



THE UNIVERSITY
of ADELAIDE

**Structured Output Prediction and Binary Code
Learning in Computer Vision**

by

Guosheng Lin

A thesis submitted in fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Engineering, Computer and Mathematical Sciences
School of Computer Science

December 2014

Declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I give consent to this copy of my thesis when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library catalogue and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Signed: _____

Date: _____

Acknowledgements

Above all, I would like to express my profound gratitude to my principal supervisor Prof. Chunhua Shen for his continued guidance and strong support throughout my time as his student. He taught me not only fundamental and advanced knowledge of machine learning, but also scientific thinking and writing skills. He gave me insightful guidance and creative suggestions on my research problems. He made himself available to me at any time for discussing research problems. This thesis would not have been possible without his guidance.

I would like to thank my co-supervisor Prof. David Suter. He encouraged me working on my research interests, helped me to develop scientific thinking, inspired me thinking outside the box, and improved my presentation skills. I would also like to thank my co-supervisor Dr. Tat-Jun Chin. His questions raised in meetings helped me deliver clear and convincing presentations of my ideas.

I would like to thank Prof. Anton van den Hengel for his great support and invaluable collaboration. He gave me important suggestions on presentation, and offer great help on paper writing and revision. I am also grateful to Dr. Qinfeng (Javen) Shi. His insightful viewpoints and suggestions helped me improve my research methodologies and paper writing.

Many thanks go to ACVT researchers (current or previous) for their kindness and beneficial discussions: Dr. Xi Li, Dr. Sakrapee (Paul) Paisitkriangkrai, Dr. Peng Wang, Dr. Lingqiao Liu and Dr. Jin Yu. I would also like to thank all of my lab mates, especially Quoc-Huy Tran, Zhenhua Wang, Rui Yao, Fumin Shen and Zhen Zhang.

Finally, my special appreciation is dedicated to my parents, brother, sister for their everlasting support, and to my girlfriend with whom I share my success and frustration.

Publications

This thesis is based on the content of the following peer-reviewed conference and journal papers:

1. Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel and David Suter; “*Fast Supervised Hashing with Decision Trees for High-Dimensional Data*”; In proceeding of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
2. Guosheng Lin, Chunhua Shen, Jianxin Wu; “*Optimizing Ranking Measures for Compact Binary Code Learning*”; In proceeding of European Conference on Computer Vision (ECCV), 2014.
3. Chunhua Shen, Guosheng Lin, Anton van den Hengel; “*StructBoost: Boosting Methods For Predicting Structured Output Variables*”; IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2014.
4. Guosheng Lin, Chunhua Shen, David Suter and Anton van den Hengel; “*A General Two-Step Approach to Learning-Based Hashing*”; In proceeding of International Conference on Computer Vision (ICCV), 2013.
5. Xi Li*, Guosheng Lin*, Chunhua Shen, Anton van den Hengel and Anthony Dick (*** indicates equal contribution**); “*Learning Hash Functions Using Column Generation*”; In proceeding of International Conference on Machine Learning (ICML), 2013.
6. Guosheng Lin, Chunhua Shen and Anton van den Hengel; “*Approximate Constraint Generation for Efficient Structured Boosting*”; In proceeding of International Conference on Image Processing (ICIP), 2013.
7. Guosheng Lin, Chunhua Shen, Anton van den Hengel and David Suter; “*Fast Training of Effective Multi-class Boosting Using Coordinate Descent Optimization*”; In proceeding of Asian Conference on Computer Vision (ACCV), 2012.

Abstract

Faculty of Engineering, Computer and Mathematical Sciences
School of Computer Science

Doctor of Philosophy

by Guosheng Lin

Machine learning techniques play essential roles in many computer vision applications. This thesis is dedicated to two types of machine learning techniques which are important to computer vision: structured learning and binary code learning. Structured learning is for predicting complex structured output of which the components are inter-dependent. Structured outputs are common in real-world applications. The image segmentation mask is an example of structured output. Binary code learning is to learn hash functions that map data points into binary codes. The binary code representation is popular for large-scale similarity search, indexing and storage. This thesis has made practical and theoretical contributions to these two types of learning techniques.

The first part of this thesis focuses on boosting based structured output prediction. Boosting is a type of methods for learning a single accurate predictor by linearly combining a set of less accurate weak learners. As a special case of structured learning, we first propose an efficient boosting method for multi-class classification, which can be applied to image classification. Different from many existing multi-class boosting methods, we train class specified weak learners by separately learning weak learners for each class. We also develop a fast coordinate descent method for solving the optimization problem, in which we have closed-form solution for each coordinate update.

For general structured output prediction, we propose a new boosting based method, which we refer to as StructBoost. StructBoost supports nonlinear structured learning by combining a set of weak structured learners. Our StructBoost generalizes standard boosting approaches such as AdaBoost, or LPBoost to structured learning. The resulting optimization problem is challenging in the sense that it may involve exponentially many variables and constraints. We develop cutting plane and column generation based algorithms to efficiently solve the optimization. We show the versatility and usefulness of StructBoost on a range of problems such as optimizing the tree loss for hierarchical multi-class classification, optimizing the Pascal overlap criterion for robust visual tracking and learning conditional random field parameters for image segmentation.

The last part of this thesis focuses on hashing methods for binary code learning. We develop three novel hashing methods which focus on different aspects of binary code learning. We first present a column generation based hash function learning method for preserving triplet based relative pairwise similarity. Given a set of triplets that encode the pairwise similarity comparison information, our method learns hash functions within the large-margin learning framework. At each iteration of the column generation procedure, the best hash function is selected. We show that our method with triplet based formulation and large-margin learning is able to learn high quality hash functions.

The second hashing learning method in this thesis is a flexible and general method with a two-step learning scheme. Most existing approaches to hashing apply a single form of hash function, and an optimization process which is typically deeply coupled to this specific form. This tight coupling restricts the flexibility of the method to respond to the data, and can result in complex optimization problems that are difficult to solve. In this chapter we propose a flexible yet simple framework that is able to accommodate different types of loss functions and hash functions. This framework allows a number of existing approaches to hashing to be placed in context, and simplifies the development of new problem-specific hashing methods. Our framework decomposes the hashing learning problem into two steps: hash bit learning and hash function learning based on the learned bits. The first step can typically be formulated as binary quadratic problems, and the second step can be accomplished by training standard binary classifiers. These two steps can be easily solved by leveraging sophisticated algorithms in the literature.

The third hashing learning method aims for efficient and effective hash function learning on large-scale and high-dimensional data, which is an extension of our general two-step hashing method. Non-linear hash functions have demonstrated their advantage over linear ones due to their powerful generalization capability. In the literature, kernel functions are typically used to achieve non-linearity in hashing, which achieve encouraging retrieval performance at the price of slow evaluation and training time. We propose to use boosted decision trees for achieving non-linearity in hashing, which are fast to train and evaluate, hence more suitable for hashing with high dimensional data. In our approach, we first propose sub-modular formulations for the hashing binary code inference problem and an efficient GraphCut based block search method for solving large-scale inference. Then we learn hash functions by training boosted decision trees to fit the binary codes. We show that our method significantly outperforms most existing methods both in retrieval precision and training time, especially for high-dimensional data.

Dedicated to my family.

Contents

Declaration	iii
Acknowledgements	v
Publications	vii
Abstract	ix
Contents	xiii
List of Figures	xvii
List of Tables	xxi
1 Introduction	1
1.1 Structured Learning	1
1.2 Binary Code Learning	3
1.3 Contributions	6
2 Background Literature	9
2.1 Structured Learning	9
2.1.1 Structured SVM	10
2.2 Boosting	12
2.2.1 Column generation boosting	12
2.2.2 Column generation for multi-class boosting	15
2.2.2.1 MultiBoost with hinge loss	16
2.2.2.2 MultiBoost with exponential loss	17
2.3 Binary Code Learning	17
2.3.1 Locality-sensitive hashing	19
2.3.2 Spectral hashing	20
2.3.3 Self-taught hashing	21
2.3.4 Supervised hashing with kernel	24

3	Fast Training of Effective Multi-class Boosting Using Coordinate Descent	29
3.1	Introduction	29
3.1.1	Main Contributions	31
3.1.2	Notation	31
3.2	The Proposed Method	32
3.2.1	Column generation for MultiBoost _{cw}	33
3.2.2	Fast coordinate descent	35
3.3	Experiments	40
3.3.1	UCI datasets	41
3.3.2	Handwritten digit recognition	42
3.3.3	Three Image datasets: PASCAL07, LabelMe, CIFAR10	43
3.3.4	Scene recognition	43
3.3.5	Traffic sign recognition	44
3.3.6	FCD evaluation	44
3.4	Conclusion	45
4	StructBoost: Boosting Methods for Predicting Structured Output Variables	49
4.1	Introduction	50
4.1.1	Main contributions	51
4.1.2	Related work	51
4.1.3	Notation	53
4.2	Structured boosting	54
4.2.1	1-slack formulation for fast optimization	56
4.2.2	Cutting-plane optimization for the 1-slack primal	58
4.2.3	Discussion	60
4.3	Examples of StructBoost	60
4.3.1	Binary classification	60
4.3.2	Ordinal regression and AUC optimization	61
4.3.3	Multi-class boosting	61
4.3.4	Hierarchical classification with taxonomies	62
4.3.5	Optimization of the Pascal image overlap criterion	64
4.3.6	CRF parameter learning	65
4.4	Experiments	68
4.4.1	AUC optimization	69
4.4.2	Multi-class classification	70
4.4.3	Hierarchical multi-class classification	72
4.4.4	Visual tracking	73
4.4.5	CRF parameter learning for image segmentation	76
4.5	Conclusion	83
5	Learning Hash Functions Using Column Generation	85
5.1	Introduction	85
5.1.1	main contribution	86
5.1.2	Related work	87
5.1.3	Notation	88

5.2	The proposed method	88
5.2.1	Learning hash functions with squared hinge loss	89
5.3	Hashing with general smooth convex loss functions	93
5.3.1	Hashing with logistic loss	95
5.4	Hashing with ℓ_∞ norm regularization	96
5.5	Extension of regularization	97
5.6	Experiments	98
5.6.1	Dataset description	98
5.6.2	Experiment setup	102
5.6.3	Competing methods	102
5.6.4	Evaluation criteria	102
5.6.5	Quantitative comparison results	103
5.7	Conclusion	103
6	A General Two-Step Approach to Learning-Based Hashing	105
6.1	Introduction	105
6.2	Two-Step Hashing	107
6.3	Solving binary quadratic problems	109
6.4	Experiments	122
6.4.1	Comparing methods	122
6.4.2	Dataset description	123
6.4.3	Evaluation measures	124
6.4.4	Using different loss functions	124
6.4.5	Training time	125
6.4.6	Using different hash functions	125
6.4.7	Results on large datasets	126
6.5	Conclusion	126
7	Fast Supervised Hashing with Decision Trees for High-Dimensional Data	127
7.1	Introduction	128
7.2	The proposed method	129
7.2.1	Step 1: Binary code inference	132
7.2.2	Step 2: Learning boosted trees as hash functions	135
7.3	Experiments	136
7.3.1	Comparison with KSH	140
7.3.2	Comparison with TSH	142
7.3.3	Experiments on different features	145
7.3.4	Comparison with dimension reduction	147
7.3.5	Comparison with unsupervised methods	148
7.3.6	More features and more bits	149
7.3.7	Large dataset: SUN397	150
7.4	Conclusion	151
8	Conclusion	153
8.1	Future work	154
A	Appendix for MultiBoost_{cw}	157

A.1	Dual problem of $\text{MultiBoost}_{\text{cw}}$	157
A.2	$\text{MultiBoost}_{\text{cw}}$ with the hinge loss	158
B	Appendix for StructBoost	161
B.1	Dual formulation of n -slack	161
B.2	Dual formulation of 1-slack	162
C	Appendix for CGHash	165
C.1	Learning hashing functions with the hinge loss	165
C.1.1	Using ℓ_1 norm regularization	165
C.1.2	Using l_∞ norm regularization	166
	Bibliography	169

List of Figures

1.1	An example of the image segmentation task. The first row is the input images and the second row is the segmentation label masks. The label mask is the structured output that we aim to predict, which identifies target objects (cars here) from the background.	2
1.2	An illustration of hashing based similarity preserving	4
1.3	An illustration of image retrieval. The first column shows query images, and the rest are retrieved images in the database. These retrieved images are expected to be semantically similar to the corresponding query images.	4
3.1	Results on 2 UCI datasets: VOWEL and ISOLET. CW and CW-1 are our methods. CW-1 uses stage-wise setting. The number after the method name is the mean value with standard deviation of the last iteration. Our methods converge much faster and achieve competitive test accuracy. The total training time and the solver time of our methods both are less than that of MultiBoost [1].	41
3.2	Experiments on 2 handwritten digit recognition datasets: USPS, PENDIG-ITS. CW and CW-1 are our methods. CW-1 uses stage-wise setting. Our methods converge much faster, achieve best test error and use less training time. Ada.MO has similar convergence rate as ours, but requires much more training time. With a maximum training time of 1000 seconds, Ada.MO failed to finish 500 iterations on all datasets.	42
3.3	Experiments on handwritten digit recognition datasets: MNIST. CW and CW-1 are our methods. CW-1 uses stage-wise setting. Our methods converge much faster, achieve best test error and use less training time. Ada.MO has similar convergence rate as ours, but requires much more training time. With a maximum training time of 1000 seconds, Ada.MO failed to finish 500 iterations on this dataset.	43
3.4	Results on a traffic sign dataset: GTSRB. CW and CW-1 (stage-wise setting) are our methods. Our methods converge much faster, achieve best test error and use less training time.	44
3.5	Experiments on 3 image datasets: PASCAL07, LabelMe and CIFAR10. CW and CW-1 are our methods. CW-1 uses stage-wise setting. Our methods converge much faster, achieve best test error and use less training time.	46
3.6	Experiments on 2 scene recognition datasets: SCENE15 and a subset of SUN. CW and CW-1 are our methods. CW-1 uses stage-wise setting. Our methods converge much faster and achieve best test error and use less training time.	47

3.7	Solver comparison between FCD with different parameter setting and LBFGS-B [2]. One column for one dataset. The number after “FCD” is the setting for the maximum iteration (τ_{max}) of FCD. The stage-wise setting of FCD is the fastest one. See the text for details.	48
4.1	The hierarchy structures of two selected subsets of the SUN dataset [3] used in our experiments for hierarchical image classification.	63
4.2	Classification with taxonomies (tree loss), corresponding to the first example in Figure 4.1.	64
4.3	AUC optimization on two UCI datasets. The objective values and optimization time are shown in the figure by varying boosting (or column generation) iterations. It shows that 1-slack achieves similar objective values as n -slack but needs less running time.	70
4.4	Test performance versus the number of boosting iterations of multi-class classification. StBoost-stump and StBoost-per denote our StructBoost using decision stumps and perceptrons as weak learners, respectively. The results of SSVM and SVM are shown as straight lines in the plots. The values shown in the legend are the error rates of the final iteration for each method. Our methods perform better than SSVM in most cases. . .	71
4.5	Bounding box overlap in frames of several video sequences. Our StructBoost often achieves higher scores of box overlap compared with other trackers.	75
4.6	Bounding box center location error in frames of several video sequences. Our StructBoost often achieves lower center location errors compared with other trackers.	76
4.7	Some tracking examples for several video sequences: “coke”, “david”, , “walk”, “bird” and “tiger2” (best viewed on screen). The output bounding boxes of our StructBoost better overlap against the ground truth than the compared methods.	77
4.8	Some segmentation results on the Graz-02 dataset (car). Compared with AdaBoost, structured output learning methods (StructBoost and SSVM) present sharper segmentation boundaries, and better spatial regularization. Compared with SSVM, our StructBoost with non-linear parameter learning performs better, demonstrating more accurate foreground object boundaries and cleaner backgrounds.	78
4.9	Some segmentation results on the Graz-02 dataset (bicycle). Compared with AdaBoost, structured output learning methods (StructBoost and SSVM) present sharper segmentation boundaries, and better spatial regularization. Compared with SSVM, our StructBoost with non-linear parameter learning performs better, demonstrating more accurate foreground object boundaries and cleaner backgrounds.	79
4.10	Some segmentation results on the Graz-02 dataset (person). Compared with AdaBoost, structured output learning methods (StructBoost and SSVM) present sharper segmentation boundaries, and better spatial regularization. Compared with SSVM, our StructBoost with non-linear parameter learning performs better, demonstrating more accurate foreground object boundaries and cleaner backgrounds.	80

4.11	Some segmentation results on the Graz-02 dataset (person). Compared with AdaBoost, structured output learning methods (StructBoost and SSVM) present sharper segmentation boundaries, and better spatial regularization. Compared with SSVM, our StructBoost with non-linear parameter learning performs better, demonstrating more accurate foreground object boundaries and cleaner backgrounds.	81
5.1	Results of precision-recall (using 64 bits). It shows that our CGHash performs the best in most cases.	98
5.2	Precision of top-50 retrieved examples using different number of bits. It shows that our CGHash performs the best in most cases.	99
5.3	Nearest-neighbor classification error using different number of bits. It shows that our CGHash performs the best in most cases.	100
5.4	Performance of CGHash using different values of K ($K \in \{3, 10, 20, 30\}$) on the SCENE-15 dataset. Results of precision-recall (using 60 bits), precision of top-50 retrieved examples and nearest-neighbor classification are shown.	101
5.5	Two retrieval examples for CGHash on the LABELME and MNIST datasets. Query examples are shown in the left column, and the retrieved examples are shown in the right part.	104
6.1	An illustration of Two-Step Hashing	110
6.2	Some retrieval examples of our method TSH on CIFAR10. The first column shows query images, and the rest are top 40 retrieved images in the database. False predictions are marked by red boxes.	115
6.3	Some retrieval examples of our method TSH on CIFAR10. The first column shows query images, and the rest are top 40 retrieved images in the database. False predictions are marked by red boxes.	116
6.4	Results on 2 datasets of supervised methods. Results show that TSH outperforms others usually by a large margin. The running up methods are STHs-RBF and KSH.	117
6.5	Results on 2 datasets for comparing unsupervised methods. Results show that TSH outperforms others usually by a large margin.	118
6.6	Results on SCENE15, USPS and ISOLET for comparing with supervised and unsupervised methods. Our TSH perform the best.	119
6.7	Results on 2 datasets of our method using different hash functions. Results show that using kernel hash function (TSH-RBF and TSH-KF) achieves best performances.	120
6.8	Code compression time using different hash functions. Results show that using kernel transferred feature (TSH-KF) is much faster then SVM with RBF kernel (TSH-RBF). Linear SVM is the fastest one.	121
6.9	Comparison of supervised methods on 2 large scale datasets: Flickr1M and Tiny580k. Our method TSH achieves on par result with KSH. TSH and KSH significantly outperform other methods.	121
6.10	Comparison of unsupervised methods on 2 large scale datasets: Flickr1M and Tiny580k. The first row shows the results of supervised methods and the second row for unsupervised methods. Our method TSH achieves on par result with KSH. TSH and KSH significantly outperform other methods.	122

7.1	Some retrieval examples of our method FastHash on CIFAR10. The first column shows query images, and the rest are retrieved images in the database.	137
7.2	Some retrieval examples of our method FastHash on ESPGAME. The first column shows query images, and the rest are retrieved images in the database. False predictions are marked by red boxes.	138
7.3	Comparison of KSH and our FastHash on all datasets. The precision and recall curves are given in the first two rows. The precision curves of the top 2000 retrieved examples are given on the last 2 rows. The number after “KSH” is the number of support vectors. Both of our FastHash and FastHash-Full outperform KSH by a large margin.	141
7.4	Comparison of various combinations of hash functions and binary inference methods. Note that the proposed FastHash uses decision tree as hash functions. The proposed decision tree hash function performs much better than the linear SVM hash function. Moreover, our FastHash performs much better than TSH when using the same hash function in Step 2.	143
7.5	Results on high-dimensional codebook features. The precision and recall curves are given in the first two rows. The precision curves of the top 2000 retrieved examples are given on the last 2 rows. Both our FastHash and FastHash-Full outperform their comparators by a large margin.	146
7.6	The retrieval precision results of unsupervised methods. Unsupervised methods perform poorly for preserving label based similarity. Our FastHash outperform others by a large margin.	150
7.7	The precision curve of top 2000 retrieved examples on large image dataset SUN397 using 1024 bits. Here we compare with those methods which can be efficiently trained up to 1024 bits on the whole training set. Our FastHash outperforms others by a large margin.	151

List of Tables

4.1	AUC maximization. We compare the performance of n -slack and 1-slack formulations. “–” means that the method is not able to converge within a memory and time limit. We can see that 1-slack can achieve similar AUC results on training and testing data as n -slack while 1-slack is significantly faster than n -slack.	69
4.2	Multi-class classification test errors (%) on several UCI and MNIST datasets. 1-v-a SVM is the one-vs-all SVM. StBoost-stump and StBoost-per denote our StructBoost using decision stumps and perceptrons as weak learners, respectively. StructBoost outperforms SSVM in most cases and achieves competitive performance compared with other multi-class classifiers. . . .	72
4.3	Hierarchical classification. Results of the tree loss and the 1/0 loss (classification error rate) on subsets of the SUN dataset. StructBoost-tree uses the hierarchy class formulation with the tree loss, and StructBoost-flat uses the standard multi-class formulation. StructBoost-tree that minimizes the tree loss performs best.	72
4.4	Average bounding box overlap scores on benchmark videos. Struck ₅₀ [4] is structured SVM tracking with a buffer size of 50. Our StructBoost performs the best in most cases. Struck performs the second best, which confirms the usefulness of structured output learning.	73
4.5	Average center errors on benchmark videos. Struck ₅₀ [4] is structured SVM tracking with a buffer size of 50. We observe similar results as in Table 4.4: Our StructBoost outperforms others on most sequences, and Struck is the second best.	74
4.6	Image segmentation results on the Graz-02 dataset. The results show the the pixel accuracy, intersection-union score (including the foreground and background) and $precision = recall$ value (as in [5]). Our method StructBoost for nonlinear parameter learning performs better than SSVM and other methods.	77
5.1	Summary of the 6 datasets used in the experiments.	102
6.1	Results (using hash codes of 32 bits) of TSH using different loss functions, and a selection of other supervised and unsupervised methods on 3 datasets. The upper part relates the results on training data and the lower on testing data. The results show that Step-1 of our method is able to generate effective binary codes that outperforms those of competing methods on the training data. On the testing data our method also outperforms others by a large margin in most cases.	114

6.2	Training time (in seconds) for TSH using different loss functions, and several other supervised methods on 3 datasets. The value inside a brackets is the time used in the first step for inferring the binary codes. The results show that our method is efficient. Note that the second step of learning the hash functions can be easily parallelised.	114
7.1	Comparison of KSH and our FastHash. KSH results with different number of support vectors. Both of our FastHash and FastHash-Full outperform KSH by a large margin in terms of training time, binary encoding time (Test time) and retrieval precision.	139
7.2	Comparison of TSH and our FastHash for binary code inference in Step 1. The proposed Block GraphCut (Block-GC) achieves much lower objective value and also takes less inference time than the spectral method, and thus performs much better.	140
7.3	Comparison of combination of hash functions and binary inference methods. The proposed decision tree hash function performs much better than linear SVM hash function. Moreover, our FastHash performs much better than TSH when using the same hash function in Step-2.	144
7.4	Comparison of TSH and our FastHash. Results of TSH with the linear SVM and the budgeted RBF kernel [6] hash functions (TSH-BRBF) for the Step-2 are presented. Our FastHash outperforms TSH by a large margin both on training speed and retrieval performance.	145
7.5	Results using two types of features: low-dimensional GIST features and the high-dimensional codebook features. Our FastHash and FastHash-Full outperform the comparators by a large margin on both feature types. In terms of training time, our FastHash is also much faster than others on the high-dimensional codebook features.	147
7.6	Results of methods with dimension reduction. KSH, SPLH and STHs are trained with PCA feature reduction. Our FastHash outperforms others by a large margin on retrieval performance.	148
7.7	Performance of our FastHash on more features (22400 dimensions) and more bits (1024 bits). It shows that FastHash can be efficiently trained on high-dimensional features with large bit length. The training and binary coding time (Test time) of FastHash is only linearly increased with the bit length.	149
7.8	Results on the large image dataset SUN397 using 11200-dimensional codebook features. Our FastHash can be efficiently trained to large bit length (1024 bits) on this large training set. FastHash outperforms other methods by a large margin on retrieval performance.	149