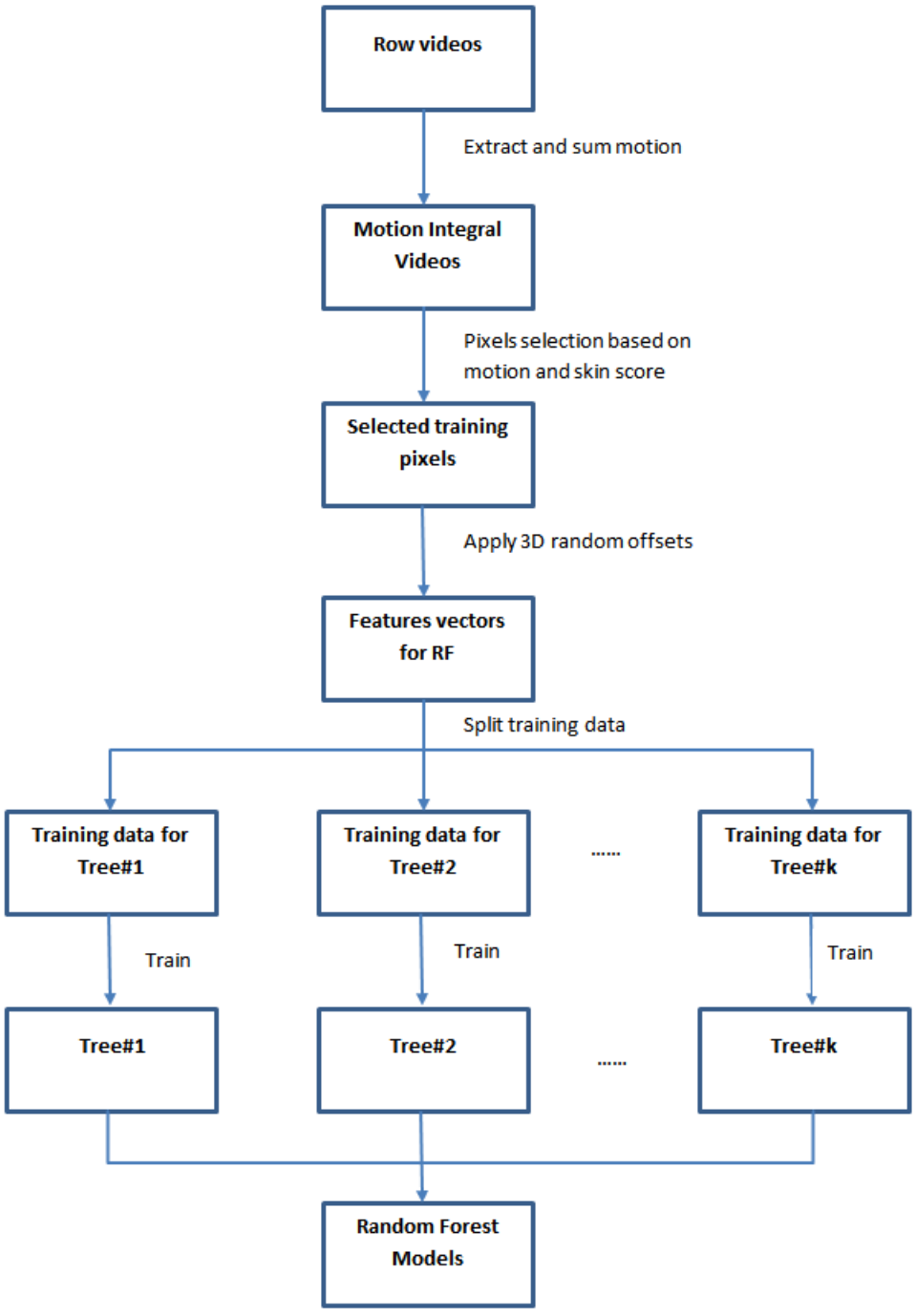

Figure 6.4: A Flowchart summarizing random forest training process

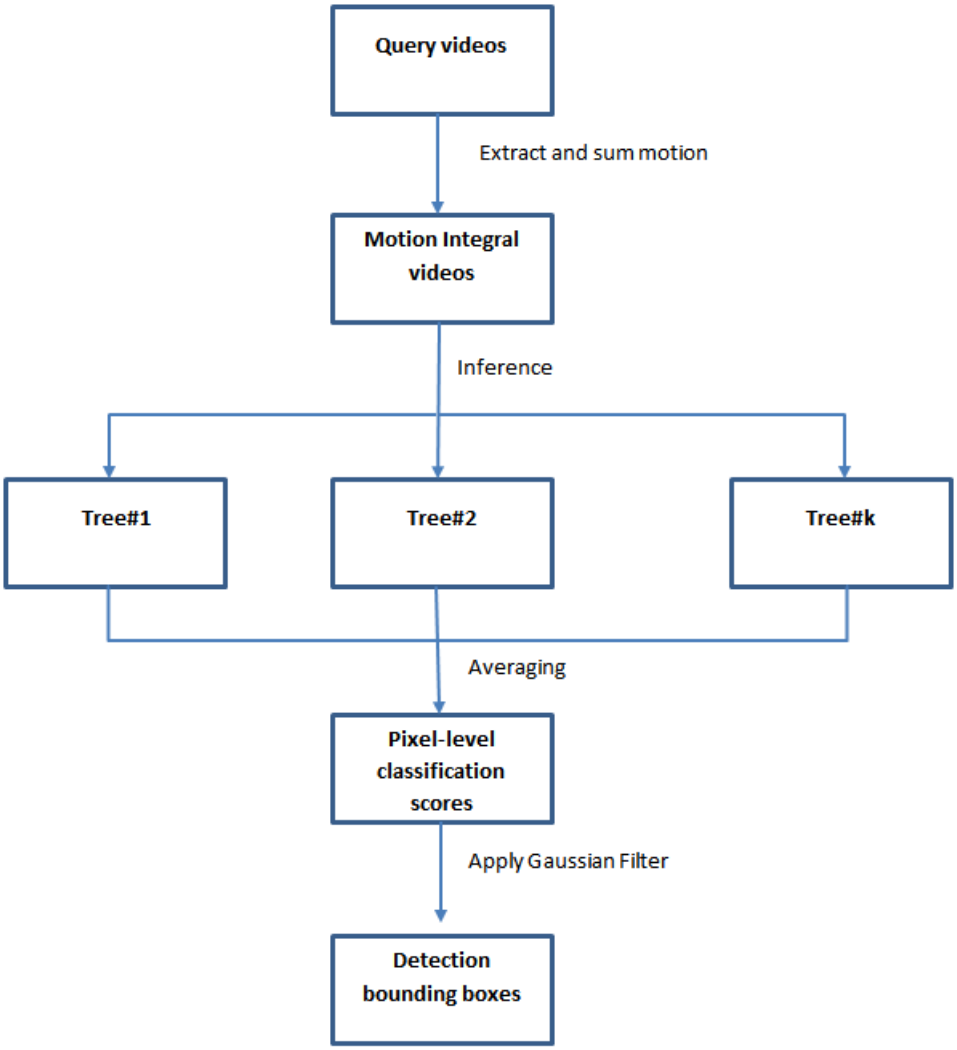

Figure 6.5: A Flowchart summarizing random forest testing process

Detecting individual components would lead to the global objects detection. Since these variables (components) are latent, there is no need for manual annotation for each component. The only required annotation region is the entire bounding box of the objects. The method is extremely famous as it has won the PASCAL object detection in 2010.

Chain detection method [24] is a recent hands detection literature published in Computer Vision and Pattern Recognition (CVPR) 2010. The model of the method is, again, a graphical model. However, unlike many other works, the graph structure is a chain instead of a star. The intuition is that, given a known reliable starting point, for examples, face, we can trace as a chain to the interest object (eg. hands). The annotation requirements of this method is the reliable starting location which is face in our dataset and the hands locations.

Finally, the multi-proposal method [23] won the best industrial paper award from British Machine Vision Conference (BMVC) in 2011. The idea of the method is using combination of classification scores as based features for the final SVM classifier. The based classifier consists of hand shape classifier, context classifier and skin region classifier.

6.3.2 Quantitative Measures

Given a detected bounding box center position for frame I, $h(I)$, a center position of hand bounding box ground truth, $g(I)$ and a face width in a given frame, $f(I)$, we define our accuracy measurement function as

$$accuracy(h(I), g(I), f(I)) = \begin{cases} 1; & \|h(I) - g(I)\| <= \frac{f(I)}{2} \\ 0 & Otherwise \end{cases} \tag{6.1}$$

Intuitively, the face size is used to determine the threshold of correct distance between detecting boxes and ground truth.

If the hand detector has produced K candidate bounding boxes, we consider each of the true hand locations to have been correctly detected if the above equation gives 1 when applied to that hand location and one of the K candidate positions. We note that, for 1-handed signs we are only concerned with detecting, at each frame, the

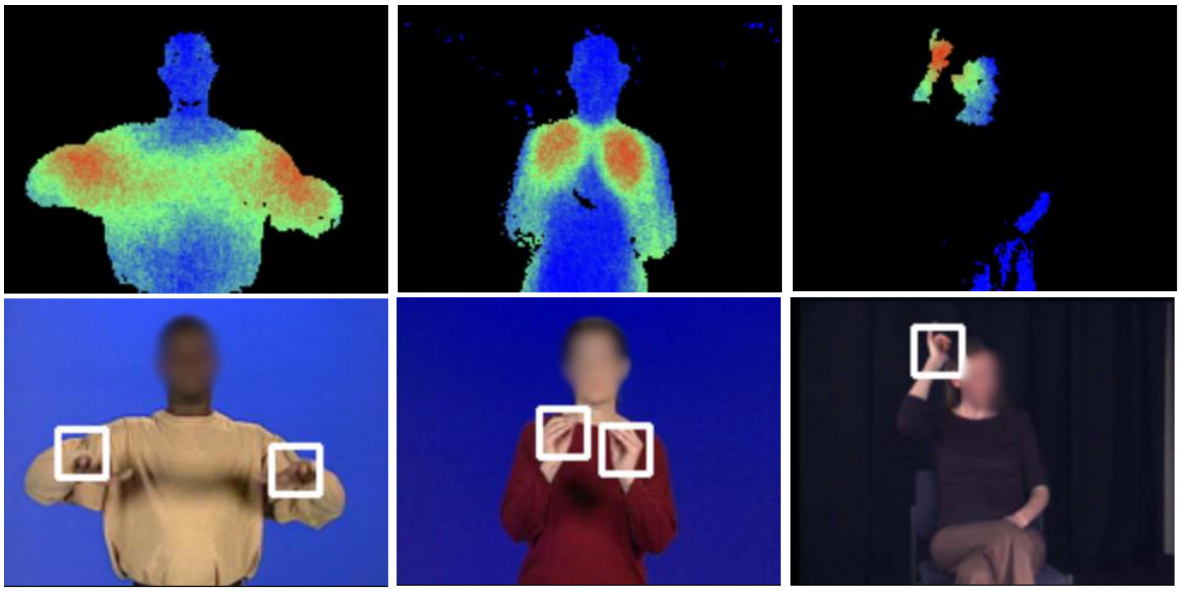

Figure 6.6: Classification score visualization in color coded images and their corresponding hands detection result. From low to high score, color changes from blue-green-red.

hand used for signing, so the ground truth contains only one true hand location per frame. For 2-handed signs the ground truth contains two hand locations per frame.

To compute the percentage of accurate detection over a set of frames, we simply compute the ratio of true hand locations that have been correctly detected for those frames, over the total number of true hand locations that the ground truth contains for that set of frames.

6.4 Results

6.4.1 Pixel-level classification

First, we measure the quality of pixel classification. Here we use annotations that label each pixel as a hand or non-hand pixels, and we compare those annotations with the pixel-level classification produced by the decision forest and the skin and

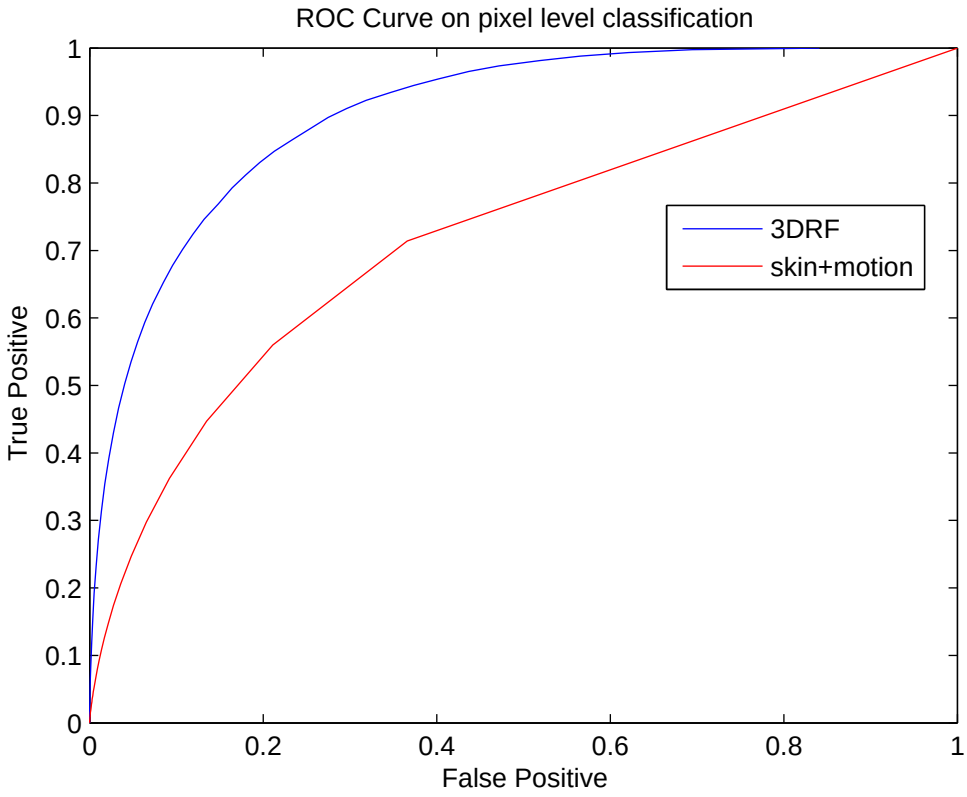

Figure 6.7: ROC Curves on pixel level classification on ASL dataset. X-axis represents false positive rates and Y-axis represents true positive rates.

motion detector method of Alon et al. [75]. Figure 6.7 displays ROC curves obtained by varying the classification threshold. The highest F1 score of our method is 0.8260 comparing to 0.68 of skin and motion method. Unfortunately, we cannot acquire the pixel level classification on other competitor's methods including multiple proposal hand detectors [23], Chain model [24] and Discriminatively Trained Deformable Part Models of Felzenszwalb et al [76] since those methods do not classify each pixel individually but rather give a windows based classification.

6.4.2 Comparisons with state-of-the-art methods

In this experiment, we compare our method with one baseline method and 3 current state-of-the-art methods of hands detection. Namely, we used the simple skin and motion method of [75] as a baseline method, and we compared also with the state-of-the-art methods of multiple proposal hand detectors [23], Chain model [24] and Discriminatively Trained Deformable Part Models of Felzenszwalb et al [76]. For the method of [23], we used the implementation provided by the authors of that paper. However, the authors of [23] were not able to provide us with training code (which we did request), so we did not train that method on our training data. Instead, we simply used the already trained detector that the authors provided us with. On the other hand, for both method of [24] and [76], the authors provided us with training code as well, so the results that we show for these 2 methods were obtained by using the same training data (per test set) as in our method.

6.4.2.1 Results on ASL dataset

Figure 6.8 and 6.9 show the results on 1-handed and 2-handed signs respectively. The x axis shows the number of candidate bounding boxes produced by the hand detector. This number is a free parameter that we can set as we wish, for each of the detector methods we evaluate. The Y-axis represents accuracy. For example, point(7, 0.8) on the plot means that when using 7 candidates per frame, for 80% of true hand regions, there is a correct candidate within the threshold used in Equation 6.1.

It can be seen that our approach gets better accuracy than the baseline skin and motion method of [75], as well as the multiple proposal [23], Chain model [24] and Deformable model [76], for both 1 handed and 2 handed signs, regardless of the number of hand candidates. Looking at number of candidates $k = 1$ for 1-handed signs and $k = 2$ for 2-handed signs respectively, our approach had an accuracy of

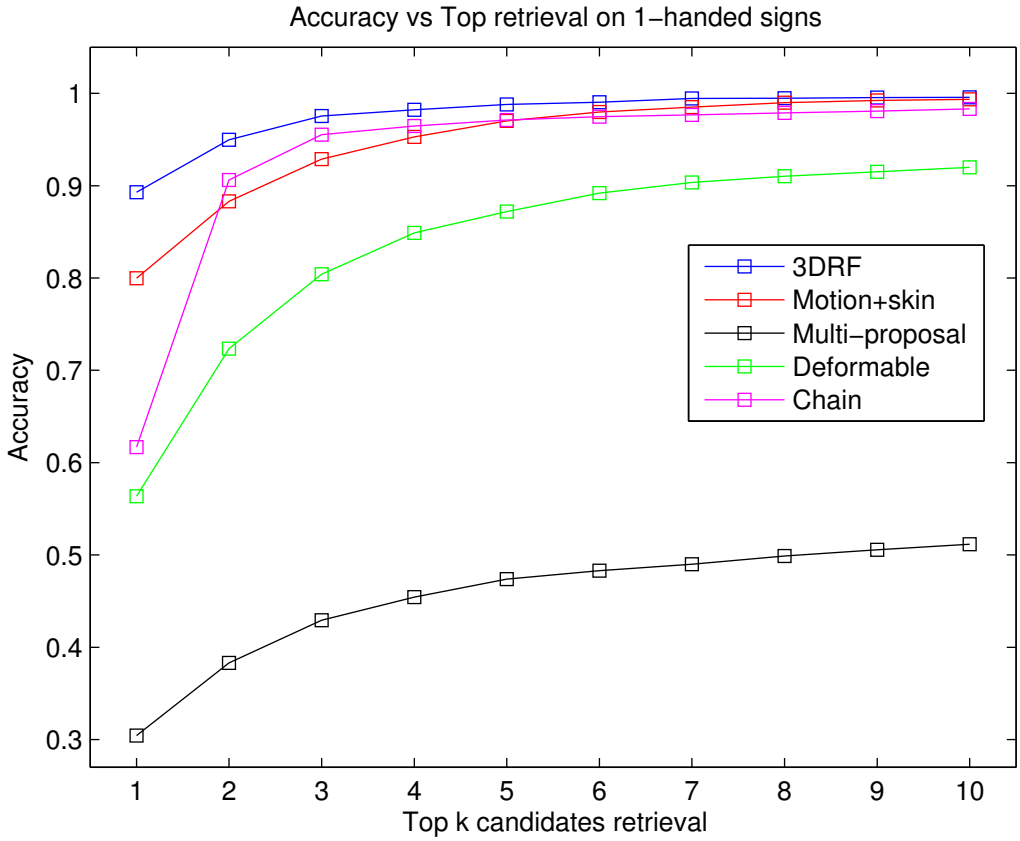

Figure 6.8: Accuracy vs Top k bounding boxes retrieval on 1-handed signs. X-axis represents number of detected bounding boxes and Y-axis is accuracy. 3DRF is our method, Motion+skin is the skin and motion detector of [75]. Multi stands for the multiple proposals method of [23], Chain stands for the chain model of [24] and Deformable Model stands for Discriminatively Trained Deformable Part Models of Felzenszwalb et al [76].

89.29% for 1-handed signs and 86.18% for 2-handed signs. Chain model [24] offers comparable results to our method especially at higher number of retrieval candidates. However, it does not get high accuracy at top-1 candidate, the most interested number in our experiment. Deformable Part Model [76], despite being the state-of-the-art method in term of windows matching model, does not work well according to the experiment. This is because hands shape in sign language can be in any articulated

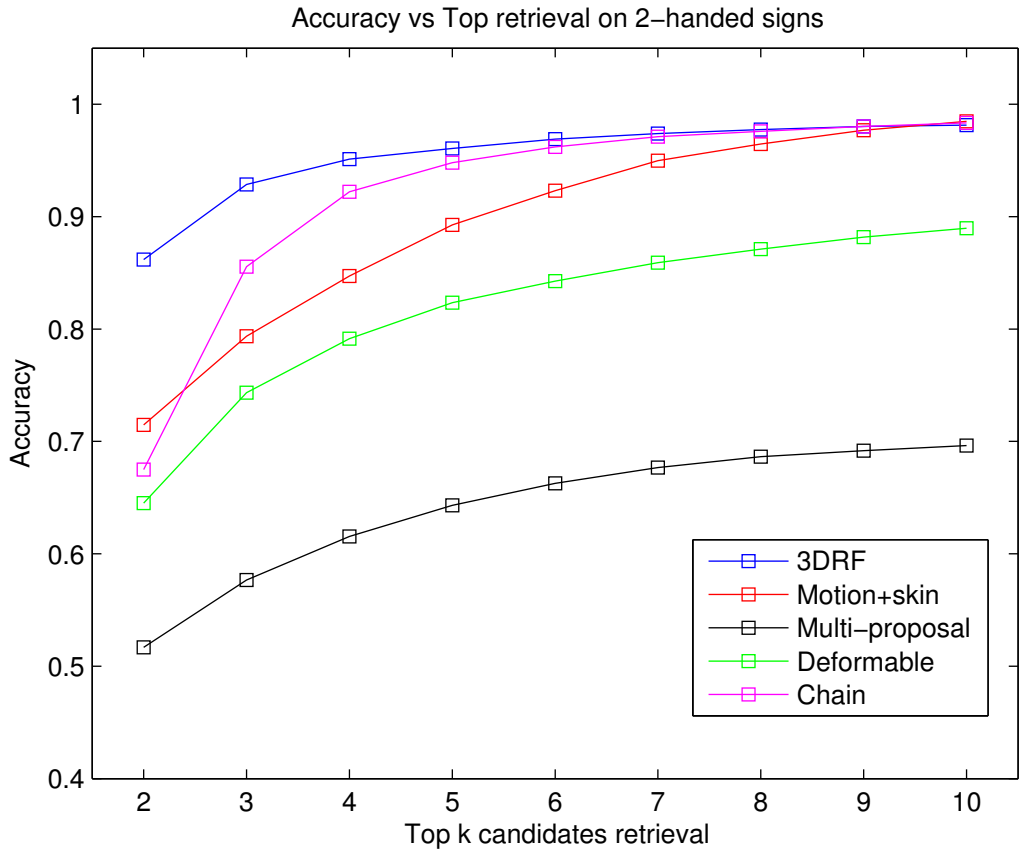

Figure 6.9: Accuracy vs Top k bounding boxes retrieval on 2 handed signs. X-axis represents number of detected bounding boxes and Y-axis is accuracy. 3DRF is our method, Motion+skin is the skin and motion detector of [75]. Multi stands for the multiple proposals method of [23], Chain stands for the chain model of [24] and Deformable Model stands for Discriminatively Trained Deformable Part Models of Felzenszwalb et al [76].

shapes. Training one template model would not be suffice to capture all shapes, as such, resulting in mediocre results. This experiment confirms our assumption mentioned in the previous chapter.

The reason our method works better than Multiple proposal [23], Chain model [24] and Deformable Model [76] is that, in all of these approaches, the goal is finding hands in static images. All three approaches use gradient based features like HoG

or SIFT as input features to represent shapes. However, in motion frames, hand shape appearance is not so clear due to motion blur from movement whereas our approach focuses on motion. On the other hand, our method uses motion, which is a powerful cue, not available for use in the static images that [23], [24] and [76] work with. Interestingly, even the baseline skin and motion method also works better than [23] and [76], because although it uses a very simple appearance model (just skin and non-skin color histograms), it also uses frame differencing, which is a powerful cue not utilized in [23] and [76].

It can be seen that the accuracy of Deformable model [76] provides better results than Multi proposal approach [23]. This might come as a surprise since [23] is the improved version of [76], designed specifically for hands detection. However, due to the fact that we have trained the model on our dataset for [76] experiment, whereas we use pre-trained model for [23], this makes [76] gets better results than [23] since the model fits more to the dataset.

In general, accuracy on 1-handed signs is better than 2-handed signs. This is because, in general, 2-handed signs have much more motion variation than 1-handed signs. Thus, they are harder to recognize.

Another factor worth mentioning is running time. Both [23] and [24] takes about 1-2 minutes per frame to detect hands, [76] takes roughly about 5 seconds, while our approach only takes an average of 1.58 seconds per frame on Intel Xeon E3-1270 at 3.5 GHz desktop machine. Times are measured using a single thread implementation, and a desktop machine based on 160×120 image. The running time can be further improved by implementing on GPU, a route which we have yet to explore.

Figures 6.10 and 6.11 shows 3D random forest results on individual signers for 1-handed signs and 2-handed signs respectively. For 1-handed sign, the accuracy for

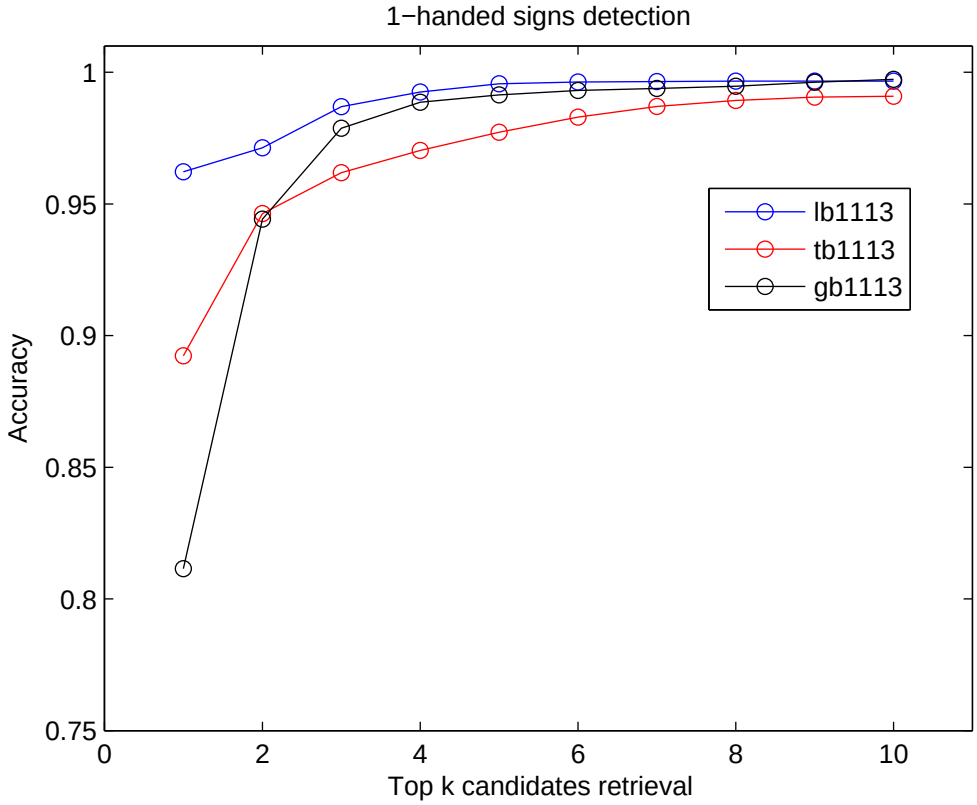

Figure 6.10: Accuracy vs Top k bounding boxes retrieval on 1-handed signs on each individual signers. X-axis represents number of detected bounding boxes and Y-axis is accuracy.

top 1 retrieval is 96.22%, 89.24% and 81.15% for lb1113, tb1113 and gb1113 signers respectively. For 1-handed sign, the accuracy for top 2 retrieval is 88.36%, 84.13% and 83.72% for lb1113, tb1113 and gb1113 signers respectively. According to the result, lb1113 is the best in term of detection accuracy, tb1113 comes second and gb1113 is the last.

The reason gb1113 has the worst accuracy is because both lb1113 and tb1113 was recorded from the same camera in the same room and in the same environment. As such, fundamental image transformation such as scale, illumination, frame rate and background color remains the same. This does not hold true for gb1113 where the

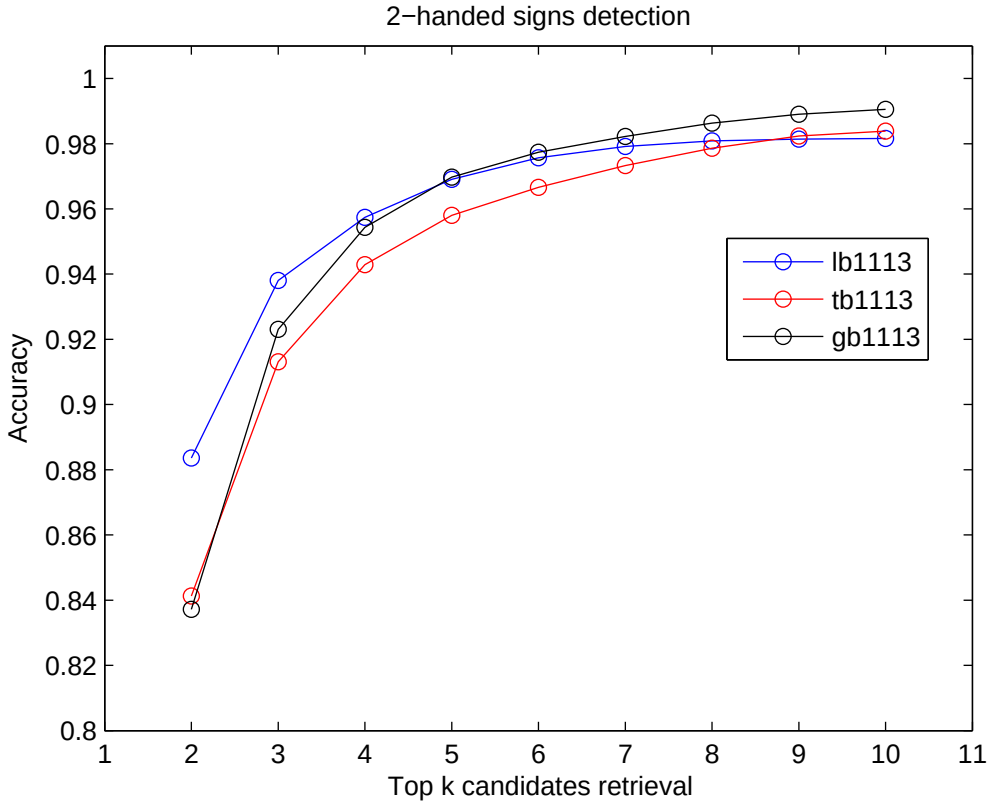

Figure 6.11: Accuracy vs Top k bounding boxes retrieval on 2-handed signs on each individual signers. X-axis represents number of detected bounding boxes and Y-axis is accuracy.

data consists of multiple signers wearing different clothing and have half frame rate of that of tb1113 and lb1113. In our experiment, when testing on tb1113 or lb1113, the training set contains large number of videos with the same environment. Consequently, scaling and frame rate normalization problems has been reduced. For gb1113 testing, however, both scale and frame rate normalization still holds. According to the results, this shows that there are still more room to improved on normalization.

6.4.2.2 Results on TV Footage dataset

In this section, we conduct the experiment on British TV Footage dataset [2]. The dataset has a total of 6,000 TV footage captured images but there are only 342 images with annotated hands region. As such, we randomly select 228 of annotated images as a training set and use the remaining 114 as a testing set. To implement our method, we follow the background segmentation of [73] to segment the signer from the noisy background. Then, we run hands detectors on segmented images. Implementation detail is the same as we described in section 6.2.3 without cross validation method and frame sampling process. Note that all the competitors' results are all obtained on segmented images as well.

Figure 6.13 shows the results on TV Footage dataset. As can be seen, 3DRF achive 98% detection accuracy on top-2 detection. Such high accuracy comes from the fact that this is user dependent scenario. As such, offset normalization is no longer an issue. Chain model [24] also get very good result at 92% accuracy on top-2 retreival. Overall, our method works better than all competitors with the exception of Chain model [24] that gives comparable results.

It can be seen that the results on TV footage dataset is much better than that of ASL for 3DRF and other models as well. This is because, on TV footage datset, the training set and test set are user dependent. They are both captured in the same environment. As such, it is reasonable to get better result on TV footage dataset than ASL.

6.4.3 Compare 3D Offset Space with 2D Offset Space

In this experiment, to demonstrate the importance of 3D offset space, we compare the difference using 3D offset and 2D offset space. To implement 2D offset space,

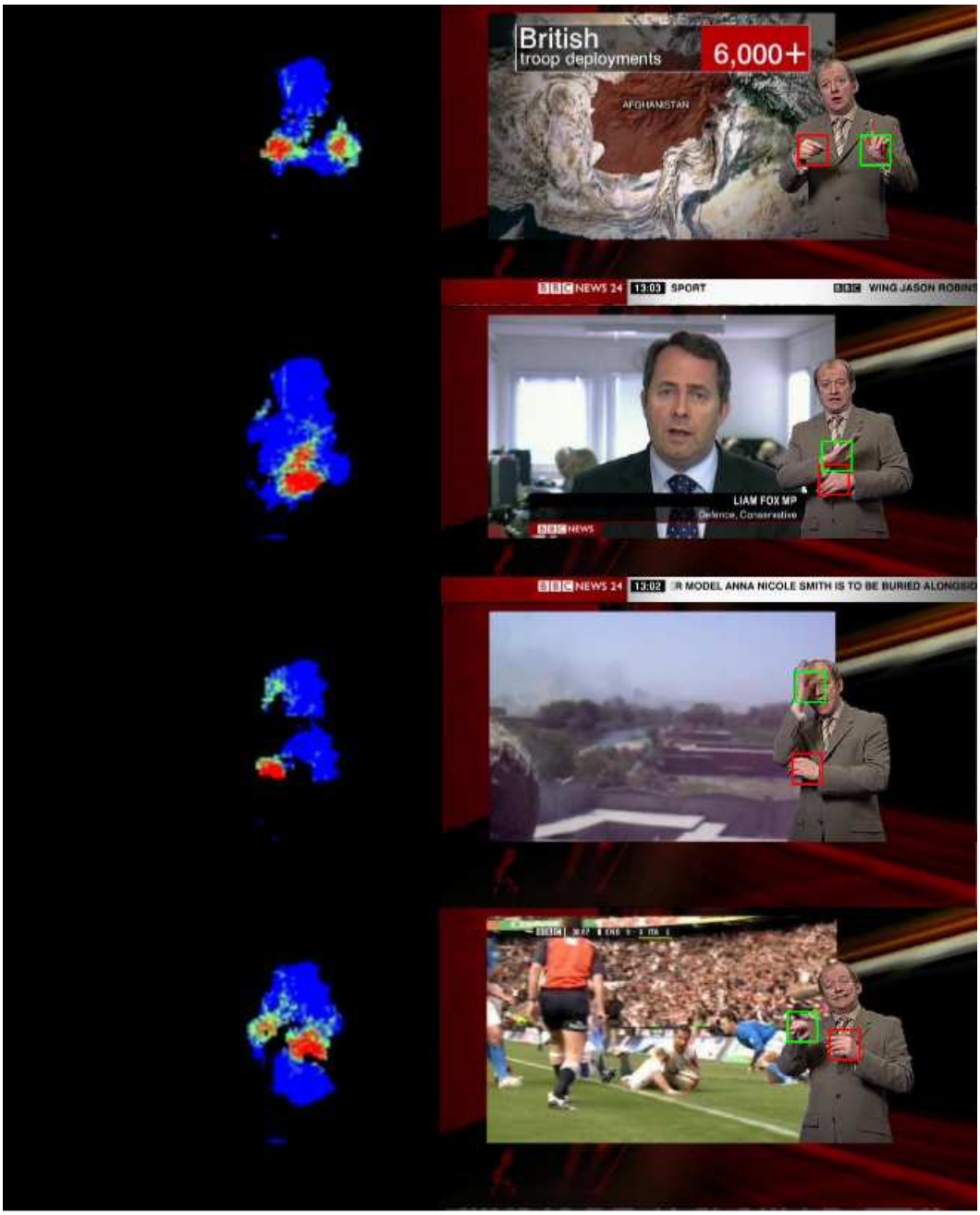

Figure 6.12: Results visualization on TV Footage dataset [2]. The left panel is a color coded on pixel classification score where from low to high score, the color ranging from blue-green-red. The right side is the corresponding detection bounding box on the original images

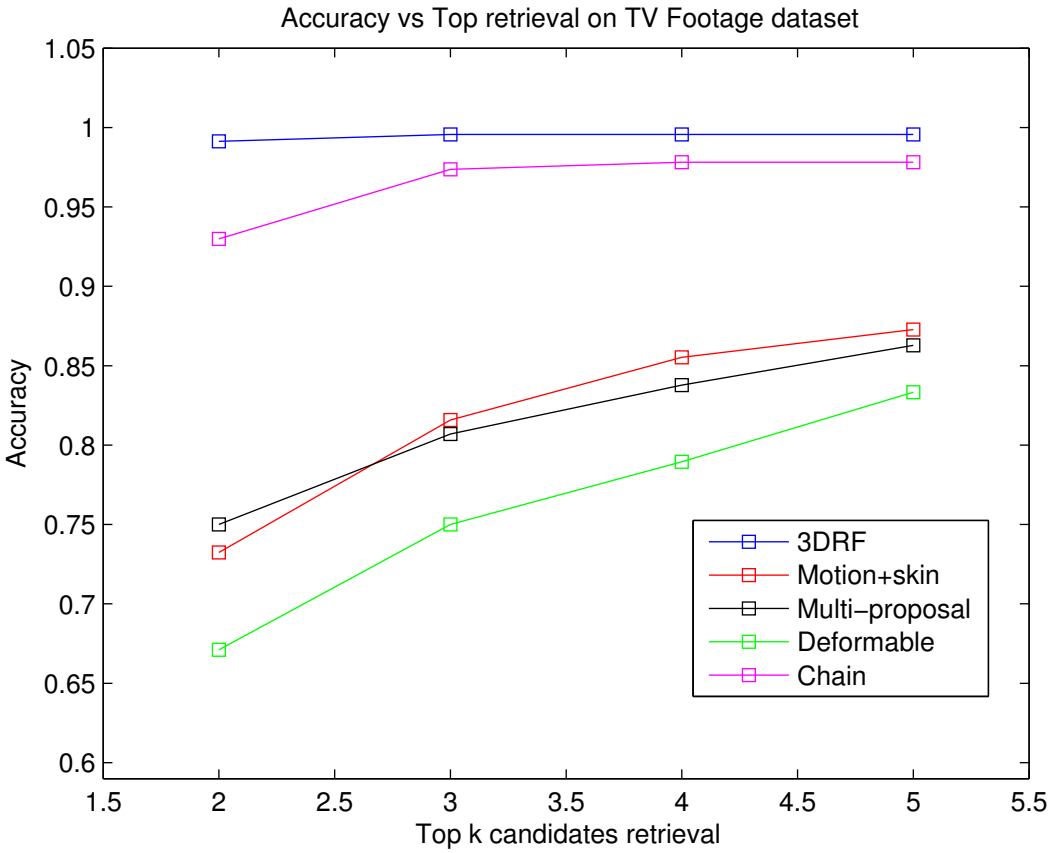

Figure 6.13: Accuracy vs Top k bounding boxes retrieval on TV Footage dataset. X-axis represents number of detected bounding boxes and Y-axis is accuracy. 3DRF is our method, Motion+skin is the skin and motion detector of [75]. Multi stands for the multiple proposals method of [23], Chain stands for the chain model of [24] and Deformable Model stands for Discriminatively Trained Deformable Part Models of Felzenszwalb et al [76].

we still use motion integral as pixel features as shown in Equation 5.6. However, the offset difference value is now computed with Equation 5.4 as in the work of Shotton et al. [1] and Charles et al. [73], so that we always compare pixels of the same frame.

Figure 6.14 shows the experiment result. According to the result, we get significantly better accuracy with 3D offset than 2D offset for both 1-handed and 2-handed frames. With 3D offsets, we achieve 89.29% and 86.18% accuracy on 1-handed and

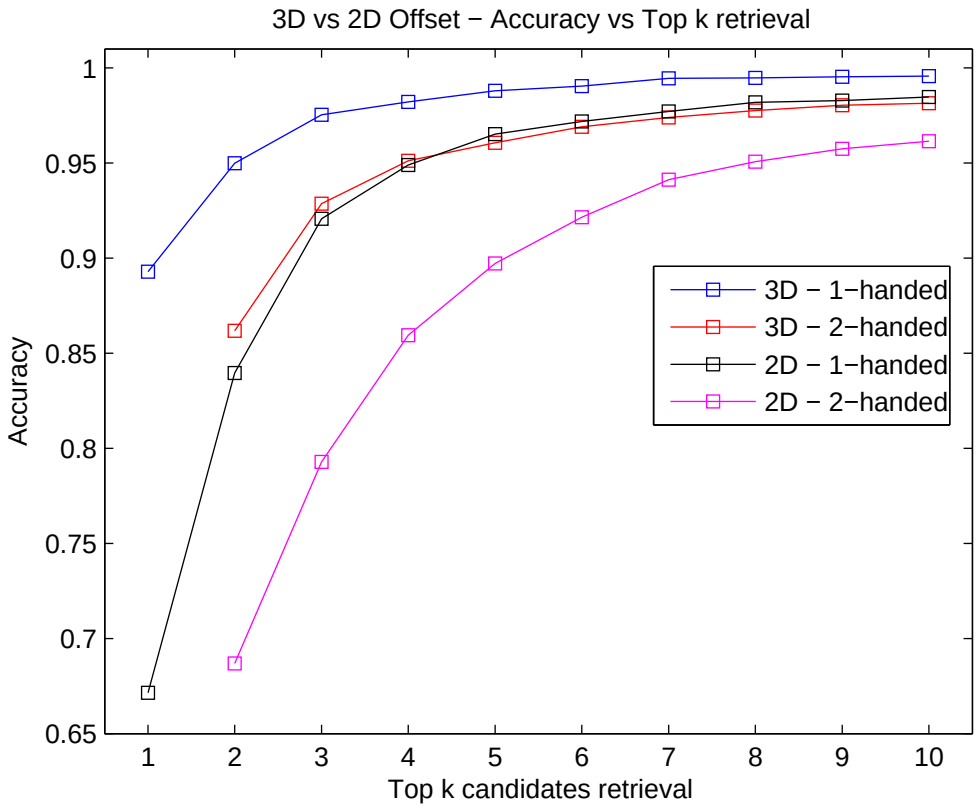

Figure 6.14: Accuracy vs Top k bounding boxes retrieval on both lb1113 and gb1113 dataset, comparing results between 3D offset space and 2D offset space.

2-handed signs with $k = 1$ and 2 respectively, compared to 67.14% and 68.68 % accuracy when using 2D offsets. We conclude from this experiment that adding another dimension to the offset space yields significantly better detection accuracy.

6.5 Discussion and Future Works

There are many aspects of 3D Random Forest that can be improved. First, let us discuss the problems of the approach.

1. Random offsets only works if the image background is clean or consistent across training/ test sets. Otherwise, it would not work. For examples, if the back-

ground of the images contains motion all over the place, it will introduce noise unrelated to the gesture, resulting in noisy features.

2. Unlike the extreme popular deep learning model, features engineering is still required in random forest since the model itself is not sophisticated enough to capture the pattern from raw pixels.
3. The idea of scaling and frame rate normalization using face size is not mathematically driven. Rather it comes from human intuition. There might be a better way to do normalization than using face size.
4. Detecting hands in 2 handed signs frames are harder than detecting in 1-handed signs. This is because we built straightforward models from pixel to class label without any hidden variables at all.
5. Need better method to detect bounding boxes from pixel classification scores.
6. No tracking method applied

To improve the methods or trying to alleviate the mentioned problems, we can try the following

1. Providing that we have enough training data, we might want to replace random forest model with neural network as, in theory, would work better. In addition, using neural network might alleviate the need of features engineering.
2. To address the problem of scaling difference, instead of trying to find the global normalized factor, we can 1) try to find local normalization factor depending on pixels or other local features. 2) Not doing any normalization at all, instead try detecting on different scales.
3. The problem of noisy background is a difficult one to solve. Where, one can argue that sophisticated model such as deep learning might be able to do features selection and only consider the relevant features, in practice, it is very hard to overcome without tremendous number of data points. Some solutions that I

can come up with including using reliable segmentation algorithm to segment background from foreground. However, the problem of segmentation itself is known as a hard problem. Second option is using of local features that should be able to help with noisy data.

4. We can introduce hidden variables to help with the case of 2-handed signs. The straightforward way is changing from random forest to fern. As fern is based on generative model, introducing a hidden variable would not be too difficult.
5. Apply some tracking method such as condensation on top of the detection result.
6. Use a smarter way such as learning the bounding box size and location from the pixel classification scores instead of averaging the area.

CHAPTER 7

Discussion and Conclusions

We have proposed a sign language recognition system with focus on hands detection problem. The usual steps of solving the problem is

1. Finding hands or other vital body parts such as elbow, face or other parts. This is called interest region.
2. Extract features vectors from the interest regions. The popular features includes gradient and motion information vectors such as Histogram of Gradients and Optical Flow.
3. Do a classification based on time series model such as Hidden Markov Model, Conditional Random Field etc.

At the time of this writing, one particular model, neural network, should be noted due to its cheer popularity. With deep learning models, the problem of features extraction, or features engineering, becomes less relevant. Furthermore, it has been proven in other applications such as objects recognition that the model can provide higher accuracy than other models. However, we should note that it is not straightforward to deploy deep learning model on top of sign language recognition due to the following reason, not sufficient amount of training data.

It can be argued that deep learning model can learn features mapping and can provide hidden variables without any human knowledge provided that the number of layers is large enough. As such, features engineering is no longer required. However, the real issue here is that the number of required parameters for a sophisticated deep model must also be significantly large to represent a deep network. This means that

the number of required training data must also grow large as well in order to build a good parameters estimation and avoid overfitting. This huge required training data is not as easy to find in sign language datasets. Unlike many other applications such as, general objects recognition or scene recognition, we usually have a large number of signs (classes) with few number of training per sign. Therefore, applying one single deep learning model onto the dataset will usually get overfitting model, resulting in terrible accuracy.

There are 2 ways to resolve or alleviate the issue. We either reduce the number of required parameters or we acquire more training data. Reducing the number of layers is the quick fix solution but we will lose what makes deep learning model so special comparing to others which are features mapping, hidden variables and the model complexity. Cutting down layers meaning that we usually lose the benefit of the model. As such, features engineering and features selection is still a must to accommodate fewer layers of network. We might be able to take the benefit of deep network by training multiple shallow networks where each one responds for different tasks. For examples, one network is defined for features mapping and the other is defined for recognition.

Generating more training data is another approach to tackle too few training data. This approach can be done by training generative model from the existing data. Then, we can use the generative model to produce more data points. However, if the number of training data for the generative model is too small in the first place, then the generated data will look the same as the existing ones and does not yield any other useful information.

Since using deep learning model is really not applicable to the problem. The author believes that the future of research topics in the area should relate to dynamic features selection. For instances, in some signs, it is more useful to focus on shape

based features such as gradient rather than motion based features such as optical flow. In the author's points of view, this could lead to more significantly improvement in recognition accuracy.

Another drawback of the current system is one-vs-all model. As mentioned previously, in sign language videos, there are thousands of classes. Training thousands of sophisticated models where each represents a class just to do a one-vs-all classification in the end will not yield great results since it does not have any discriminating information to distinguish between classes. This is especially true in sign language dataset due to large number of classes. One way to tackle this problem is to train one large discriminative model representing all classes, for examples, Hidden Conditional Random Field [17]. This will help provide the discriminate information between classes. In addition, it will also improve the speed of recognition since we only need to apply a query video to one model not to all classes models.

There are many more directions where the future research can go. It is very likely that there will be much more training data in the future. Thus, many models which are not applicable today might be able to do tomorrow. This leaves to the interpretation of the new generation of researchers.

REFERENCES

[1] J. Shotton, A. W. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 1297–1304.

[2] P. Buehler, M. Everingham, D. P. Huttenlocher, and A. Zisserman, "Long term arm and hand tracking for continuous sign language TV broadcasts," in *British Machine Vision Conference(BMVC)*, 2008.

[3] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[4] M.-y. Chen and A. Hauptmann, "Mosift: Recognizing human actions in surveillance videos," 2009.

[5] G. Willems, T. Tuytelaars, and L. Van Gool, "An efficient dense and scale-invariant spatio-temporal interest point detector," in *Computer Vision–ECCV 2008*. Springer, 2008, pp. 650–663.

[6] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb

[7] S. R. Eddy, "Hidden markov models," *Current opinion in structural biology*, vol. 6, no. 3, pp. 361–365, 1996.

[8] A. D. Wilson and A. F. Bobick, "Parametric hidden markov models for gesture recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 9, pp. 884–900, 1999.

[9] T. Starner and A. Pentland, "Real-time american sign language recognition from video using hidden markov models," in *Motion-Based Recognition*. Springer, 1997, pp. 227–243.

[10] F.-S. Chen, C.-M. Fu, and C.-L. Huang, "Hand gesture recognition using a real-time tracking method and hidden markov models," *Image and Vision Computing*, vol. 21, no. 8, pp. 745–758, 2003.

[11] R. Yang and S. Sarkar, "Gesture recognition using hidden markov models from fragmented observations," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1. IEEE, 2006, pp. 766–773.

[12] A. Corradini, "Dynamic time warping for off-line recognition of a small gesture vocabulary," in *Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 2001. Proceedings. IEEE ICCV Workshop on*. IEEE, 2001, pp. 82–89.

[13] G. Ten Holt, M. Reinders, and E. Hendriks, "Multi-dimensional dynamic time warping for gesture recognition," in *Thirteenth annual conference of the Advanced School for Computing and Imaging*, vol. 300, 2007.

[14] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff, "A unified framework for gesture recognition and spatiotemporal gesture segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 9, pp. 1685–1699, 2009.

[15] H. Wang, R. Stefan, S. Moradi, V. Athitsos, C. Neidle, and F. Kamangar, "A system for large vocabulary sign search."

[16] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[17] S. B. Wang, A. Quattoni, L.-P. Morency, D. Demirdjian, and T. Darrell, "Hidden conditional random fields for gesture recognition," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 1521–1527.

[18] S. R. Eddy, "Hidden markov models," *Current opinion in structural biology*, vol. 6, no. 3, pp. 361–365, 1996.

[19] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." in *International Conference on Machine Learning (ICML)*, 2001, pp. 282–289.

[20] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[21] Z. Zhang, R. Alonzo, and V. Athitsos, "Experiments with computer vision methods for hand detection," in *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA '11. New York, NY, USA: ACM, 2011, pp. 21:1–21:6. [Online]. Available: http://doi.acm.org/10.1145/2141622.2141648

[22] ——, "Experiments with computer vision methods for hand detection," in *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, 2011, pp. 21:1–21:6.

[23] A. Mittal, A. Zisserman, and P. H. Torr, "Hand detection using multiple proposals." in *BMVC*, 2011, pp. 1–11.

[24] L. Karlinsky, M. Dinerstein, D. Harari, and S. Ullman, "The chains model for detecting parts by their context," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 25–32.

[25] A. Kar, "Skeletal tracking using microsoft kinect," *Methodology*, vol. 1, pp. 1–11, 2010.

[26] V. M. Z. Zhang, W.H. Liu and V. Athitsos, "A viewpoint-independent statistical method for fall detection," in *International Conference on Pattern Recognition*, Nov 2012, pp. 3626–3630.

[27] e. a. Vondrick Carl. (2013, Jan.) Hoggles: Visualizing object detection features. [Online]. Available: http://web.mit.edu/vondrick/ihog/

[28] C. Conly, P. Doliotis, P. Jangyodsuk, R. Alonzo, and V. Athitsos, "Toward a 3d body part detection video dataset and hand tracking benchmark," in *Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA '13. New York, NY, USA: ACM, 2013, pp. 2:1–2:6. [Online]. Available: http://doi.acm.org/10.1145/2504335.2504337

[29] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 24, no. 4, pp. 509–522, 2002.

[30] K. Grauman and T. J. Darrell, "Fast contour matching using approximate earth mover's distance," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004, pp. I: 220–227.

[31] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *IEEE International Conference on Computer Vision (ICCV)*, 2003, pp. 750–757.

[32] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *International Conference on Very Large Databases (VLDB)*, 1999, pp. 518–529.

[33] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006, pp. 459–468.

[34] A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik, "Recognizing objects in range data using regional point descriptors," in *European Conference on Computer Vision*, vol. 3, 2004, pp. 224–237.

[35] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *International Conference on Very Large Databases (VLDB)*, 2007, pp. 950–961.

[36] A. Andoni and P. Indyk, "Efficient algorithms for substring near neighbor problem," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006, pp. 1203–1212.

[37] J. Buhler, "Efficient large-scale sequence comparison by locality-sensitive hashing," *Bioinformatics*, vol. 17, no. 5, 2001.

[38] R. Panigrahy, "Entropy based nearest neighbor search in high dimensions," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006, pp. 1186–1195.

[39] C. Faloutsos and K. I. Lin, "FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *ACM International Conference on Management of Data (SIGMOD)*, 1995, pp. 163–174.

[40] I. Guyon, L. Schomaker, and R. Plamondon, "Unipen project of on-line data exchange and recognizer benchmarks," in *12th International Conference on Pattern Recognition*, 1994, pp. 29–33.

[41] J. B. Kruskall and M. Liberman, "The symmetric time warping algorithm: From continuous to discrete," in *Time Warps*. Addison-Wesley, 1983.

[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[43] V. Athitsos, "Learning embeddings for indexing, retrieval, and classification, with applications to object and shape recognition in image databases," Ph.D. dissertation, Boston University, 2006.

[44] *Poser 5 Reference Manual*, Curious Labs, Santa Cruz, CA, August 2002.

[45] Q. Yuan, S. Sclaroff, and V. Athitsos, "Automatic 2D hand tracking in video sequences." in *IEEE Workshop on Applications of Computer Vision*, 2005, pp. 250–256.

[46] H. Barrow, J. Tenenbaum, R. Bolles, and H. Wolf, "Parametric correspondence and chamfer matching: Two new techniques for image matching," in *International Joint Conference on Artificial Intelligence*, 1977, pp. 659–663.

[47] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios, "Nearest neighbor retrieval using distance-based hashing," in *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ser. ICDE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 327–336. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2008.4497441

[48] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, "Overview of the face recognition grand challenge," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 947–954.

[49] P. Dreuw, C. Neidle, V. Athitsos, S. Sclaroff, and H. Ney, "Benchmark databases for video-based automatic sign language recognition," in *International Conference on Language Resources and Evaluation*, 2008.

[50] A. Torralba, K. P. Murphy, and W. T. Freeman, "Sharing visual features for multiclass and multiview object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 29, no. 5, pp. 854–869, 2007.

[51] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[52] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing multiclass to binary: a unifying approach for margin classifiers," *Journal of Machine Learning Research*, vol. 1, pp. 113–141, 2000.

[53] V. N. Vapnik, *The nature of statistical learning theory.* Springer-Verlag New York, Inc., 1995.

[54] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *International Conference on Very Large Databases (VLDB)*, 1999, pp. 518–529.

[55] A. Stefan, V. Athitsos, Q. Yuan, and S. Sclaroff, "Reducing jointboost-based multiclass classification to proximity search," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[56] C. Böhm, S. Berchtold, and D. A. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322–373, 2001.

[57] G. R. Hjaltason and H. Samet, "Index-driven similarity search in metric spaces," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 4, pp. 517–580, 2003.

[58] I. Jolliffe, *Principal Component Analysis.* Springer-Verlag, 1986.

[59] G. Hjaltason and H. Samet, "Properties of embedding methods for similarity searching in metric spaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 25, no. 5, pp. 530–549, 2003.

[60] C. Domeniconi, J. Peng, and D. Gunopulos, "Locally adaptive metric nearest-neighbor classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 9, pp. 1281–1285, 2002.

[61] L. Wiskott, J.-M. Fellous, N. Krüger, and C. von der Malsburg, "Face recognition by elastic bunch graph matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 775–779, 1997.

[62] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *International Journal of Computer Vision*, vol. 61, no. 1, pp. 55–79, 2005.

[63] D. Ramanan, "Learning to parse images of articulated bodies," in *NIPS*, vol. 1, no. 6, 2006, p. 7.

[64] Y. Yang and D. Ramanan, "Articulated pose estimation with flexible mixtures-of-parts," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 1385–1392.

[65] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[66] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester, "Cascade object detection with deformable part models," in *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*. IEEE, 2010, pp. 2241–2248.

[67] C. Keskin, F. Kiraç, Y. E. Kara, and L. Akarun, "Randomized decision forests for static and dynamic hand shape classification," in *Computer Vision and Pattern*

Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on. IEEE, 2012, pp. 31–36.

[68] P. Kohli, M. Pelillo, and H. Bischof, "Context-sensitive decision forests for object detection," 2012.

[69] J. Gall and V. Lempitsky, "Class-specific hough forests for object detection," in *Decision Forests for Computer Vision and Medical Image Analysis*. Springer, 2013, pp. 143–157.

[70] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Van Gool, "Random forests for real time 3d face analysis," *International Journal of Computer Vision*, vol. 101, no. 3, pp. 437–458, 2013.

[71] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. Fitzgibbon, "Efficient regression of general-activity human poses from depth images," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 415–422.

[72] J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon, "The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 103–110.

[73] J. Charles, T. Pfister, M. Everingham, and A. Zisserman, "Automatic and efficient human pose estimation for sign language videos," *International Journal of Computer Vision*, pp. 1–21, 2011.

[74] "Interactive segmentation tool-box," http://www.cs.cmu.edu/ mohitg/segmentation.htm.

[75] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff, "A unified framework for gesture recognition and spatiotemporal gesture segmentation," *Pattern Analysis*

and Machine Intelligence, IEEE Transactions on, vol. 31, no. 9, pp. 1685–1699, Sept 2009.

[76] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester, "Discriminatively trained deformable part models, release 4," http://people.cs.uchicago.edu/ pff/latent-release4/.

[77] "The gallaudet dictionary of american sign language," http://gupress.gallaudet.edu/bookpage/GDASLbookpage.html.

[78] M. Jones and J. Rehg, "Statistical color models with application to skin detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999, pp. I:274–280.