Learning Efficient Binary Representation for Images with Unsupervised Deep Neural Networks

by

Fangrui Liu

B.Eng., Beijing University of Technology, 2019

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE COLLEGE OF GRADUATE STUDIES

(Electrical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

June 2021

© Fangrui Liu, 2021
The following individuals certify that they have read, and recommend to the College of Graduate Studies for acceptance, a thesis/dissertation entitled:

**Learning Efficient Binary Representation for Images with Unsupervised Deep Neural Networks**

submitted by Fangrui Liu in partial fulfilment of the requirements of the degree of Master of Applied Science

Zheng Liu, School of Engineering
Supervisor

Homayoun Najjaran, School of Engineering
Supervisory Committee Member

Chen Feng, School of Engineering
Supervisory Committee Member

Julian Cheng, School of Engineering
University Examiner
Abstract

Coding deficiency, which refers to information insufficiency a code can carry, is one of the barriers to high-performance representation learning. Unsupervised binary representations have broader applications than other representations but suffer from the same problem. This work addresses the coding deficiency from two perspectives: biases on single binary neurons and correlation between pairs. A normalization layer and a mutual information loss are introduced to encourage lower code bias and less conflict when learning unsupervised hash for images. Learning uniform distribution for binary neurons is crucial to keep every learned bit informative, which motivates the proposed normalized binary layer. Experiments suggest that the proposed normalization can enhance the code quality by having lower biases, especially in small code lengths. Also, a mutual information loss on individual stochastic binary neurons is proposed to reduce the correlation between binary neurons, discouraging code conflict by minimizing mutual information on the learned binary representation and diverging the code distribution before optimizing it in the next epoch. Performance benchmarks on image retrieval with the unsupervised binary code is conducted on four open datasets. Both the proposed approaches help the model to achieve state-of-the-art accuracy on image retrieval task for all those datasets, which validates their effectiveness in improving unsupervised hashing efficiency.
Lay Summary

Hashing is an important technique for searching. However, traditional hashing is unable to search for perceptual artefacts in high dimensional data like images. Recent research works suggest unsupervised neural networks can learn a hashing function without any label, turning the quick search on images into reality. Preserving more information on the learned hash code is one of the key research topics in this area. This work aims at encouraging more information kept from two aspects, learning balanced and independent binary neurons. Experiments conducted in this work convinced that the proposed methods increased the performance by nearly 10%, which means more similar images can be retrieved by nearest neighbours using the hash code.
# Table of Contents

Abstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii

Lay Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iv

Table of Contents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v

List of Tables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . viii

List of Figures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ix

Acknowledgements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . x

Dedication . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xi

## Chapter 1: Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.3 Research Gaps and Objectives . . . . . . . . . . . . . . . . . . . . . . 6

1.4 Thesis Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

## Chapter 2: Literature Review . . . . . . . . . . . . . . . . . . . . . . . . 9

2.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.2 Binary Representation Learning . . . . . . . . . . . . . . . . . . . . . . 9

2.2.1 Supervised Binary Representation Learning . . . . . . . . . . . . . . 10

2.2.2 Unsupervised Binary Representation Learning . . . . . . . . . . . . . 10
# TABLE OF CONTENTS

2.3 Improving Representation Quality Implicitly .......................... 13
  2.3.1 Classification: Learn Class-wise Discriminative Feature  ...... 13
  2.3.2 Autoencoders: Replicate Identical Distribution ................. 13
  2.3.3 Unsupervised Methods: Keep Sample’s Individuality .......... 14
2.4 Learning Representation with External Constraints ................. 15
2.5 Summary ............................................................... 16

Chapter 3: Maximizing Self Information with Normalization ............ 17
  3.1 Centering Distribution for Binary Neurons Keeps Information .... 17
  3.2 Normalized Binary Hash Layer ...................................... 18
    3.2.1 Learning Balanced Binary Neurons ............................. 18
    3.2.2 Normalizing Distribution with Consistency Constraint ....... 21
    3.2.3 Estimating Expected Value with Momentum .................. 23
  3.3 Experiments .......................................................... 24
    3.3.1 Datasets .......................................................... 24
    3.3.2 Experimental Setup ............................................. 25
    3.3.3 Evaluation Metrics .............................................. 26
    3.3.4 Qualitative and Quantitative Results .......................... 26
  3.4 Summary ............................................................... 29

Chapter 4: Minimizing Mutual Information for Unsupervised Hash ....... 32
  4.1 Discourage Correlation for Binary Neurons Avoids Code Conflicts .. 32
  4.2 Bit-wise Mutual Information Loss ................................... 33
    4.2.1 Convexity on Mutual Information .............................. 33
    4.2.2 Approximated Gradient for Joint Probability ................. 35
    4.2.3 ε-Convergence of Joint Probability Approximation .......... 39
    4.2.4 Scatter and Learn .............................................. 41
  4.3 Experiments .......................................................... 43
    4.3.1 Datasets .......................................................... 44
## TABLE OF CONTENTS

4.3.2 Experimental Setup ........................................ 45
4.3.3 Evaluation Metrics .......................................... 45
4.3.4 Evaluation Results .......................................... 47
4.3.5 Empirical Analysis ........................................... 49
4.4 Summary .......................................................... 55

Chapter 5: Conclusions ............................................. 57
  5.1 Summary of Contributions .................................... 57
  5.2 Limitations and Future Study ................................ 58

Bibliography ........................................................ 60
List of Tables

Table 3.1  MAP Evaluation on MNIST and SVHN .......................... 26
Table 3.2  MAP Evaluation on CIFAR-10 ................................. 29
Table 4.1  Table of Notation ................................................. 32
Table 4.2  MAP Evaluation with Unsupervised Binary Representation ... 47
Table 4.3  MAP Evaluation with Unsupervised Binary Representation ... 47
Table 4.4  MAP Results on Models Trained with Different $\beta$ ............ 49
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Difference between Byte Hashing and Semantic Hashing</td>
<td>2</td>
</tr>
<tr>
<td>3.1</td>
<td>Dividing a Continuous Neuron into Binary. Gray areas refer to the positives.</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Retrieved Images on Different Hash Length with MNIST</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>t-SNE Visualization on Different Hash Length with MNIST</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>Comparison on Code Quality with Normalization. Baseline is trained with batch size 32 while the proposed is trained with batch size 512.</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Precision-Recall Curves of Ours and Compared Methods on CIFAR-10 Dataset</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>t-SNE Visualization on CIFAR-10 Dataset</td>
<td>50</td>
</tr>
<tr>
<td>4.3</td>
<td>Retrieved Images on Right with Query Image on Left from NUS-Wide with 16 Bits Embedding</td>
<td>51</td>
</tr>
<tr>
<td>4.4</td>
<td>Precision-Recall Curve with Different $\beta$ on 16 Bits</td>
<td>51</td>
</tr>
<tr>
<td>4.5</td>
<td>Binary Space Utilization under Different $\beta$</td>
<td>52</td>
</tr>
<tr>
<td>4.6</td>
<td>Mean Average Precision on Different Noise Rate</td>
<td>53</td>
</tr>
<tr>
<td>4.7</td>
<td>Minimizing Mutual Information over Binary Embedding. This figure demonstrate the effect of minimizing mutual information with the proposed method.</td>
<td>55</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to thank Dr Zheng Liu for his devoted support to my thesis and my Master’s program. I also would like to appreciate TerraSense Analytics Ltd, Canada and MITACS Accelerate Program for their generous help and funding. I would also like to thank the two courses I took, Stochastic Process by Professor Dr Paramjit Gill and Convex Optimization by Professor Dr Heinz Bauschke. Those two courses I took really helped my research and writing a lot. I would also appreciate valuable advice from Hongguang and Junchi in Intelligent Sensing Diagnostic and Prognostic Research Laboratory at University of British Columbia Okanagan.
Dedication

I will not be able to finish my master program without support from my family and my loved partner, who has always stayed with me during these two years. Though this journey is challenging, I still enjoyed it quite much with the love you gave to me all the time.

To my beloved Jing, and my loved parents.
Chapter 1

Introduction

1.1 Motivation

Digital images are one of the major media we use over the Internet. There are nearly 657 billion images uploaded every year. Conventional byte hashing functions interpret bytes into a unique index as shown in Figure 1.1 (a), which exaggerates the difference in raw text for fast retrieval and matching in search engines. However, hashing an image as bytes does not provide any benefit, as modern search engines are eager to search semantically related images instead of the exact same images. As shown in Figure 1.1 (b), semantic hashing intends to find hashes with perceptual interpretation, matching semantically similar images even they are far in byte distance. Accurate semantic hash plays an important role in real applications. Conventional database systems are unable to index and retrieve high dimensional data concerning its semantic meaning. Or one can annotate the semantic label to every element in the database, but which is nearly impossible in practice. Hashing with automated perceptual interpretation can accelerate the whole process at no cost of human labour. It will speed up the commercial applications such as semantic search engine and recommendation system with considerable recall rate preserved. For most cases, image hashing refers to find semantic hashes for images, so do this thesis. Precise semantic hashing could achieve high performance on image searching, enabling quick search on image contents with large databases, but with the prior of precise perceptual interpretation.

Finding proper perceptual interpretation for semantic hashing is hard as it requires a deeper understanding beyond patterns and textures. Besides, convolutional neural net-
1.1. Motivation

![Diagram](image)

Figure 1.1: Difference between Byte Hashing and Semantic Hashing

works are capable of interpreting intricate information into semantic representations to facilitate recognition or discrimination of high dimensional inputs [29]. More research works have already suggested that deep neural network can understand objects like humans, making them capable of doing quick recognition of objects [19, 23, 27, 60, 68] with previously learned patterns. Those advances in computer vision could improve semantic hashing quality with better perceptual interpretation.

However, abstracting semantic hash with even good perceptual interpretation is still a challenging task. Learning coding with limited code length acquires more sparsity and capacity, demanding sufficient margin to distinguish samples at the perceptual level. Recent research works have demonstrated that neural networks are effective in finding
1.1. Motivation

compact representation for data like images and videos [2, 34, 72] with limited discrete code. Some researchers even push this task further, managing to extract binary representation for high dimensional data which is also known as deep data hashing [24, 48]. The result is good but still cannot satisfy high-performance semantic hashing for high-quality image search, which attracts interests to more efficient semantic hashing.

With the demand for efficient image retrieval, finding semantic hash with deep neural networks, which is also called binary representation learning, is a popular topic in academia [35, 42, 70, 74, 76, 82]. Binaries are the native form of data in digital systems as it is robust and sparse during computation and transmission. Also, it reduces the size and complexity while keeping dimensionality which affects its representativity the most. Research works have proved that even with embarrassingly small binary representations, the neural network can still find universal embeddings that discriminate samples over different categories [3, 4].

Representation learned with class labels are cheap as discriminative representations can also be obtained implicitly with fully supervised classification tasks so can the binary representations. Regarding millions of images on the Internet, learning universal binary representation with annotations is impractical in the real world. Therefore, unsupervised learning frameworks, which has broader application with less cost on annotation, began to attract attention in semantic hashing community.

Unsupervised binary representation for images, also known as unsupervised image hash, is the main research topic in this thesis. Binary representations boost the speed of semantic image retrieval with competitive performance to continuous code and more sparsity in binaries. Even simple metrics like Hamming distance is effective to retrieve similar images in the domain with binary representations. Other applications like cross-modal retrieval [35, 79], face identification [70] have proven that binary code is sufficiently informative to distinguish samples in large domains. On the other hand, unsupervised binary representations are ideal for unknown inputs with no annotation. No prior knowledge like bounding boxes and labels is needed to train an unsupervised
1.2. Problem Statement

binary hashing algorithm. The learned unsupervised binary representation is capable of carrying precise semantic meaning in application like multimedia retrieval [75, 77] without concern on annotations. In conclusion, across different domains, an unsupervised hash can be widely utilized in applications that require compact representation for unlabeled data.

Considerable progress has also been made on research works on finding unsupervised semantic hashing for images. The concept of ‘similar’ and ‘dissimilar’ is widely introduced in this topic to recapture information effectively without any supervision [65, 78]. Metrics like cosine similarity is applied to preserve the similarity of the binarized representation from the feature holds for the original data [67]. Other constraints are also used to minimize the error between binary code and the learned continuous representation [30].

Most research works on unsupervised semantic hashing focus on replicating or improving high-quality representations extracted from well-researched tasks such as classification and object detection. However, the representation obtained in pretext tasks is not the only factor that affects the code quality, but the hashing function itself should also be regularized at the same time. It is argued that current semantic hashing functions have no such proper regularization that ensures code quality, which refers to the coding efficiency for binary representation learning.

1.2 Problem Statement

Coding efficiency, which describes the degree of information preserved in limited binary code, is the main challenge for unsupervised binary representation learning. As the binary code is compact and sparse, the lower dimensional binary space has fewer keys in the code space to interpret information for the full domain. It will drastically reduce the expressiveness by shrinking the value set from the real numbers to binaries. Binary code might be less informative if learned without proper constraints. Those
1.2. Problem Statement

deficiencies may come from biases on single-bit binary code and correlations among pairs of binary code. Self-information entropy and mutual information entropy are two critical metrics that measure the amount of information contained by stochastic variables, especially convenient for binaries. Those two entropies from information theory considers distribution biases and correlations on probability distributions, inspiring the core components in this research. Therefore, this thesis intends to improve code efficiency from two perspectives motivated by information theory: biases and correlation among binary neurons, referring to maximizing self entropy and minimizing mutual information respectively.

Uniformly distributed variable obtains the most information preserved among all other distributions. More specifically, a single binary neuron can obey a uniform distribution with a simple normalization constant, suggesting the random variable should be normalized to the centroid before being mapped to binary to preserve the most information in the input domain.

Furthermore, it is argued that the conventional unsupervised hashing suffers from inefficient use of the code space, which harms the performance of distinguishing samples in the domain. Conflicts in code space would cause ambiguity on identifying samples from others. Current methods [48, 67, 78] only apply loss that maximizes the similarity and consistency of feature, aligning hidden code and the hashed code, which might push solutions into local optimums. It will cause conflicts where samples should be discriminated against but share the same or similar binary code during optimization. Therefore, proper relaxations need to be proposed to encourage more information preserved while eliminating correlation between every bit pairs, which helps the network to jump out of the local optimum with better sparsity in the binary coding spaces.
1.3 Research Gaps and Objectives

Unsupervised binary representation learning for images, also called unsupervised image hashing, intends to obtain a semantic hashing function for quick image retrieval with compact binary hashes. Recent research works on image hashing gives promising results but only tries to replicate the spatial distribution from pretext tasks. However, all those constraints do not consider the effect of code biases and correlations, which are the major barrier during optimization to hashing functions. Similarity learning is broadly introduced in unsupervised binary representation learning. The similarity loss will regress the distance for discrete pairs in binary space, which might lead to poor code efficiency as the solutions are trapped at a local optimum. To compensate for the defect brought by current objective functions, proper constraints on coding efficiency need to be involved to relax the solution from the local optimum.

This thesis will manage to improve hash quality with unsupervised deep neural networks. To be more precise, improving code efficiency on unsupervised binary representation will be the main focus of this thesis, which will be achieved from two perspectives: maximizing self-entropy and minimizing mutual information. Entropy measures information that a random variable carries. As stated above, a solution guided by only simple loss in similarity and consistency may result in biased hashing with low code efficiency. Maximizing entropy could encourage higher code efficiency over the learning domain. On the other hand, mutual information can evaluate the correlation between random variables with joint distributions. Minimizing mutual information would encourage learned representation to diverge as bit-wisely identical independent variables. Works are using mutual information to evaluate class-wise distance with binary representations [3, 4]. However, the joint distribution is incompatible with differential learning functions, so approximation on gradients needs to be introduced to maintain an end-to-end training scheme.

Information theory provides a sufficient basis for the analysis of binary representations. Both self entropy and mutual information are good criteria for binary represen-
1.4 Thesis Organization

Coding efficiency is the main thread of this thesis and will be carried in every chapter. This thesis will first summarize the recent research works on unsupervised binary representation learning. There is a blank on external constraints on coding efficiency, which this thesis will propose. To fill the research gap, we propose a normalized binary layer and a mutual information loss in Chapter 3 and Chapter 4 to discourage code biases on single-bit binary code and the correlations among pairs of those respectively. Each proposed method are verified with experiments on open datasets with common metrics on image hashing. Those two proposed methods can effectively tackle coding efficiency from two aspects: the assumption on code biases that harms self-information on a single binary neuron and code correlation among pairs of binary neurons that cause more code conflicts. Limitations are also discussed for future research works.

This thesis makes the following main contributions:

1. Makes a comprehensive survey on unsupervised binary representation and summarizes major results;

2. Designs a normalized binary layer to encourage more information to be learned on every bit in binary representation with normalization;

3. Manages to tackle the hash conflict issue between bit pairs with a novel mutual information loss;

4. Evaluated the effectiveness of the proposed methods with ablation studies and
1.4. Thesis Organization

compared with the state-of-the-art unsupervised hashing algorithms with common protocols;

This thesis has five chapters. The first chapter introduces the background and the motivation of this study. Chapter 2 categorizes recent research works on unsupervised hashing and other related research works on mutual information in deep learning and regularization methods on deep clustering. Chapter 3 contains the first major contribution of this work, where we propose the normalized binary layer. The normalization will map the learned distribution to a uniform one to maximize the information that every bit carries. Chapter 4 makes the next major contribution, which is a mutual information loss with gradients from joint probability approximation. Chapter 5 summarizes the contributions and drawbacks of the proposed methods. Possible directions for future study are also discussed in this chapter.
Chapter 2

Literature Review

2.1 Overview

Representation learning is a complex research direction where multiple primal topics are involved. For instance, continuous representation learning, where the network learns high dimensional feature representation, aims at achieving state-of-the-art performance with high-quality features. The representation should be sufficiently informative to distinguish images from others. This topic is widely discussed in both supervised and unsupervised learning. Binary representation learning, suffers from high sparsity in binary spaces, addressing the same problem for both the fully supervised and unsupervised approaches. This chapter will cover recent literature on improving representation quality on both continuous feature learning, or in another word, the coding efficiency on binary representation learning with deep networks.

2.2 Binary Representation Learning

Binary representation learning, which is also called hashing, is an extreme case for discrete feature learning. It intends to learn binary representation for every data in the domain. Early works on deep hashing intend to replicate the rich representation quality on continuous deep features. Generally, binary representation learning can be divided into two categories: fully supervised and unsupervised.
2.2. Binary Representation Learning

2.2.1 Supervised Binary Representation Learning

Directly mapping the continuous representation into binary space is lossy and unstable. Hence, a secure approach is to learn binary representation with labels to replicate what is done on continuous feature learning. In fact, binary representations are capable of identifying human faces [70]. Semantic Hashing [63] introduced deep learning in binary representation learning, where the network learns compact binary hash with Restricted Boltzmann Machine. Then convolutional neural network hashing [76] brought CNN to deep hashing for images, learning hash codes in two stages with a coarse-to-fine manner. Until Network in Network Hashing (NINH) [40], no approach propagates feedback from hash function to CNN. NINH utilizes a triplet ranking loss to reconstruct relative similarity among images, learning both the hash code and representation jointly. Deep pairwise-supervised hashing [43] replaced that triplet loss with a pairwise label which inspired semantic similarity preservation of database in Deep Supervised Hashing [47] that maximizes the discriminability of the obtained hash codes. Supervised hash can also be generalized for multi-modal data retrieval [35]. In conclusion, binary representations are capable of extracting both perceptual and conceptual information that leads to more practical applications in the future. A mutual information loss is first introduced to divergence class-wise representation distribution in supervised binary representation learning [3, 4]. The network is guided with mutual information to separate hash from other semantic categories.

2.2.2 Unsupervised Binary Representation Learning

Conventional Unsupervised Hashing

Conventional unsupervised hashing always uses affine projection on the original or kernelized feature. Local Sensitive Hashing [18] is a typical conventional hashing algorithm. It clusters linearly projected data and assign the same binary code to data that belongs to the same cluster. LSH can be generalized to the Kernelized LSH [39], which
2.2. Binary Representation Learning

takes the higher dimensional data with kernels. There are also some other LSH-based hashing algorithms, like super-bit LSH [32] and non-metric LSH [54]. However, the local sensitive scheme does not guarantee high accuracy in describing semantic nearest neighbours in practice, as it assumes the natural input feature space is conceptually continuous and smooth as the observed spatial distribution.

Iterative Quantization [20] optimizes the hashing matrix with iterative projection and thresholding according to the data. Graph theory is also introduced in unsupervised hashing. Spectral Hashing [73] consider the hashing as a graph partitioning problem and the hash can be retrieved by selecting a subset on the eigenvectors of the graph Laplacian. Anchor Graph Hashing [49] and Discrete Graph Hashing [48] followed this idea and the hash can be spontaneously obtained regarding the training data.

Unsupervised Hashing with Deep Network

Unsupervised deep hashing can be categorized into two series: discriminative network hashing and generative network hashing. Semantic consistency constrained discriminative networks is one major direction for unsupervised hashing methods. Rotation augmentation could retain rotation invariance in discriminative networks to maximize semantic information from the binary representation [45]. Quantization loss and uniform distribution on the learned hash are also considered to keep the capability of the binary hash to the original continuous feature representation. A triplet loss [30] is used to generate discriminative hash while keeping consistency and capability on the binary hash. Samples can be considered as positives and negatives when being compared to the original. Positive samples, which is generated with random rotation from the original, should be closer to the original input than the negatives. Also, the angular distance among samples provides evidence to similarity [28] as the binary code only take the sign of features into its account. Furthermore, the relation can be learned with mild assumption [78]. Similarity can be predicted with a Bayesian optimal classifier during the training process to find the distilled data pair which is further used in hash
2.2. Binary Representation Learning

learning. Alternatively, graphs can also describe feature similarity [64], guiding the network to search better hash for the data distribution. Samples are defined as vertices and the similarity are described as edges in the graph. For each iteration, the edges will be reinitialized with the current similarity on learned features.

Generative methods are also popular in unsupervised binary representation learning. Adversarial learning seems to be effective in extracting binary hash code. BinGAN [84] introduces distance matching and entropy regularization with discriminator to learn how to hash effectively. Similarly, HashGAN [17] also adopted a generator-discriminator architecture to learn binary representation. Both uniform distribution and minimal entropy are considered in an adversarial learning framework. Variational AutoEncoders is also effective in finding a proper hash function on datasets. DVB [65] considers reconstruction on the learned bits, which can reflect the information that a hash can retain and also can be a criterion to maximize the representativity of the binarized representations.

In conclusion, unsupervised binary representation learning always considers semantic consistency on the binarized embedding as well as the uniform code distribution in the full domain. The ultimate goal of encouraging the code to be evenly distributed can be achieved by avoiding hash conflict that harms binary code’s representativity in the code space. However, those code constraints are usually accumulated within mini-batches, which doesn’t represent the actual distribution over the full domain. Also, the criterion like entropy and matrix regularization is not effective enough to avoid local optimum. Most works will be compared in experiments to emphasize the effectiveness of the proposed method.
2.3 Improving Representation Quality Implicitly

2.3.1 Classification: Learn Class-wise Discriminative Feature

Representation learning with deep neural networks is also called deep feature learning. The success of modern computer vision algorithms in areas such as object classification and detection heavily depends on high-quality features. Deep convolutional neural networks such as VGG [66] and ResNet [23] are capable of performing accurate classification and extracting perceptual representation that can be quickly transferred to different domains [14, 50]. Attention-based methods focus on improving feature representation by localizing discriminative parts, utilizing part annotations for more visual details [44, 80]. However, training high-quality features with labels requires a huge amount of annotations. Hence, research works on autoencoders and unsupervised methods became popular ever since.

2.3.2 Autoencoders: Replicate Identical Distribution

Self-supervised feature learning refers to training with labels generated from the original data. It is a popular and well-researched topic in recent years, and more of them began to focus on learning high-quality features, or what is often called embedding. Similar to features extracted with fully-supervised networks, representations learned from self-supervised frameworks is also capable of being transferred to other pretext tasks, which demonstrates its high information preserved, such as video recognition [13] and motion capture [71]. Autoencoders, a series of symmetric neural networks which intend to reconstruct the original data at the output, are powerful tools for self-supervised feature representation learning. Early researches [25, 26] have already investigated compact feature representation learning years ago. They are elegant and versatile that has been extensively studied for various tasks that rely on learning hidden representations [62]. It inspires famous works such as variational autoencoder [37] and adversarial autoencoder [52] that intend to generate realistic data with rich representa-
2.3. Improving Representation Quality Implicitly

With probabilistic modelling, the representations learned by those autoencoders are robust to noises and other interference. Furthermore, discrete representation learning has been explored in VQ-VAE [72] where discrete mapping function has been used on the latent space for generative modelling. Further work [59] has shown that discrete latent representation can generate competitive results compared to state-of-the-art adversarial approaches. All those researches on discrete feature learning suggest that it is still possible to expect high performance on feature representability with more compact discrete code.

2.3.3 Unsupervised Methods: Keep Sample’s Individuality

Unsupervised learning does not require any labels and only work with strong constraints that scatter representation over the representation space. CNN can learn effective visual representations for transfer learning by solving jigsaw puzzles [56]. DCGAN [58] also suggests that unsupervised learning can learn a hierarchy of representations on images. There is also work focusing on learning global conceptual information [21] with an unsupervised framework to extract the conceptual representation of data. Other strategies like rotation are also introduced to self-supervised frameworks to retain semantic information [8]. Memory banks are useful to implement pairwise comparison and are used in self-supervised learning to encourage semantic distance among samples [53]. Those are strong defences on the power of neural networks have the capability of learning patterns without any supervision from data. Unsupervised representation learning can be further combined with clustering [6] to capture complementary statistics from unlabeled data. Those work are crucial evidence on the capability of neural networks in learning robust patterns, which inspire us to think about the feasibility of learning unsupervised binary embedding for the images.
2.4 Learning Representation with External Constraints

Research works on extra constraints that eliminate conflicts in the code space is also a direction to high-quality representation learning. Feature robustness and sparsity would be improved if there is no semantic overlap between each key in the sparse space, especially for the binary case. To compensate for that, a relative entropy minimization is introduced to gather similar data into a rough cluster by reconstructing the original example from distorted data [16]. Another entropy regularization on embedding with adaptive weights is proposed to enhance the feature robustness in fuzzy k-mean clustering [81]. Other regularizations on clustering, like a differentiable constraint on cluster size [15] and structural regularization [69], are also introduced to avoid embedding conflicts in the continuous space.

Mutual information plays an important role in regularization as it provides a correlation between groups in the domain. It has much smaller computation on binary embedding which makes it easier to implement with differentiable learning frameworks. The network is guided with mutual information to separate hash from other semantic categories. In semi-supervised clustering, mutual information can evaluate the co-existence on the assigned label and the prediction [33, 57], which helps to retain the consistency on unlabeled data. Mutual information can also be used for locality-sensitive hashing (LSH) scheme in unsupervised hashing methods. It is proved that $f$-divergence of the framework with two-side approximation [7] can discriminate distributions better than simple Hamming distance in the LSH scheme.

Generally, current research works on regularization with mutual information mainly focus on three aspects: Diverging local representation distribution by minimizing a class-wise, locality-sensitive mutual information loss to the learned binary representation or a consistency constraint on between pseudo label and prediction. Besides, such a criterion can be applied on individual bits to regularize the correlation among each
2.5 Summary

Recent research works on image hashing are inspiring as they have already achieved impressive results on binary representation learning. But for unsupervised cases, there are still spaces to improve as current methods only try to replicate the spatial distribution from pretext tasks. Most of those constraints only consider the consistency between the continuous code and the binaries, which ignores the effect of code biases and correlations that block the optimization to high-performance hashing functions. On the other hand, although current researchers have done many works on improving feature quality, only a few of them investigated external constraints on hashing or clustering algorithms. Those two points justify the research gap and motivation of this thesis, which builds the basis of this research.
Chapter 3

Maximizing Self Information with Normalization

3.1 Centering Distribution for Binary Neurons Keeps Information

Information entropy measures the uncertainty of a random variable. Higher entropy suggests that the variable contains more information than others. A normalization method is proposed in this chapter to encourage more information to be kept by learning balanced binary distribution on every binary neuron. Uniformly distributed random variables on features are the most desired distribution for the binarization process, as it carries the most uncertainty, which is considered as information, among many other distributions. Hence, this thesis proposes a normalization approach to encourage the appearance of uniformly distributed binary neurons. Recent hashing algorithms usually apply similarity loss to preserve the semantic relationship between samples in input space that learn with pairs and suffer from instability with large batch size. The proposed normalization enables a larger batch size during training with similarity loss, boosting the training process with better performance on lower code length. A larger batch size will take more pairs into account when evaluating the similarity error, leading to more generality in domain with hash in better quality.

This chapter will begin with a justification on learning balanced binary neurons in Section 3.2.1, followed by an introduction to the proposed binary layer and its corre-
sponding constraint term in Section 3.2.2. The on-the-fly estimation on expected value is covered in Section 3.2.3.

3.2 Normalized Binary Hash Layer

3.2.1 Learning Balanced Binary Neurons

To state the problem, every element of the learned latent code in network can be considered as obeying a distribution, and the following sections will denote the random variable $X_i$ by an element of the continuous latent code $X$. The expectation of every random variable $X_i$ is denoted by $E[X_i]$.

The main target of the binary quantization layer is to map continuous representations $X \in \mathbb{R}^N$ to binary neurons $B \in \{-1, 1\}^N$. The mapping function may produce some biases that can cause information loss on the quantized binary code. For instance, an element $X_i$ in the latent code $X$ can be interpreted as the presence of a single feature. The binarized neurons would be less informative without alignment to distribution centroid as the more bias on the presence or absence to the existence of a feature. As shown in Figure 3.1, the mapping function discriminates samples disregarding its expected value, leading to more biased behaviour when quantizing the continuous representation. Positive outputs will dominate the negatives, providing less information than the uniformly distributed binary neurons. To normalize the continuous code, the latent code $X_i$ from the network can be decomposed as

$$X_i = N_i + \mu_i$$  \hspace{1cm} (3.1)

where $N_i \in N$ is the normalized random variable and $i$ denotes the index for each dimension, $\mu_i \in \mu$ is the expected value for random variable $X_i \in X$.

With the normalized feature representation $N$, binary neurons can trivially produce unbiased hash with a mapping function that maps the continuous input to binary hash.
3.2. Normalized Binary Hash Layer

Figure 3.1: Dividing a Continuous Neuron into Binary. Gray areas refer to the positives.
3.2. Normalized Binary Hash Layer

Assuming that every element $N_i$ in $N$ is all aligned to their expected value, the probability density function can be simplified as the Bernoulli distribution that resembles its normalized original continuous distribution. Therefore, the recovered biased latent code $\hat{X}$ reconstructed from discrete latent code $B$ can be written as

$$\hat{X}_i = B_i + \mu_i$$

$$= f_{\{-1,1\}}(N_i) + \mu_i$$

(3.2)

where the mapping function $f_{\{-1,1\}} : \mathbb{R} \rightarrow \{-1,1\}$ should be a function that evenly divide the distribution into two piece. The scalar $\mu_i$ is an element from the expected value vector $\mu \in \mathbb{R}^N$, and discrete element $B_i$ belongs to the discrete latent code representation $B$ where $B \in \{-1,1\}^N$.

According to the information theory, uniformly distributed binary random variables could preserve maximal information from the continuous features. We adopted a simple sign function $f_{\{-1,1\}} : \mathbb{R} \rightarrow \{-1,1\}$ as

$$f_{\{-1,1\}} : \mathbb{R} \rightarrow \{-1,1\} : x \mapsto \begin{cases} 1 & x \in (0, \infty) \\ -1 & x \in (-\infty, 0] \end{cases}$$

(3.3)

Theoretically, we cannot adopt this function as activation as the sign function is not differentiable. However, inspired by GreedyHash [67], we could derive indirect gradients for the binary hash with normalization, which will be then covered in Section 3.2.2.

In conclusion, the continuous latent code should be normalized before being translated into binaries to encourage binary neurons to obey a uniform Bernoulli distribution. The maximal entropy appears at uniform distribution so that those balanced neurons can recapture maximal information that improves the hashing representativity. Aligning the distribution with the expected value will evenly divide the probability into two pieces, mapping the continuous feature with less distortion and information loss.
3.2. Normalized Binary Hash Layer

3.2.2 Normalizing Distribution with Consistency Constraint

This section will introduce a differentiable quantization method with learnable parameters, which can be applied in most unsupervised hashing frameworks. Intuitively, binary mapping functions are not differentiable and no gradient is available for backpropagation. However, an approximation can be found by regularizing the output from the binary layer, which allows us to bypass the mapping function with the direct gradient from the quantized code.

The proposed binary quantization layer takes the continuous latent code $X$ from the previous layer and outputs a compact binary code $B$. Inputs will be normalized by the estimated expected value $E[\hat{X}]$ to eliminate the bias on distribution, which refers to the unbalanced probability distribution when binarizing continuous features. Estimation of expected value is obtained regarding current mini-batch at training stage and the accumulated expected value is used during inference. Then the mapping function $f_{\{-1,1\}}$ is applied to map the representation from real space $\mathbb{R}^{N}$ to binary $\{-1,1\}^{N}$.

Gradients for binary quantization function are hard to derive. Inspired by Greedy-Hash [67], binary representation can be updated using indirect gradient flow, allowing the derivative of the binary output to be adopted when updating the continuous code. Intuitively, the update process can be formulated as (3.4). For convenience, we replaced the continuous representation $X$ using decomposed normalized output $N$ with the estimated mean $\hat{\mu}$.

$$
N^{t+1} = N^t - \eta \frac{\partial L^t}{\partial N^t}
$$

$$
\approx B^t - \eta \frac{\partial L^t}{\partial B^t}
$$

where $N$ and $B$ are normalized feature and the binary code respectively, $L$ is the loss function and $\eta$ is the learning rate and $\frac{\partial L^t}{\partial N^t}$ is the partial derivatives at step $t$. One approach to link the gradients back to the network is to let the normalized output $N$ approximate the binarized output $f_{\{-1,1\}}(N^t)$ with a regularization term $\|N^t - f_{\{-1,1\}}(N^t)\|_2^2$. 

21
3.2. Normalized Binary Hash Layer

With this approximation, updates on the normalized code can be then derived as (3.5) and (3.6) by adding a duplicated $B^t$.

$$N^{t+1} = N^t - \eta \frac{\partial L^t}{\partial N^t}$$

$$= N^t - B^t + B^t - \eta \frac{\partial L^t}{\partial N^t}$$

$$= N^t - f_{\{-1,1\}}(N^t) + B^t - \eta \frac{\partial L^t}{\partial N^t}. \quad (3.5)$$

Obviously, with the regularization term, $\|N^t - f_{\{-1,1\}}(N^t)\|_2^2$ will tend to be zero, which makes the update even simpler as shown in

$$N^{t+1} \approx B^t - \eta \frac{\partial L^t}{\partial N^t}. \quad (3.6)$$

The gradient can be bypassed and used for update the network. The overall derivative on the binary representation in the proposed binary layer is described in

$$\frac{\partial L^t}{\partial B^t} \approx \frac{\partial L^t + \lambda \partial L^t_{\text{reg}}}{\partial N^t} = \frac{\partial L^t}{\partial N^t} + \lambda \|N_t - f_{\{-1,1\}}(N^t)\|. \quad (3.7)$$

where $L_{\text{reg}}$ refers to the regularization term. Then the updated binary code can be derived as

$$B^{t+1} = f_{\{-1,1\}}(N^{t+1})$$

$$= f_{\{-1,1\}}(B^t - \eta \frac{\partial L^t}{\partial N^t})$$

$$= f_{\{-1,1\}} \left( N^t - \eta \frac{\partial L^t}{\partial N^t} + \lambda \|N_t - f_{\{-1,1\}}(N^t)\| \right). \quad (3.8)$$

Since the direct gradients from the binary layer are used to train the network, the update on the Binary code can be trivially formulated as with gradient $\eta \frac{\partial L^t}{\partial N^t}$. By bypassing the
3.2. Normalized Binary Hash Layer

gradients, the proposed binary layer can now enable a differentiable hashing function that can be applied to end-to-end training for unsupervised hashing networks with normalization.

3.2.3 Estimating Expected Value with Momentum

The previous section introduces the proposed differentiable binary layer with regularization. Precise estimation of expected value is crucial to the proposed binary quantization. However, estimating the expected value on a changing distribution is hard. The mean of the learned random variable will shift according to the training gradients. A similar on-the-fly estimation strategy to the one in Batch Normalization [31] is adopted to estimate the expected value, which updates the estimated mean with small momentum after every network forward. The mean vector for every element in latent code over each batch is estimated and collected to approximate the actual element-wise mean of the latent code over the whole domain.

In the training process, the binary layer accumulates the estimated expected value \( \hat{\mu} \) using mini-batch mean \( \hat{E}[\bar{X}] \) by

\[
\hat{\mu}_t^{t+1} = (1 - \epsilon)\hat{E}[\bar{X}_t] + \epsilon \hat{\mu}_t^t.
\] (3.9)

where \( \epsilon \) is the momentum that controls the update process of the estimated statistics, \( t \) is the iteration index in training steps and \( \bar{X} \) is a mini-batch of samples. The binary layer will only consider the local mean \( E(\bar{X}) \) to quantize in training iterations and update the estimated mean according to (3.9). Direct forward with local estimation ensures the reliability and efficiency in training while the momentum update stabilizes the estimation. The continuous latent code is then normalized with the mini-batch mean \( \hat{\mu}_t^t \) for the \( t \)-th step. And for inference, the layer will simply use the accumulated momentum expected value to quantize.

There are several benefits for a momentum estimation on expected value. Binary
quantization is sensitive to pre-normalization. A slight shift in expected value will drastically change the binary distribution, which will make the learned bits flip at the output. Most deep hashing algorithm only takes small mini-batch when doing gradient descent updates. With a normalization before binarization, the network can now learn with large mini-batches, accelerating training with better performance. The local expected value within the mini-batch would be close to the actual average value with larger batch size. The expected value is initialized $\hat{\mu}_0$ with zeros and momentum $\epsilon$ are set to a small value so that $\hat{\mu}_t$ will track the estimation from the origin.

With those techniques, the network can learn an aligned binary distribution with a more stable update. Those centred binary random variable will encourage larger information entropy, which indicates more information will be kept on individual bits.

3.3 Experiments

Binary latent codes have a much smaller representation space that may lead to a significant performance drop in distilling explanatory factors from images. However, the conducted experiments in Section 3.3.4 have shown that the binary latent code can provide sufficient information for tasks like classification and image recovery. Both qualitative and quantitative experiments are conducted on MNIST [41], SVHN [55], CIFAR-10 [38].

3.3.1 Datasets

**MNIST** MNIST is an image dataset for handwritten digits classification. It contains 60,000 images for training and 10,000 images for validation. Images are in greyscale with a shape of $28 \times 28$. A pre-trained model for MNIST evaluation is adopted from a GitHub repository [1] and only two MLP layers are used as the feature extractor to capture good representation for handwritten data. It contains 256 neurons in both two layers and all parameters in those two layers are fixed during the training.
stage for hashing. In the image retrieval task, the training set is used for both training and retrieval purpose while the validation set is used as a query set. Top 1,000 samples are considered when accumulating MAP scores.

**SVHN** The Street View House Numbers (SVHN) dataset is an RGB image dataset with digits from real-world scenes. It is similar to MNIST as it contains digits from one to ten but more variety with RGB images from different perspectives. It contains 10 classes for classification and 73,257 images for training and 26,032 images for validation. RGB images are cropped and aligned to $32 \times 32$ squares with the present digits at the centre. A pre-trained model from [1] is also used to extract precise features for digits in variant fonts and colours. Similar to the retrieval setup for MNIST, the network will train and retrieve images with the training set and query with the validation split. MAP scores are computed on the top 1,000 retrieved samples.

**CIFAR-10** CIFAR-10 is a image dataset with 60,000 tiny RGB images from 10 identical categories. Normally the dataset is split into 50,000 to 10,000 regarding training and validation phases. The images are all $32 \times 32$ chunks with the objects in the centre. A VGG-16 [66] model trained with ImageNet [11] serves as a fixed feature extractor. The overall network is trained with the 50,000 training split and query the model with images from the validation set as input. The model will search in the training set and return the top 5,000 similar images from the database.

### 3.3.2 Experimental Setup

The same training setup is applied to every dataset in the benchmark experiments. The initial learning rate is set to $1 \times 10^{-3}$ and the learning rate decays every 100 epochs. Training batch size is set to 512 and hyper-parameter $\varepsilon$ is set to 0.999 in the benchmark experiments. No update will be applied to the parameters in feature extractors and the Stochastic Gradient Descent optimizer optimizes the network with a weight decay of $5 \times 10^{-5}$.
3.3.3 Evaluation Metrics

Image Retrieval will take Mean Average Precision (MAP) as a numerical metric. MAP is widely used in quality evaluation on binary representation [28, 30, 45, 78]. Simple distance function on binary representation, like Hamming distance, can measure similarity among samples. The precision will decrease while increasing the number of images retrieved. Precision-Recall curve can be easily illustrated with discrete sample rate at different P-R pairs and mean average precision (MAP) are adopted to evaluate the overall performance on the retrieval task under different thresholds.

3.3.4 Qualitative and Quantitative Results

The extracted binary representation was applied on several tasks, for example, image retrieval, to test its quality from both categorical and semantic perspectives. It is also visualized with the learned code distribution using t-SNE [51] to demonstrate the code representability. Experiment results show that the code learned with normalization outperforms the baseline significantly.

Image Retrieval

Retrieving semantically associated data from the hash database implies the code’s capability on discriminating samples from different categories. Retrieval performance is evaluated on various open datasets, and collected the MAP evaluation results as shown in Table 3.1.

<table>
<thead>
<tr>
<th></th>
<th>MNIST</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4 bits</td>
<td>8 bits</td>
</tr>
<tr>
<td>Baseline$^{1}$</td>
<td>0.340</td>
<td>0.750</td>
</tr>
<tr>
<td>Baseline$^{2}$</td>
<td>0.204</td>
<td>0.482</td>
</tr>
<tr>
<td>Proposed</td>
<td><strong>0.687</strong></td>
<td><strong>0.951</strong></td>
</tr>
</tbody>
</table>

1 Trained with batch size 32
2 Trained with batch size 512
3.3. Experiments

Normalization helps models to generalize better, especially for hashes with a smaller length. For the MNIST dataset, the 8 bits model with the proposed normalization significantly outperforms baselines trained with both batch size of 32 and 512. The larger batch size can provide more information to loss function which would give biased gradients that interfere with the optimization. With the proposed normalization, bias is eliminated before binarization as the expected value is more accurate in both the training and validation set. Moreover, for the SVHN dataset, the baseline cannot learn anything with a larger batch size but it can learn with normalization as it eliminates the imbalance among propagated gradients.

Retrieved samples are illustrated in different code length in Figure 3.2. Models with the proposed normalization are capable of retrieving semantically relevant samples with smaller code length, even with a 4 bits code. Theoretically, for datasets like MNIST, the minimal code length is $\log_2 10 \approx 3.34$ bits. The proposed normalization can even help 4 bits binary code to retrieve similar code, which suggests it can push models to the theoretical binary coding limit. It can help us to understand how neural networks build compact coding system spontaneously.

Another retrieval experiment on a larger RGB dataset like CIFAR-10 is conducted. The results are reported in Table 3.2. The result suggests that the advantage of the proposed normalization can still hold in natural image retrieval tasks. However, the performance is not as significant as it did in simpler datasets like MNIST and SVHN. But this benchmark is still a piece of strong evidence to support the proposed normalization as it allows models to update faster with better performance in smaller code.

**Visualization on Binary Representation**

Visualization techniques like t-SNE [51] are performed on code to visualize the learned binary neurons. The visualization is demonstrated in Figure 3.3. The learned binary code can form different clusters while discriminating the outliers aside. More
3.3. Experiments

(a) 4 bits

(b) 8 bits

(c) 16 bits

(d) 32 bits

Figure 3.2: Retrieved Images on Different Hash Length with MNIST
3.4 Summary

<table>
<thead>
<tr>
<th></th>
<th>16 bits</th>
<th>32 bits</th>
<th>64 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline(^1)</td>
<td>0.472</td>
<td>0.547</td>
<td>0.589</td>
</tr>
<tr>
<td>Baseline(^2)</td>
<td>0.190</td>
<td>0.106</td>
<td>0.106</td>
</tr>
<tr>
<td>Proposed</td>
<td>0.498</td>
<td>0.536</td>
<td>0.589</td>
</tr>
</tbody>
</table>

\(^1\) Trained with batch size 32
\(^2\) Trained with batch size 512

identical information is kept with binary code with more bits. Smaller code length will spontaneously gather samples into semantic clusters. Furthermore, an extremely small length setup like the 4 bits one will limit the code expressiveness as it does not have enough space for the model to search for the global optimum. Hence, more code conflict will occur with fewer bits learned.

The learned binary code is also analyzed visually without the proposed normalization. The code is trained with the same setup used in other experiments except for the mini-batch size of 32. According to visualization demonstrated in Figure 3.4, the baseline suffers more on code conflict and there are obvious overlaps among semantic clusters. But the proposed normalization will encourage more information kept among binary neurons, providing more space available to search for optimal binary coding in the domain.

3.4 Summary

This chapter introduced the first contribution on binary quantization, learning balanced binary neurons with normalization. A novel binary layer that balances the Bernoulli distribution is proposed in this chapter, encouraging more information to be kept in the binary representation learned from deep neural networks.

Binary quantization is an approximation of learned feature and error on the mapped distribution need to be avoided to retain the information. Misalignment on approximation would cause quantization bias on binary representation, which would cause more
3.4. Summary

Figure 3.3: t-SNE Visualization on Different Hash Length with MNIST

information loss on the approximation. A normalization method is proposed to compensate for the bias during quantization, encouraging the binary layer to approximate a normal distribution with less information loss.
Figure 3.4: Comparison on Code Quality with Normalization. Baseline is trained with batch size 32 while the proposed is trained with batch size 512.
Chapter 4

Minimizing Mutual Information for Unsupervised Hash

4.1 Discourage Correlation for Binary Neurons Avoids Code Conflicts

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_i$</td>
<td>Output binary random variable on $i$-th position</td>
</tr>
<tr>
<td>$P(B_i)$</td>
<td>Marginal Probability of when $B_i$ is true</td>
</tr>
<tr>
<td>$H(B_i)$</td>
<td>Entropy of $B_i$</td>
</tr>
<tr>
<td>$I(B_i;B_j)$</td>
<td>Mutual information between $B_i$ and $B_j$</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Network parameter</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Learning rate for parameter update</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>Error on approximated joint probability</td>
</tr>
<tr>
<td>$\Delta_i$</td>
<td>Update gradient on $P(B_i)$</td>
</tr>
</tbody>
</table>

Mutual information plays an important role in binary representation learning as it provides a correlation between groups in the domain. Mutual information is introduced to diverge class-wise representation distribution in supervised binary representation learning [3, 4]. Conversely, the proposed Scatter and Learn algorithm minimizes mutual information between bit pairs in hash code, encouraging less code conflict in the learned unsupervised binary representation.

Mutual information indicates the mutual dependencies between two random variables. The learned binary representation can also be considered as a group of random variables $P(B_i)$ for the $i$-th in $N$ bits. Intuitively, the correlation between arbitrary pair
{B_i, B_j} can be eliminated by minimizing the mutual information $I(B_i; B_j)$. Minimizing the mutual information can diverge the samples to use up the whole binary coding space. Generally, the unsupervised binary representation learning may suffer from local optimum, while mutual information minimization would help to jump out of it. Gradients from mutual information minimization guide the model to a more spare solution that provides better accuracy and discriminative representations to describe the original data.

Unlike encouraging maximal entropy on every binary neuron in Chapter 3, minimizing mutual information eliminates correlations instead of code biases. These two proposed methods can be applied to the same model at the same time, constraining the code bias and correlation spontaneously.

Section 4.2.2 first discusses how those gradients are connected from the joint probability to the binaries with the approximated gradient. As the gradients are obtained from the approximated joint probability, it is necessary to prove the approximation will eventually converge at the real magnitude of joint probability, which will be covered in Section 4.2.3. At the end of Section 4.2, Section 4.2.4 introduces an application of the proposed method, regularizing the unsupervised binary representation learning by minimizing the mutual information among bits it learned. All used notations are listed in Table 4.1.

### 4.2 Bit-wise Mutual Information Loss

#### 4.2.1 Convexity on Mutual Information

Convexity brings a stable solution to optimization, so we need to find convexity for more stable optimization on mutual information. In this section, we will derive the proof to justify the proposed mutual information loss. The definition of mutual
4.2. Bit-wise Mutual Information Loss

Information can be written as

\[ I(B_i; B_j) = \sum_{B_i \in \{0,1\}} \sum_{B_j \in \{0,1\}} P(B_i, B_j) \log \left( \frac{P(B_i, B_j)}{P(B_i)P(B_j)} \right). \] (4.1)

Without any loss of generality, the mutual information can be considered as a function of \( P(B_i, B_j) \) in (4.1), where \( P(B_i, B_j) \) can be trivially obtained from deep networks with binary outputs. When both \( P(B_i) \) and \( P(B_j) \) are fixed, mutual information \( I(B_i; B_j) \) is convex on \( P(B_i, B_j) \), as it stated in Theorem 4.1.

**Theorem 4.1** (Convexity on Mutual Information). For fixed \( P(B_i) \) and \( P(B_j) \), \( I(B_i; B_j) \) is convex in \( P(B_i, B_j) \).

The proof on Theorem 4.1 is trivial with Theorem 4.4 on page 26 in [61] as the second derivatives of function \( \cdot \log(\cdot) \) is positive on \([0,\infty] \). A sketch of this proof is stated in Proof 4.2.1. By deriving from the definition, we successfully modelled the minimization of mutual information into a convex optimization problem. The problem aims at minimizing \( I(B_i; B_j) \) for arbitrary pair \( \{i, j\} \) on the binary representation over the joint probability \( P(B_i, B_j) \), which is guaranteed to have an optimum as the mutual information is strictly convex on the joint probability.

**Proof on Convexity of Mutual Information.** If we fix the marginal probability \( P(B_i) \) and \( P(B_j) \), the minimizing mutual information is the same as minimizing

\[ \sum_{B_i \in \{0,1\}} \sum_{B_j \in \{0,1\}} P(B_i, B_j) \log(P(B_i, B_j)) \]

which can be simplified as

\[ f(x) = \sum_i x_i \log x_i \]

Hence, proof on conditional convexity on mutual information can be proved by the
4.2. Bit-wise Mutual Information Loss

Convexity of the sum of $(\cdot)\log(\cdot)$. The second derivatives can be written as

$$(x\log x)'' = \frac{1}{x}$$

which is always positive for $x \in [0, \infty]$, so does its sum. According to Theorem 4.4 in [61], function $(\cdot)\log(\cdot)$ is convex as its second derivative is positive. Therefore minimizing mutual information with fixed marginal probabilities is a convex problem regarding the non-negativity of probability.

In our implementation the gradients on mutual information will only update with term $\frac{I(B_i; B_j)}{P(B_i, B_j)}$ to match the convex condition described in Theorem 4.1. The gradient will converge to zeros during this convex optimization, which will stabilize the minimization when served as relaxation in the proposed framework.

4.2.2 Approximated Gradient for Joint Probability

Minimizing mutual information is an optimization problem and gradients are required to update parameters. The partial derivative of $I(B_i; B_j)$ on $P(B_i, B_j)$ is easy to obtain but it is not trivial for the partial derivative of $P(B_i, B_j)$ on $B_i$. Therefore, the proposed method will mainly focus on bridging gradient for $P(B_i, B_j)$.

Let $f$ be the differentiable learning function and $b_i$ be the $i$-th output among $N$ bits from function $f(\cdot; \theta)$ with parameter $\theta$. The target function is mutual information $I(B_i, B_j)$ which is going to be minimized. The partial derivative $\frac{\partial I(B_i, B_j)}{\partial \theta}$ is needed to minimize the observed mutual information. Then the overall partial derivative chain can be derived as

$$\frac{\partial I(B_i, B_j)}{\partial \theta} = \frac{\partial I(B_i, B_j)}{\partial P(B_i, B_j)} \frac{\partial P(B_i, B_j)}{\partial B_i} \frac{\partial B_i}{\partial \theta} \frac{\partial f}{\partial f}.$$  \hfill (4.2)

There are two parts that are not differentiable in the derivative chain. One is the binary layer part $\frac{\partial b_i}{\partial f}$ and another is the joint probability part $\frac{\partial P(B_i, B_j)}{\partial b_i}$. For the binary
4.2. Bit-wise Mutual Information Loss

part, a simple straight through gradient strategy [67] is applied to update the binary representation with the continuous gradient with a code constraint. Then the only problem would be the derivative on joint probability.

Statistically, accumulating partial derivatives to a random variable from joint probability does not make sense. However, the gradient is required to link the output to inputs in differentiable learning functions. A pair of binary variables are either positively or negatively associated. By assuming the weak irrelevance on binary random variable pairs, the gradient can be easily obtained to update according to the computed statistics.

Positive association [12] is proposed to describe a pair of variables \( B_i \) and \( B_j \) that satisfies \( \text{Cov}[B_i, B_j] \geq 0 \) where \( \text{Cov}[B_i, B_j] = E[B_i, B_j] - E[B_i]E[B_j] \). Specifically, the expectation of a binary random variable is the probability when itself is positive, making it easier to find its lower bound for the joint probability. Hence, for a pair of positively associated binary variables, the upper bound can be easily derived as

\[
P(B_i, B_j) \geq P(B_i)P(B_j). \tag{4.3}
\]

Negative association [36] is the opposite to its positive counterpart, where \( \text{Cov}[B_i, \bar{B}_j] \leq 0 \). The notations from previous derivation is inherited for clarification by assuming pair \( \{B_i, B_j\} \) is in positive association. So that for a pair of negatively associated binary variables \( \{B_i, \bar{B}_j\} \), the lower bound can be easily obtained as

\[
P(B_i, \bar{B}_j) \leq P(B_i)P(\bar{B}_j). \tag{4.4}
\]

Inspired by those boundaries from positively and negatively associated pairs, we can assume the term \( |P(B_i, B_j) - P(B_i)P(B_j)| \) is bounded and relatively close to zero, which means that the joint probability can be decomposed as multiplication with error \( \varepsilon \). With (4.3) and (4.4) it is trivial to obtain the inequality for joint probability for a pair
of positively associated binary random variables, as derived

\[
\begin{align*}
P(B_i, B_j) &\approx P(B_i)P(B_j) \\
P(B_i, \bar{B}_j) &\approx P(B_i) - P(B_i)P(B_j) \\
P(\bar{B}_i, B_j) &\approx -P(B_i)P(B_j) + P(B_j) \\
P(\bar{B}_i, \bar{B}_j) &\approx 1 - P(B_i) - P(B_j) + P(B_i)P(B_j). 
\end{align*}
\]

(4.5)

Similarly, the negative will also provide a reversed boundary for \( P(B_i, B_j) \). A pair of associated binary codes are either positively or negatively associated in statistics. Intuitively, each pair of independent binary code will be not associated when it is at the optimal solution subject to the mutual information. The goal is to eliminate association among pairs of hash. When it is in its optimal solution, all inequalities in (4.5) will turn into equalities, or it will have a positive or negative error from the joint to the multiplication of marginals. A slack variable \( \varepsilon \) can be introduced here to describe the upper or lower limit of the error. Then it would be trivial to prove the \( \varepsilon \)-Convergence on this approximation with this slackness variable \( \varepsilon \).

Joint probability for associated pair can be rewritten as below with a slack variable \( \varepsilon \) as

\[
P(B_i, B_j) = P(B_i)P(B_j) - \varepsilon \geq P(B_i)P(B_j)
\]

(4.6)

where the slack variable \( \varepsilon \) is bounded, which means it will never go infinity as both the joint \( P(B_i, B_j) \) and \( P(B_i)P(B_j) \) are bounded. Notably, when the slackness variable \( \varepsilon \) goes to zero, the approximation will be exact equation and confirming that two binary neurons \( B_i \) and \( B_j \) are independent, otherwise correlated. As we only doing the approximation only on the backpropagation side with a small learning rate, hence the effect of the slackness variable is minor to the actual optimization process.

Both the upper bound and lower bound of joint probability can be obtained by mixing positive and negative association condition. The slack variable \( \varepsilon \) will converge and squeeze the joint probability to be independently multiplied. It will stagger around zero
4.2. Bit-wise Mutual Information Loss

as the variable pairs will switch between positive and negative association frequently. The derivative obtained by the expanded value can provide an approximated gradient that establishes a relationship between joint probability and the output binary representation. In the next section, the convergence on the optimization with approximated derivatives will be discussed along with the slack variable $\epsilon$.

With the slackness $\epsilon$, the derivative can be trivially derived as

$$\frac{\partial P(B_i, B_j)}{\partial B_i} \approx \frac{\partial P(B_i)P(B_j)}{\partial B_i} = P(B_j)$$  \hspace{1cm} (4.7)

where $\frac{\partial P(B_i)}{\partial B_i} = \frac{1}{N}$ for statistics over $N$ samples as the unconditional probability density function of binary variable $B_i$ can be considered as summation over positive samples. The derivatives of the rest conditions are stated in (4.8). The gradient can be calculated according to those approximated derivatives

$$\frac{\partial P(B_i, \bar{B}_j)}{\partial B_i} \approx \frac{\partial P(B_i)P(\bar{B}_j)}{\partial B_i} = 1 - P(B_j)$$

$$\frac{\partial P(B_i, \bar{B}_j)}{\partial B_j} \approx \frac{\partial P(\bar{B}_i)P(B_j)}{\partial B_j} = -P(B_j)$$  \hspace{1cm} (4.8)

$$\frac{\partial P(\bar{B}_i, B_j)}{\partial B_i} \approx \frac{\partial P(\bar{B}_i)P(B_j)}{\partial B_i} = -(1 - P(B_j)).$$

From an analytic perspective, compensating the bi-variant function with a function of one variable is over-acting on the gradient. The slack variable will bounce between the optimum which would cause diverge with large step sizes. However, it will eventually provide a more precise gradient with the convergence of $\epsilon$. Further discussion will be covered in the next section and it is proved that with small step sizes the slack variable will converge to 0 at last.

The final gradient will accumulate the gradients on $B_i$ with respect to other bits $\{B_{j_1}, ..., B_{j_N}\}$, and the actual accumulated joint probability will be used to compute loss when the network feeds forward.
4.2. Bit-wise Mutual Information Loss

4.2.3 ε-Convergence of Joint Probability Approximation

The convergence of joint probability approximation needs to be proved to ensure accurate update before being applied during optimization. Section 4.2.2 has introduced a slack variable ε that defines the upper and lower bound of the difference between the estimated and approximated joint probability.

In (4.7), the partial derivative is obtained using the approximated joint probability \( P(B_i)P(B_j) - \varepsilon \). A learning step-size \( \eta' \) is introduced to formulate the updated joint probability with the approximated gradient. Then a single iteration at \( i \)-th step can be derived as

\[
P^{t+1}(B_i, B_j) = P^t(B_i, B_j) - \eta' \frac{\partial P^t(B_i, B_j)}{\partial P^t(B_i)}
\]  

\[
= P^t(B_i, B_j) - \eta' P^t(B_j).
\]

By introducing the slack variable \( \varepsilon \) and the gradient \( \Delta_i^t \) and \( \Delta_j^t \) on \( P^t(B_i) \) and \( P^t(B_j) \) respectively for the \( t \)-th iteration, \( P^t(B_i, B_j) \) can be replaced with \( P^t(B_i)P^t(B_j) + \varepsilon \) according to (4.6) and also substitute \( P^t(B_i) - \Delta_i^tP^t(B_i) \) for \( P^{t+1}(B_i) \). We can derive the previous equation (4.9) as

\[
P^{t+1}(B_i)P^{t+1}(B_j) - \varepsilon^{t+1} = P^t(B_i)P^t(B_j) - \varepsilon^t - \eta' P^t(B_j)
\]

\[
(P^t(B_i) - \Delta_i^t)(P^t(B_j) - \Delta_j^t) - \varepsilon^{t+1} = P^t(B_i)P^t(B_j) - \varepsilon^t - \eta' P^t(B_j)
\]

\[
P^t(B_i)P^t(B_j) - \Delta_i^tP^t(B_j) - \Delta_j^tP^t(B_i) + \Delta_i^t\Delta_j^t - \varepsilon^{t+1} = P^t(B_i)P^t(B_j) - \varepsilon^t - \eta' P^t(B_j)
\]

\[
- \Delta_i^tP^t(B_j) - \Delta_j^tP^t(B_i) + \Delta_i^t\Delta_j^t - \varepsilon^{t+1} = -\varepsilon^t - \eta' P^t(B_j).
\]

(4.10)

Obviously, the error difference on slack variable \( \varepsilon \) can be easily derived from iterations to iterations by deriving the update function to the joint probability with approximation. The error difference \( \varepsilon^{t+1} - \varepsilon^t \) can be illustrated as equation below

\[
\varepsilon^{t+1} - \varepsilon^t = \eta' P^t(B_j) - \Delta_i^tP^t(B_j) - \Delta_j^tP^t(B_i) + \Delta_i^t\Delta_j^t.
\]

(4.11)

To prove the convergence, all differences are accumulated to form a series in \( t \). Then
the inequality can build up a relationship between $\varepsilon^T - \varepsilon^1$ and the series described in (4.11). As both the learning rate converges to 0 and the probability is non-negative and bounded, it can be trivially implied that series $\sum_{i=1}^{\infty} - \eta^i P(B_j)$ is convergent. With all derivation stated above, the problem is transformed into proof on the convergence of series shown in

$$
\sum_{t=1}^{T} (\varepsilon^{t+1} - \varepsilon^t) = \sum_{t=1}^{T} \left( \eta^i P^t(B_j) - \Delta_i^t P^t(B_j) - \Delta_j^t P^t(B_i) + \Delta_i^t \Delta_j^t \right)
$$

$$
= \sum_{t=1}^{T} \left[ \eta^i P^t(B_j) - \Delta_i^t \left( P^t(B_j) - \frac{\Delta_i^t}{2} \right) - \Delta_j^t \left( P^t(B_i) + \frac{\Delta_i^t}{2} \right) \right]
$$

$$
= \sum_{t=1}^{T} \left[ \eta^i P^t(B_j) - \Delta_i^t \frac{P^{t+1}(B_j) + P^t(B_j)}{2} - \Delta_j^t \frac{P^{t+1}(B_i) + P^t(B_i)}{2} \right].
$$

(4.12)

Then we can shrink the right hand side and simplify Eq.(4.12) as a series described below

$$
\sum_{t=1}^{T} \left[ \eta^i P^t(B_j) - \Delta_i^t \frac{P^{t+1}(B_j) + P^t(B_j)}{2} - \Delta_j^t \frac{P^{t+1}(B_i) + P^t(B_i)}{2} \right].
$$

(4.13)

The update $\Delta^t$ is controlled by the step size $\eta_t$ and the step size $\eta^t$ will tend to be zero as step count $t$ increases. Then both $|\Delta_i^t|$ and $|\Delta_j^t|$ converges. Also, with the bounded term $\frac{P^{t+1}(B_j) + P^t(B_j)}{2}$ and $\frac{P^{t+1}(B_i) + P^t(B_i)}{2}$, last two components in (4.13) is also convergent. Then all components in the original series in (4.13) is convergent, which means series $\sum_{t=1}^{T} (\varepsilon^{t+1} - \varepsilon^t)$ is convergent. Therefore, $\sum_{t=1}^{T} (\varepsilon^{t+1} - \varepsilon^t)$ tend to be constant if $T \to \infty$, which suggests

$$
\lim_{t \to \infty} (\varepsilon^{t+1} - \varepsilon^t) = 0.
$$

In another word, the error brought by approximated joint probability gradient will converge to a constant during optimization. However, the error is not guaranteed to converge to zero as the convergence condition for series $\sum_{t=1}^{T} (\varepsilon^{t+1} + \varepsilon^t)$ is hard to find in
4.2. Bit-wise Mutual Information Loss

Proof on $\epsilon$-Convergence is a weak condition that can guide us to design the algorithm. In the experimental implementation, a small multiplier is designed to diminish the error produced during approximation. A larger multiplier will shuffle the code distribution disregarding its original semantic relation and increase $\epsilon$, which will cause divergence on the derivative that comes from an approximated joint probability. On the other hand, we also adopt random initialization to improve the initial irrelevance between binary neurons, which gives a smaller error $\epsilon$ for the algorithm to start with.

4.2.4 Scatter and Learn

Minimizing mutual information can diverge code distribution in binary space. This section will introduce the proposed relaxed algorithm *Scatter and Learn*. The minimization is served as a shuffling process that encourages the encoder to use up the full binary space. A proper amount of push will guide the model to escape the local optimum. Both the network and loss function setup in GreedyHash [67] is kept as the shuffling process is independent of the regular unsupervised hashing optimization.

To make the update precisely, the full dataset will be used to provide an accurate estimation of probabilities. For a finite dataset, the probability distribution can be treated as an approximation of the realistic distribution. Hence, it is essential to assume that there are enough samples to the observed probability distribution function $\hat{P}(B_i)$ to be closed to the realistic probability distribution function $P(B_i)$.

The proposed approach *Scatter and Learn* is described as pseudo-code in Algorithm 1. The shuffling process always happens at the beginning of every epoch, encouraging the network to fill up the whole binary space. It will diverge the code distribution and help to jump out of the local optimum on the learned hash.

A regular optimization on the unsupervised learning process is also needed to collaborate optimization on binary code. In the experimental implementation, a cosine similarity loss [67] is adopted on binary code from the actual learning part in the pro-
4.2. Bit-wise Mutual Information Loss

posed Scatter and Learn algorithm. It will stimulate the learned binary code to imitate
the angular relationship on the input feature as shown below

\[ L_{sim} = \| \text{sim}(H_1, H_2) - \text{sim}(B_1, B_2) \|_2^2 \]  

(4.14)

where \( H_1 \) and \( H_2 \) are input feature. \( B_1 \) and \( B_2 \) are the hash and the function \( \text{sim}(\cdot) \) is
the cosine similarity function defined as

\[ \text{sim}(A, B) = \frac{AB}{\|A\|\|B\|} \]  

(4.15)

where \( A \) and \( B \) are vectors with same number of dimension.

Recalling the objective function that optimizes the model, the consistency regu-
larization \( L_{\text{reg}} \) defined in (4.16) is combined with the cosine similarity loss \( L_{sim} \) with
hyper-parameter \( \alpha \) as the regular unsupervised hashing loss \( L_r \). The consistency loss
will align representation between two sides at the binarization and the cosine similarity loss \( L_{sim} \) is a regular unsupervised hash learning step. Mutual information loss \( L_m \)
is updated with all samples collected in the training set before every epoch to capture
accurate estimation on mutual information. The algorithm is described as Algorithm 1.

\[ L = L_{sim} + L_{\text{reg}} = \| \text{sim}(H_1, H_2) - \text{sim}(B_1, B_2) \|_2^2 + \alpha \|H - B\|_2^2 \]  

(4.16)

In the experimental implementation, a pre-trained VGG-16 [66] extracts features
from images without optimizing its parameters and the generated binary representa-
tion is collected over the whole dataset to compute the mutual information. Only a
naive fully connected layer is adopted to consume the input feature to the learned hash.
To stabilize the training, the proposed method minimizes mutual information only on
\( P(B_i, B_j) \) and gradients from marginal probability are cut from the backpropagation.
Also, the mutual information is accumulated as a triangular matrix to stabilize train-
4.3 Experiments

**Algorithm 1: Scatter and Learn**

**Result:** Obtain function parameter $\theta$

Initialize network parameter $\theta$, learning rate $\eta$, hyper-parameter $\alpha$ and $\beta$;

**foreach** epoch **do**

Estimate joint probability $\hat{P}(B_i, B_j)$ for $i, j \in 1..N$;

Calculate mutual information loss $\beta L_m$;

Update $\theta$ with the approximated derivative $\eta \frac{\partial L_m}{\partial \theta}$;

**foreach** minismatch in dataset **do**

   Calculate regular loss $L_r$;

   Update $\theta$ according to $L_r = L_{sim} + \alpha L_{reg}$ with learning rate $\eta$;

**end**

**end**

ing and the loss function is applied with a multiplier $\beta$ to balance the magnitude on gradients.

Notably, the minimization process is convex so that the whole process will have a global solution to the optimization. Experimental results also suggest that the proposed mutual information loss is effective in encouraging less code conflict in the binary space.

### 4.3 Experiments

Experiments were conducted to evaluate the performance of the proposed unsupervised hashing algorithm on open datasets. For every experiment in this section, a fixed random seed with deterministic behaviour was applied to train the model. Experiments used zero as the random seed for both network and data loader to improve reproducibility. Datasets and reproducible code are provided https://github.com/mpskex/Minimizing-Mutual-Information.
4.3. Experiments

### 4.3.1 Datasets

The experiments were conducted on three open datasets: CIFAR-10 [38], NUS-Wide [9] and MS-COCO [46]. No data augmentation technique is applied during training and all class labels are not used in the unsupervised setup.

1. **CIFAR-10** is an RGB image dataset with 60K $32 \times 32$ images with class annotations from 10 different categories. Training and testing split are followed according to the CIFAR-10 (II) setting in GreedyHash [67] which takes 5,000 images each class for training and the rest of 10,000 images for querying. The training dataset will also be served as the retrieval set in this setting. The image will be resized to $224 \times 224$ before being fed to the feature extractor. Pixels are normalized according to mean pixel and standard deviation of pixels on ImageNet [11]. The top-1000 similar images will be considered in the mean average precision (MAP@1000) evaluation.

2. **NUS-Wide** contains about 270,000 images with 81 different concepts. There can be multiple concepts for a single image. A subset of 21 most common concepts is used in the conducted experiments, picking 195,834 images for the experiment. The original data split setup in [76] are kept for further performance comparison, taking 500 images from each concept for training and 100 images from each category for querying. The rest of the data is kept as a retrieval dataset, providing similar samples for every query image in the test set. Input images are resized to $256 \times 256$ before being center cropped to size $224 \times 224$. Similar to CIFAR-10(II) setup, pixels are normalized to the same mean and standard deviation. The same setting as other work [76] is also applied to evaluate the results, take top-5000 neighbours to evaluate MAP on NUS-Wide.

3. **MS-COCO** provides images from a large scope of concepts. This experiment used *trainval2014* for COCO dataset to match the setup in HashNet [5]. The pruned dataset contains 12,2218 images from 80 different classes. 5,000 images
4.3. Experiments

are randomly selected for query and another 10,000 are also picked for training purpose. The remaining is reserved as a database during MAP evaluation. The conducted experiments used the same strategy as for NUS-Wide to resize and crop images. Commonly used evaluation setting [5, 83] is also adopted on the COCO dataset to keep the consistency, considering top-5000 similar images to accumulate MAP score.

4.3.2 Experimental Setup

Experiments were conducted using a batch size of 32 and a learning rate of $1 \times 10^{-3}$. Hyper-parameters were set to $\alpha = 0.1$ in the experiments and $1 \times 10^{-4}$, $1 \times 10^{-3}$, $1 \times 10^{-2}$ on $\beta$ for the 16-bits, 32-bits, 64 bits model respectively. A multi-step learning rate decay is applied in the experiments. It will decay at a rate of 0.1 every 100 epoch. The networks are trained for 300 epochs with a standard SGD optimizer with the momentum of 0.9 and weight decay of $5 \times 10^{-4}$. Notably, a special optimizer is applied to mutual information minimization, which does not have any momentum or weight decay. This strategy can ensure that the network will only take the precise gradient to shuffle the binary representation.

4.3.3 Evaluation Metrics

Apart from the mean average precision (MAP) mentioned in Chapter 3, the precision-recall curve will present the performance with more visual details. It illustrates the relation between precision and recall, which is another form to describe retrieval performance over different settings. Notably, the integral of the precision-recall curve is positively related to the MAP score.
4.3. Experiments

Figure 4.1: Precision-Recall Curves of Ours and Compared Methods on CIFAR-10 Dataset
4.3. Experiments

Table 4.2: MAP Evaluation with Unsupervised Binary Representation

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10</th>
<th>NUS-WIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>SpherH [24]</td>
<td>0.254</td>
<td>0.291</td>
</tr>
<tr>
<td>ITQ [20]</td>
<td>0.305</td>
<td>0.325</td>
</tr>
<tr>
<td>DGH [48]</td>
<td>0.335</td>
<td>0.353</td>
</tr>
<tr>
<td>DeepBit [45]</td>
<td>0.194</td>
<td>0.249</td>
</tr>
<tr>
<td>SGH [10]</td>
<td>0.435</td>
<td>0.437</td>
</tr>
<tr>
<td>BinGAN [84]</td>
<td>0.476</td>
<td>0.512</td>
</tr>
<tr>
<td>HashGAN [17]</td>
<td>0.447</td>
<td>0.463</td>
</tr>
<tr>
<td>DVB [65]</td>
<td>0.403</td>
<td>0.422</td>
</tr>
<tr>
<td>DistillHash [78]</td>
<td>0.284</td>
<td>0.285</td>
</tr>
<tr>
<td>GreedyHash [67]</td>
<td>0.448</td>
<td>0.473</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td><strong>0.507</strong></td>
<td><strong>0.562</strong></td>
</tr>
</tbody>
</table>

Table 4.3: MAP Evaluation with Unsupervised Binary Representation

<table>
<thead>
<tr>
<th>Method</th>
<th>MS-COCO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>SpherH [24]</td>
<td>0.516</td>
</tr>
<tr>
<td>ITQ [20]</td>
<td>0.598</td>
</tr>
<tr>
<td>DGH [48]</td>
<td>0.613</td>
</tr>
<tr>
<td>DeepBit [45]</td>
<td>0.407</td>
</tr>
<tr>
<td>SGH [10]</td>
<td>0.594</td>
</tr>
<tr>
<td>BinGAN [84]</td>
<td>0.651</td>
</tr>
<tr>
<td>DVB [65]</td>
<td>0.570</td>
</tr>
<tr>
<td>GreedyHash [67]</td>
<td>0.582</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td><strong>0.703</strong></td>
</tr>
</tbody>
</table>

4.3.4 Evaluation Results

Benchmark on Image Retrieval

Evaluation results on MAP is illustrated in Table 4.2 and Table 4.3. All methods compared are using the same deep VGG-16 feature to ensure a fair comparison. Identical training and test split setup are also used to evaluate the proposed method.

The proposed algorithm is compared with several state-of-the-art methods including SpherH [24], ITQ [20], DGH [48], DeepBit [45], SGH [10], BinGAN [84], HashGAN [17], DVB [65], DistillHash [78] and also GreedyHash [67]. According to evaluation
results on image retrieval in both Table 4.2 and Table 4.2, the proposed method has advantages on all code length settings comparing to current state-of-the-art methods. Large gaps can be observed on 64-bits with the proposed Scatter and Learn algorithm. The proposed regular optimization step is similar to GreedyHash [67] as they both use cosine similarity loss and the same regularization loss during training. It can be observed that the mutual information minimization did help the network to improve representativity over the whole domain. Even comparing to the generative method like BinGAN [84] and HashGAN [17], the proposed approach can still advance them with considerable improvement.

The proposed Scatter and Learn achieves higher performance on larger code length. Results with 64 bits code on CIFAR-10(II) achieved 59.2% on MAP score which is 9.1% higher than GreedyHash [67] and 7.2% higher than BinGAN [84]. The improvement made on NUS-Wide and MS-COCO is 4.5% and 7.5% comparing to the highest score among SOTA.

Precision-Recall curves are also collected to compare the proposed approach to others. Experiments used CIFAR-10 and $\beta$ is set as described in Section 4.2.4. Results are demonstrated in Fig. 4.1. Most compared methods cannot retrieve semantically related clusters from the database, which means those hashing approaches may not be able to generalize semantic hash for similar samples. Those methods surely provide accurate neighbours but are trapped in local optimum with no constraint on code generation. However, with the proposed mutual information, precision is well kept at low recall, which means the hashing model trained with mutual information loss can be more robust and informative on clustering semantically related samples.

**Visualization**

Visualization techniques like t-SNE [51] is adopted to evaluated code quality on the learned binary representation. Results on visualization with the 32 bits and 64 bits hash is illustrated in Fig. 4.2. The unsupervised binarized embedding is able to separate
samples with respect to their concepts. Even with a smaller form, for example, binary representation with only 32 bits, can still gather samples as neighbours with respect to the semantic label without any supervision from the ground truth.

Retrieved images are illustrated with the query image in Fig. 4.3. Four randomly selected query images are presented with 10 retrieved most similar images from the database set of NUS-Wide. The top-10 nearest neighbours are semantically related, which suggests that the proposed relaxation can help a simple model to retrieve semantic neighbours more effectively with the unsupervised binary hash.

### 4.3.5 Empirical Analysis

**Ablation Study**

Intuitively, proper amount of pushing can improve performance on hashing. Hyperparameter $\beta$ is designed to control the strength of push during optimization. Smaller value for $\beta$ will diminish the effect of minimizing mutual information while larger $\beta$ will degrade the code quality in performance on retrieval.

<table>
<thead>
<tr>
<th>$\beta$</th>
<th>0.1</th>
<th>0.01</th>
<th>0.001</th>
<th>0.0001</th>
<th>0.00001</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours 16 bits</td>
<td>0.465</td>
<td>0.474</td>
<td>0.482</td>
<td><strong>0.507</strong></td>
<td>0.493</td>
<td>0.477</td>
</tr>
<tr>
<td>Ours 32 bits</td>
<td>0.539</td>
<td>0.553</td>
<td><strong>0.562</strong></td>
<td>0.554</td>
<td>0.560</td>
<td>0.549</td>
</tr>
<tr>
<td>Ours 64 bits</td>
<td>0.591</td>
<td><strong>0.592</strong></td>
<td>0.591</td>
<td>0.590</td>
<td>0.589</td>
<td>0.589</td>
</tr>
</tbody>
</table>

Table 4.4 demonstrates the controlled experiment with hyper-parameter $\beta$ on CIFAR-10(II) dataset. Results are evaluated on MAP and every setup is the same except $\beta$. Table 4.4 proved the previously stated assumption on $\beta$, showing us that appropriate push is needed to improve the code quality. A larger $\beta$, for example, $\beta = 0.1$ will disrupt the regular unsupervised training as it leads to a lower performance that is even worse than the baseline($\beta = 0$). Also, if $\beta$ is set to a too-small value, eg. $\beta = 1 \times 10^{-5}$, the optimization will degrade to the baseline method. Precision-Recall curve with different $\beta$ values on 16 bits setup in Fig. 4.4 also gives strong evidence on the previous
4.3. Experiments

Figure 4.2: t-SNE Visualization on CIFAR-10 Dataset
4.3. Experiments

Figure 4.3: Retrieved Images on Right with Query Image on Left from NUS-Wide with 16 Bits Embedding

Figure 4.4: Precision-Recall Curve with Different $\beta$ on 16 Bits

On the other hand, the optimal value for $\beta$ increases as the code length goes up. Smaller code length will cause more accumulative gradients on each bit so that the mutual information will be more dominant compared to the regular unsupervised constraints while larger code could help the gradients to relax by distributing the error to other bits. But still, the proper $\beta$ is crucial to encourage the binary code to fill up space.

Code Analysis

Experiments also collected statistics to evaluate how will minimization of mutual information would affect the code utilization in binary space. The experiments con-
4.3. Experiments

(a) $\beta = 0.0001$

(b) $\beta = 0.001$

(c) $\beta = 0.01$

Figure 4.5: Binary Space Utilization under Different $\beta$
4.3. Experiments

trolled hyper-parameter $\beta$ to assess the binary space utilization with different setups. Experiments used the CIFAR-10 dataset and follow the same training protocol as described above. The code statistics are sorted with respect to the counted number to evaluate the utilization of binary space.

As shown in Fig 4.5, minimization of mutual information can encourage the network to use more keys in the binary space. The maximal value on single key decreases as the value of $\beta$ increases. Also, the minimum of the code count increases which means the code distribution is more flattened. It can be interpreted as more binary space is used by minimizing the mutual information with the proposed method. Though it is achieving what is expected, it does not mean larger $\beta$ is good for a hashing algorithm. According to the conducted ablation study in the previous section, larger $\beta$ will cause a performance drop as it may shuffle the code too hard when searching for generalized semantic hash in the binary space.

Robustness to Noisy Data

![Figure 4.6: Mean Average Precision on Different Noise Rate](image)

Realistic data from sensors could be noisy and the proposed method should be robust to noise. To simulate the noisy data in the wild, we applied additive noise to the
query data and retrieved images according to those noisy queries. To evaluate the robustness of the learned hash code, we controlled the scale of the additive noise from 0.0 to 2.0. Then we collect mean average precision on different noise scale and illustrated them in Fig 4.6. This experiment is conducted using 16 bits model on the NUS-Wide dataset with the proposed mutual information.

According to our experiment, the learned hash code is effectively robust to noise. Features extracted from CNN is already robust to noise as the convolutional layers consider the patterns more instead of the absolute values on pixels. Recent research works also suggested CNNs is capable of tackle adversarial samples [22]. Hence, there is no surprise to the robustness of the learned hash code.

**Effect of Mutual Information Minimization on Binary Code**

Experiments investigated effects on the proposed algorithm with approximated joint probability during optimization to support the proof on its $\varepsilon$-convergence discussed in Section 4.2.3. First, this experiment is conducted with the CIFAR-10 (II) dataset, trying to encourage the binary code to utilize the whole binary space. The network will be only optimized according to the mutual information loss on binary code. To visualize, the embedding is divided into two subsets and converted into integers respectively. Then the experiments used two converted integers to represent the sample’s coordinates on a 2D space. The visualization result will not demonstrate the semantic relationship but an optimization process that justifies the proposed motivation. Result is demonstrated in Fig. 4.7.

Networks are trained with a learning rate of $1 \times 10^{-5}$ with only the optimization step on mutual information loss. The algorithm converges in about 30 iterations. The binary embedding is visualized using the strategy discussed above. The binary code is diverging into the full space by minimizing mutual information with the proposed approach. The solution is stable after it converged, which verifies the proposed motivation and design discussed at the first of this chapter. There will be an optimal solution
4.4 Summary

This chapter proposes a novel meta-algorithm named “Scatter and Learn” for hash quality enhancement through diverging code distribution in binary space, which can
also help the model escape local minimum. As there is limited coding space in binary representation, this may cause the hash or code conflict in the binary space especially when two different samples share the same code for binary representations. In this study, we identified code conflict in a low dimensional space as a new barrier to high-quality unsupervised hashing. Minimizing mutual information can diverge the code distribution which relaxes the code conflict. The proposed learning algorithm on binary representation enables the network to fully utilize the binary space, which mitigates hash conflict on semantically dissimilar samples. With a proper amount of shuffling, the network can jump out of the local minimum and also generalize better with the proposed mutual information loss. We also provide proof of the $\varepsilon$-Convergence in the algorithm. The proposed approach is flexible and can be applied to other hashing algorithms to enhance model generality. Overall, the proposed learning algorithm Scatter and Learn can encourage less code conflict in the binary, guiding the network to generalize better and learn better binary representations.
Chapter 5

Conclusions

5.1 Summary of Contributions

This thesis focuses on improving binary representation coding efficiency with deep neural networks. Binary representation has fewer coding space than the continuous space therefore there will be a higher chance when two identically different samples share the same code in binary form. This is often recognized as hash conflict or coding deficiency in binary space. This thesis enhanced coding efficiency for binary hash from two perspectives: discourage code biases on single bit and correlations among bits. This work accomplished those two goals by introducing two concepts: a normalized binary layer and a bit-wise mutual information loss on the learned hash codes. Balanced binary neurons with normalization have higher entropy which means more information is preserved in binary form. Larger batch size was also achieved that speeds up the training process and representation quality. On the other hand, this thesis approximately minimizes mutual information to eliminate code correlation. The proposed method is validated with convergence proof and experiments both qualitatively and quantitatively. Both the two proposed methods were evaluated on public datasets with standard metrics and setups. Experiments demonstrated the effectiveness of those methods and supported the hypotheses proposed where the problem of code conflict exists and affects the hash quality in unsupervised setups.

Self entropy is maximized by normalization on binary neurons. Estimated expected value can eliminate any bias on neuron before binarization, which would spontaneously encourage more information learned on that bit. Greedy approximation between con-
tinuous and binary neurons is introduced to connect the gradient from the discrete to the continuous. The expected value of a single neuron is estimated in a large batch size with momentum updates. Experiments have suggested the proposed normalization can significantly tackle the code conflict issue, approaching the lower theoretical boundary for coding system in information theory.

From another perspective, mutual information also plays an important role in eliminating code conflict by diverging the code distribution in binary space. A novel meta-algorithm called Scatter and Learn is introduced to diverge code distribution in binary space which can enhance hash quality and also help the model escape local minimum. The thesis also provides proof on the convexity of the optimization problem and $\varepsilon$-Convergence of the proposed algorithm. Mutual information loss on binary representation can encourage the network to fully utilize the binary space, which mitigates hash conflict on semantically dissimilar samples. With a proper amount of shuffling, the network can jump out of the local minimum and also generalize better with the mutual information loss. The proposed approach is flexible and can be applied to other hashing algorithms to enhance model generality.

5.2 Limitations and Future Study

Training with normalization requires accurate mean estimation, which would be hard in an on-the-fly manner. The similarity learning process also brings noises to the learned binary code, making the estimation even harder. Current pair sampling on the similarity loss might be a cause for this noise, as the variance of pairs is not sufficient to provide a general gradient to the optimization. Considering more pairs would improve the stability and the effectiveness of the proposed normalization.

For mutual information minimization, a more concrete condition on convergence could help to optimize the gradient flow during training. Both theoretical and numerical study is necessary to investigate in order to improve efficiency. Furthermore,
the proposed mutual information minimization is too heavy to perform on very large databases. Minimizing mutual information need representation on every element in the training set to estimate joint probability. A flexible on-the-fly technique on the joint probability estimation will broaden its application and also accelerate the training possibility. Minimizing mutual information on continuous outputs is also an interesting direction as it will encourage independence among the output neurons. Learning more identical neurons will eliminate redundancy in networks and encourage more node to be pruned when reducing the size of neural networks.
Bibliography


[38] A. Krizhevsky, Learning multiple layers of features from tiny images, University of Toronto, (2012). → pages 24, 44


