Supervised Image Hashing with Proximal Methods

Cyrus Teng, Ganesh Sivaraman, Varun Manjunatha

December 21, 2014

Abstract

We present a supervised binary encoding scheme for image retrieval that learns projections using a proximal optimization algorithm. We first formulate the problem to minimize reconstruction error between a Gramian matrix formed by binary codes, and a similarity matrix obtained from ground truth labels. We optimize our convex objective using a forward-backward splitting algorithm with a proximal function that forces the reconstruction to get boxed between -1 and 1 for each step of descent. Thus, diverging from other hashing methods, our method produces naturally binary projections. We evaluate our method on the Cifar-10 image retrieval dataset, and obtain results which strongly compete with state-of-the-art hashing methods.

1 Introduction

Given a database of images, image retrieval is the problem of returning images from the database that are most similar to a query. The proliferation of images on the internet due to ubiquity of mobile phone cameras means that performing image retrieval on databases with billions of images is challenging. This is mainly due to linear time complexity of nearest neighbor retrieval algorithms. Image hashing[3, 11, 20, 13, 12, 14] alleviates this problem by obtaining similarity preserving binary codes which represent high dimensional floating point image descriptors, and offers both efficient storage and scalable retrieval with sub-linear search times. These binary hash-codes can be learned in both unsupervised and supervised settings. Unsupervised hashing algorithms map nearby points in a metric space to similar binary codes. On the other hand, supervised hashing algorithms try to preserve semantic label information in the Hamming space. This is illustrated in Figure 1.

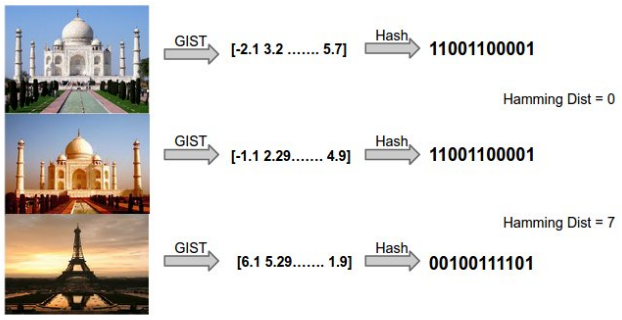

Figure 1: In this illustration, the images belonging to the “Taj Mahal” class have the same binary hash-code, unlike the image belonging to the “Eiffel Tower” class.

Images are often accompanied with a class label - these give the identity of the object in the image, under the assumption that the image contains only one object. The aim of supervised image retrieval is to retrieve images which belong to the same class as that of the query. For example, if the query is the image of a dog, then, it would be desirable that the retrieval algorithm returns other images of dogs. The aim of this work is to transform image features into binary hash codes, which enables such a supervised retrieval.

Supervised image hashing entails learning binary hash codes that best preserve class label information, and this involves solving an optimization problem.

There are many ways of formulating this problem, but we use an approach identical to [13]. Our solution to this optimization problem is unique and gives comparable performance to many state-of-the-art hashing algorithms. The time our algorithm takes in the training phase is a fraction of the time taken by [13]. We use a forward-backward splitting method, taken from [18] which ensures that the codes generated are naturally binary. This is opposed to other approaches which compute real-valued hash projections, and perform a zero-thresholding to convert them to binary.

The contributions of our paper are as follows :

- We learn binary-hash codes on a training sample using a novel forward-backward splitting approach.
- We use a method similar to the kernel trick [16] to construct binary codes for queries.
- We perform image retrieval on queries from the Cifar-10 [8] dataset to evaluate our approach.

In Section 2, we consider related work. In Section 3, we discuss the formulation of the problem and our solution to this problem in detail. In Section 4, we perform experiments to evaluate our method. We conclude in Section 5.

2 Related Work

Work on image hashing can be divided into unsupervised and supervised methods, but for the purpose of brevity, we only mention the latter. Examples of supervised hashing algorithms include [7, 13, 12, 14]. Supervised hashing algorithms are based on objective functions that minimize the difference between hamming distances and similarity of pairs of data points. Liu *et al.* [13] use the class label to determine similarity. A pair of points is considered 'similar' if they belong to the same class and 'dissimilar' otherwise, with similarity value of '1' and '-1' respectively. They utilize a simplified objective function using the relation between hamming distance and inner products of the binary codes. A sequential greedy optimization is adapted to obtain the supervised projections. FastHash [12] also uses a KSH objective function but employs decision trees as hash functions and utilizes a GraphCut based method for binary code inference. Minimal loss hashing [14] uses a structured SVM framework to generate binary codes with an online learning algorithm.

3 Method

The pipeline of our method is as follows :

- We have a dataset of N images, from which we extract d dimensional features like GIST[15], Bag-of-Words[17], Convolutional Neural Networks[9], etc. We split these into N_{db} database images and N_{query} query images.
- Out of the N_{db} database images, we choose N_{train} images to train our hashing algorithm. At the end of this step, we have learned hash functions for each of N_{train} images.
- We designate N_{anc} out of N_{train} points as "anchor points". We use these anchor points to construct hash-codes for the remaining $N_{db} - N_{train}$ database images as well as the N_{query} query images.
- We now perform evaluation of our hashing algorithm using the mean-average-precision (mAP) metric on the N_{query} query images.

3.1 Problem Formulation

Consider N_{train} randomly sampled images out of N_{db} images. Since our method is a supervised hashing approach, we create a matrix S such that :

$$S_{ij} = \begin{cases} +1 & \text{if } i^{th} \text{ and } j^{th} \text{ image belong to same class} \\ -1 & \text{otherwise} \end{cases}$$

It is quite apparent that S is a symmetric matrix of dimension $N_{train} \times N_{train}$. Let X be a $N_{train} \times b$ matrix that contains the binary hash-codes. The i^{th} row of X represents the b bit binary hash-code of the i^{th} training sample. The aim of the training phase is to learn the optimal X. The matrix $G = XX^T$

is a Gramian matrix whose element G_{ij} contains the inner product between the binary hash-code of the i^{th} and j^{th} training sample. Thus, we would like this inner product (similarity product) G_{ij} to reflect the class similarity S_{ij}. This is done using the following formulation (suggested by [13]) :

$$\min f(X) = ||XX^T - bS||_{fro}^2 \tag{1}$$

In the above equation, we are trying to minimize the sum of the squared reconstruction error while trying to force G_{ij} to emulate S_{ij} for all pairs of training samples (hence, Frobenius norm). The matrix S is multiplied with the number of bits b because the elements in S are either -1 or 1, whereas the elements of the Gramian matrix G fall in the interval $[-b, b]$. This objective function is convex for the following reason: we can rewrite $f(X) = \sum_{ij} f_{pair}(x_i, x_j)$, where $f_{pair}(x_i, x_j)$ is the reconstruction error for the ij^{th} pair. If f_{pair} is convex, then so is f, because the sum of convex functions is also convex. Thus, $f_{pair} = ||x_i x_j^T - S_{ij}||^2$ is the composition of a quadratic function (convex, since $x_i x_j^T = x_i I x_j^T$ and I is positive-semidefinite, and also non-decreasing), and the l_2 norm, which is also convex. Thus, by the vector composition rule [1], f_{pair} is a convex function, and it follows that f is also convex.

3.2 Forward-Backward Splitting Algorithm

The optimization problem described in Section 3.1 is solved using an implemention of what is known as forward-backward splitting method which is described in detail in [4, 6]. The particular algorithm is called Fast Adaptive Shrinkage/Thresholding Algorithm (FASTA) [6]. In this section, we provide a brief summary of the general methodology.

The optimization problem has the following form:

$$minimize\ f(x) + g(x) \tag{2}$$

where f is convex and differentiable with a Lipschitz-continuous gradient and g is a lower semicontinuous convex function. Because g is not differentiable, the problem cannot be solved using the general class of gradient descent methods. However, it can be shown [5] that a problem of this type has at least one solution and that its solutions are characterized by the fixed point equation

$$x = prox_{\gamma g}(x - \gamma \nabla f(x)) \tag{3}$$

where $\gamma > 0$ and $prox_{\gamma g} : \mathbb{R}^n \to \mathbb{R}^n$ is the proximal operator of g defined by

$$prox_{\gamma g}(y) = \underset{x}{argmin}\ \gamma g(x) + \frac{1}{2} \parallel x - y \parallel_2^2 \tag{4}$$

Hence, $prox_{\gamma g}$ takes a point y and find a minimizer of g that is not too far from y. Equations (3) suggests an iterative method of solving (2), using the following recursive equation [4]:

$$x_{n+1} = prox_{\gamma_n g}(x_n - \gamma_n \nabla f(x_n)) \tag{5}$$

where $x_n - \gamma_n \nabla f(x_n)$ is a gradient descent at x_n with step size γ_n and $prox_{\gamma_n g}$ takes the result of the gradient descent of f and find a minimizer of g that is close to this point.

The solution x^* of equation (4) must satisfy the optimality condition [6]

$$\gamma \mathcal{G} + (x^* - y) = 0 \tag{6}$$

where $\mathcal{G} \in \partial g(x)$ is a subgradient of g. This implies that

$$prox_{\gamma g}(y) = x^* = y - \gamma \mathcal{G} \tag{7}$$

which shows that the proximal operation is a descent step along a subgradient of g.

Therefore, each iteration in the process described by equation (5) involves a gradient descent on f followed by a descent along a subgradient of g. The gradient descent step is called the forward step and the subgradient descent step is called the backward step and this type of scheme is known as a forward-backward splitting algorithm (FSB).

Convergence of FSB is guaranteed if the step size γ_n satisfies certain stability bounds which depend on the Lipschitz constant L of ∇f. [6]. Because L is seldom known with precision, in practice, one often incorporates a backtracking line search in the algorithm which also guarantees convergence [6].

3.3 Hashing Algorithms

Our hashing algorithm utilizes the FBS method described in the previous section. Our unique approach solves the following problem:

$$minimize \ \parallel XX^T - bS \parallel_{fro}^2 + I_C(X) \qquad (8)$$

where C denotes the set $[-1,1]^{N_{train} \times b}$ and I_C is the indicator function

$$I_C(X) = \begin{cases} 0 & if\ X \in C \\ \infty & otherwise \end{cases}$$

Hence, we replace f in equation (2) with $\parallel XX^T - bS \parallel_{fro}^2$ and g with $I_C(X)$. In our FBS algorithm we thus evaluate the proximal operator

$$prox_{\gamma I_C}(Y) = \underset{X \in C}{argmin} \ \frac{1}{2} \parallel X - Y \parallel_2^2 \qquad (9)$$

The solution to equation (9) is the point in C that is closest to Y. This means that each iteration of our hashing algorithm involves two steps. First, we minimize the distance between our Gramian matrix and the similarity matrix using gradient descent where the gradient of f is

$$\nabla f(X) = 4(XX^T X - SX) \qquad (10)$$

In the second step, we project the result from the first step onto I_C. This step forces each element in our Gramian matrix to take a value between -1 and 1 before the next gradient descent step. These two steps are repeated until the residual falls below a specified tolerance level.

3.4 Out-of-sample Extension

The procedure described in Section 3.3 creates binary hash-codes for only the training samples. For the remaining images, we construct hash-codes using a heuristic method similar to kernel SVMs[16]. We refer to this as our out-of-sample extension. Recall that the matrix X now contains N_{train} binary hash-codes, each of length b bits. We designate N_{anc} out of these N_{train} points as anchor points, and these points are used to perform the out-of-sample extension. Let

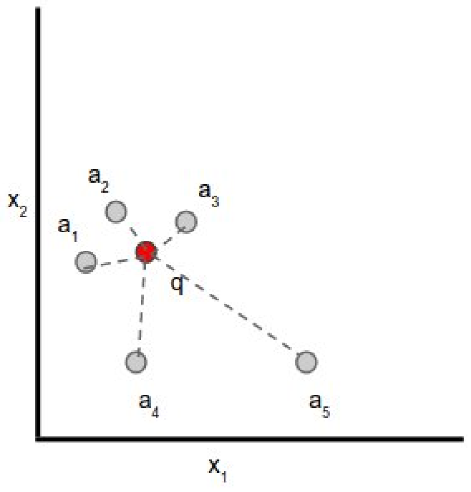

Figure 2: In this toy example, the point q is out of sample and a_i represent anchor points. Obviously, the binary hash code of q should resemble a_1, a_2 and a_3 more closely than a_4 and a_5

the original (GIST) feature for the i^{th} anchor point be a_i. Given a data point q for which we would like to obtain the binary code, we perform the following steps:

1. We compute $w_i = e^{-\frac{(||q-a_i||)^2}{\sigma^2}}$ for $i = 1, 2, ..., N_{anc}$. We fix $\sigma = 0.5$. Clearly, $w_i \in [0, 1]$

2. We now compute x_q (the binary hash-code for q) as $sign(\frac{\sum w_i a_i}{\sum w_i})$

This technique considers the binary hash for a given point q as a linear combination of binary hashes of the anchor points, which have already been computed in the training phase. The weights of this linear combination are dependent on the Euclidean distance between the data point and the anchor points in Euclidean space (Figure 2). Although this creates a computational overhead, the overhead is small as long as the number of anchor points is reasonable ($N_{anc} << N_{db}$).

4 Experiments

4.1 Evaluation metric and Dataset

In Section 3, we formulate an objective function that can be optimized to provide binary hash-codes X for a given similarity matrix S. We perform the optimization using a forward-backward splitting method. Then, we use the out-of-sample extension to produce binary hash-codes for all images in the database and query images. We now perform retrieval experiments using the mean-average-precision (mAP) metric which is defined as : $mAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$, where Q is the number of queries. Here, $AveP = \frac{\sum_{k=1} n(P(k) \times rel(k)}{number\, of\, relevant\, images}$. Intuitively, average-precision for a query image is high if the retrieval system returns relevant images (in our case, belonging to the same class) that are ranked higher than irrelevant images (in our case, belonging to different classes). Mean-average-precision simply computes the mean of average-precision over all query images.

We perform experiments on the Cifar-10 dataset [8], which consists of 60,000 color images of size 32x32 in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck), with 6,000 images per class. There are 50,000 database images and 10,000 query images. We extract GIST [15] features from these images, which is of dimension 512. We investigate the following :

1. How does mAP vary with b, the size of the binary hash-code?

2. How does mAP vary with the number of training examples?

3. How does mAP vary with N_{anc}, the number of anchor points used in the out-of-sample extension?

4.2 Results and Analysis

Table 1 shows mean-average-precision (mAP) for varying number of anchor points (N_{anc}) and length

$b \vert N_{anc}$	300	500	1000	2000	5000
12	0.189	0.187	0.182	0.189	0.193
16	0.186	0.177	0.183	0.198	0.198
24	0.190	0.192	0.194	0.205	0.215
48	0.195	0.196	0.191	0.207	0.209
64	0.195	0.193	0.191	0.207	0.213

Table 1: Comparing mAP with number of bits and number of anchor points

method	12 bits	24 bits	48 bits
LSH[3]	0.1122	0.1245	0.1188
PCAH[19]	0.1368	0.1133	0.1271
SH[20]	0.1330	0.1317	0.1352
KLSH[11]	0.1212	0.1425	0.1602
SSH[19]	0.1514	0.1595	0.1755
LDAH[2]	0.1380	0.1334	0.1267
BRE[10]	0.1817	0.2024	0.2060
MLH[3]	0.1545	0.1932	0.2074
Ours	**0.1814**	**0.2021**	**0.1978**
KSH0[13]	0.1846	0.2047	0.2181
KSH1[13]	0.2325	0.2588	0.2836

Table 2: Comparing mAP with other state of the art methods. 1000 training samples were used for all methods.

N_{train}	100	1000	2000	5000
12 bits	0.163	0.181	0.193	0.182
16 bits	0.180	0.191	0.178	0.183
24 bits	0.179	0.202	0.187	0.193
48 bits	0.180	0.197	0.198	0.191
64 bits	0.182	0.195	0.198	0.191

Table 3: Comparing mAP while varying the number of training examples

of the binary hash code (b). A reasonable conclusion here is that the performance of the algorithm increases with the number of bits, and the number of anchor points. However, as mentioned previously, both of these operations add to runtime of the algorithm. We perform these experiments on 5000 training samples, i.e., $N_{train} = 5000$.

In Table 2, we compare the performance of our method with state of the art methods until 2014. We notice that our method (highlighted in bold) performs competitively with other methods. It is beaten by KSH[13], however, the training time of our method is much shorter than the training time of KSH, which uses a sequential greedy training approach. We have not compared our method with FastHash [12], as their method uses many more training examples (50,000 as compared to our $N_{train} = 5000$). The comparison would not be fair.

In Table 2, we observe that the number of examples that our optimization algorithm is trained on has an impact on retrieval performance. However, the effect of N_{train} is more pronounced from 100 to 1000, than from 1000 to 5000. We can perhaps conclude that the performance of the retrieval system will improve with an order of magnitude increase in number of training samples.

5 Conclusion

In this project, we seek to project image features into a binary embedding space so that images belonging to the same class are nearby in Hamming distance metric. To do this, we formulate an unconstrained convex optimization problem that involved minimizing a reconstruction error. To optimize the objective function, we use a proximal forward-backward splitting algorithm, where the proximal method was approximated by a box constraint, which restricted the projection to a value between -1 and 1. In practice, a majority of the projections took exactly the values of -1 and 1. Therefore, unlike other methods which obtain binary codes by thresholding real valued projections, our method obtains natively binary projections. We also devise an out-of-sample extension, so that data points other than the training points can be projected into binary. Experiments performed on a standard dataset for hashing showed that this algorithm outperformed many state-of-the-art approaches.

References

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[2] M. M. Bronstein C. Strecha, A. M. Bronstein and Pascal Fua. LDAHash: Improved Matching with Smaller Descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1), 2012.

[3] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[4] Patrick L. Combettes and Jean-Christophe Pesquet. *Proximal Splitting Methods in Signal Processing in Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, chapter 10, pages 185–212. Springer, 2011.

[5] Patrick L. Combettes and Valérie R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling and Simulation*, 4(4):1168–1200, 2005.

[6] Tom Goldstein, Christopher Studer, and Richard Baraniuk. A field guide to forward-backward splitting with FASTA implementation. November 2014.

[7] Yunchao Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 817–824, Washington, DC, USA, 2011. IEEE Computer Society.

[8] Alex Krizhevsky. Convolutional deep belief networks on cifar-10, 2010.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012.

[10] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *in Proc. NIPS, 2009*, pages 1042–1050.

[11] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[12] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'14)*, Columbus, Ohio, USA, 2014.

[13] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[14] Mohammad Norouzi and David Fleet. Minimal loss hashing for compact binary codes. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 353–360, New York, NY, USA, June 2011. ACM.

[15] Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42:145–175, 2001.

[16] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[17] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477, October 2003.

[18] Richard Baraniuk Tom Goldstein, Ernie Esser.

[19] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for large scale search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2012.

[20] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.