Binary Deep Learning
Binary Weights and Semantic Binary Data Compression

Li, Y.

DOI
10.4233/uuid:e3b0da5b-dee4-4ff7-a8b2-b3f5bc15f72d

Publication date
2024

Document Version
Final published version

Citation (APA)
Li, Y. (2024). *Binary Deep Learning: Binary Weights and Semantic Binary Data Compression* . [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:e3b0da5b-dee4-4ff7-a8b2-b3f5bc15f72d

Binary Deep Learning

Binary weights and semantic binary data compression

BINARY DEEP LEARNING

BINARY WEIGHTS AND SEMANTIC BINARY DATA COMPRESSION

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on
Tuesday, 25 June 2024 at 10:00 o'clock

by

Yunqiang LI

Master of Science in Information and Communication Engineering,
Air Force Engineering University, Xi'an, China,
Born in Henan, China

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. M.J.T. Reinders,	Delft University of Technology, promotor
Dr. J.C. van Gemert,	Delft University of Technology, promotor
Independent members:	
Dr. C. Gao,	Delft University of Technology
Dr.ir. A.L. Varbanescu	University of Twente
Prof.dr.ir. G.J.T. Leus,	Delft University of Technology
Prof.dr. H. Corporaal,	Eindhoven University of Technology
Reserve members:	
Prof.dr. A. Hanjalic,	Delft University of Technology

ISBN 978-94-6384-601-1

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

The future of artificial intelligence (AI) is on the edge.

Yunqiang LI

CONTENTS

SUMMARY

Improving the efficiency in deploying deep neural networks (DNNs) and processing complex high-dimensional data has drawn increasing attention in recent years. Yet, the deployment of large DNN models is challenged by the high computational complexity and energy consumption, making it difficult to run on resource-constrained devices such as mobile phones. Moreover, the exploding amount of high-dimensional data requires large storage and transmission capacities which is infeasible to be processed on mobile devices.

To alleviate these limitations, this dissertation focuses on binarization techniques, including model binarization and data binarization, to improve the efficiency in terms of storage, computation and energy.

In model binarization, we binarize both the weight and activation of DNN models which can reach up to 32× memory saving and a speed up of 58×. We also develop pruning algorithms to further compress the binarized network while maintaining accuracy. To efficiently train the binarized networks, we discover new optimization methods that has less hyper-parameters and can improve the accuracy.

In data binarization, we propose deep hashing algorithms that learn smaller binary data representation. Deep hashing methods have become an effective technique for fast and efficient similarity search and retrieval of high-dimensional data items in large databases.

SAMENVATTING

Het verbeteren van de efficiëntie bij het implementeren van diepe neurale netwerken (DNNs) en het verwerken van complexe, hoog-dimensionale data heeft de laatste jaren steeds meer aandacht gekregen. Enerzijds wordt de implementatie van grote DNN modellen belemmerd door de hoge rekencomplexiteit en energieverbruik, waardoor het moeilijk is om ze uit te voeren op apparaten met beperkte middelen zoals mobiele telefoons. Anderzijds vereist de groeiende hoeveelheid hoog-dimensionale data grote opslag- en transmissiecapaciteiten, wat onuitvoerbaar is om te verwerken op mobiele apparaten.

Om de beperkingen te verlichten, richt het proefschrift zich op binarisatietechnieken, waaronder modelbinarisatie en databinarisatie, om de efficiëntie te verbeteren op het gebied van opslag, berekening en energie.

Bij modelbinarisatie binariseren we zowel het gewicht als de activatie van DNN modellen, wat kan leiden tot een geheugenbesparing tot wel 32× en een versnelling van de inferentie tot wel 58×. We passen ook het pruning-algoritme toe om het binariseerde netwerk verder te comprimeren, terwijl we toch de nauwkeurigheid behouden. Om de binariseerde netwerken efficiënt te trainen, hebben we nieuwe optimalisatiemethoden ontdekt die minder hyperparameters hebben en de nauwkeurigheid kunnen verbeteren.

Bij gegevensbinarisatie stellen we diepe hash-algoritmen voor om een kleinere binaire gegevensrepresentatie te leren. Diepe hash-methoden zijn een effectieve techniek geworden voor snelle en efficiënte gelijkeniszoekopdrachten en opvraging van hoog-dimensionale gegevenselementen in grote databases.

De voorgestelde begeleide hash-methode in hoofdstuk 2 behaalt een betere nauwkeurigheid in vergelijking met de onbegeleide hash-methode zoals in hoofdstuk 3, maar met als nadeel dat er dure geannoteerde labelbegeleiding door mensen nodig is.

1

INTRODUCTION

Nearly everyone now has a personal smartphone. This smartphone has a high-quality video camera, which, together with the popularity of current social media gives rise to huge quantities of high-dimensional digital data like video, images, and audio. Analyzing such large quantities of on-device media manually is difficult, and automatic tools are essential. To automatically process complex high-dimensional data, deep neural networks (DNNs) have received increasing attention in recent years because DNNs do exceptionally well in automatically extracting information from such unstructured data types. The rapid developments of DNNs have brought significant improvements in many applications including computer vision [1], machine translation [2], natural language processing [3] and generative models [4]. For instance, the top-1 classification accuracy on ImageNet has increased from 63.3% in 2012 by AlexNet [5] to 90.45% in 2022 by a current scaled vision transformer [6]. The successful development of DNNs allow to learn complex patterns and relationships from large-scaled datasets, fueling the deployment of more powerful AI applications which, ideally, can be applied on the phone of a user.

The automatic analysis capacity of DNNs, however, comes at the cost of huge amounts of computational power and built-in storage memory. For example the ResNet-152 [7] network contains about 60 million 32-bit floating-point parameters. The entire network needs to occupy more than 230 megabytes (MB) of storage space and use 11.3 billion floating point operations (FLOPs) to process a single 224x224 image. Running this ResNet-152 in recent smart-phone with 4K video recording at 30 frames/sec requires the device hardware to deliver 11.3 GFLOPs × 30 frames/sec × 3840 × 2160 /(224 x 224) = 56 TeraFLOPs/sec computational throughput. Yet, recent powerful mobile phone processors can achieve 2 TeraFLOPs/sec at peak. As this example illustrates, the computational demands make it difficult to deploy DNNs on devices with limited computational resources like a mobile phone.

Another key challenge related to large amounts of digital data is the transmission and storage consumption. For example, storing 1 hour YUV422 format video on a current Apple iPhone 14 Pro Max with 4K video recording at 30 frames/sec, requires 1,668 GB storage space. Yet, mobile devices such as smartphone have limited memory space. In addition, transmitting live video with YUV422 format from your Apple iPhone 14 Pro Max to other smartphones needs about 3.98 Gbps data rate for communication which is well beyond current 4G technology's peak data rate of up to around 0.1 Gbps.

Automatically analyzing large digital media on mobile devices with deep neural networks need efficient computations and also efficient storage.

1.1. MOTIVATION

The purpose of this thesis is to improve the efficiency of DNNs models and compress large-scale data sets to smaller, more manageable sizes. The following gives the motivations for these aspects.

DNN Model Compression. A smaller model means less required memory budget in deploying a DNN model. For example, mobile phone App services often prohibit users from downloading a mobile applications above 100 MB until the users connect to Wi-Fi while an original ResNet-152 model based mobile application is at least 230 MB. Typically the size of models deployed on current smartphones needs to be less than 5 MB. Another example is to deploy DNNs model on microcontrollers, the model size typically needs to be constrained to less than 250 KB. Thus, putting a large DNN model in the microcontrollers, which have extremely limited on-chip memory and flash storage, is infeasible.

Additionally, a smaller model helps improve inference speed. Many real world applications require low-latency and real-time inference. In this thesis, the compressed binary network models enable more efficient (bit-count) operations to speed up inference, showing great benefits on computationally limited platform.

Because inference speed is reduced, a smaller model also reduces the energy consumption to extend battery life. The energy consumption is mostly dominated by the memory access, where [8] shows that the energy consumed by both the on-chip and off-chip memory access contributes to 96% of the total energy consumption. The most expensive operation is the off-chip DRAM accesses which requires orders-of-magnitude higher energy and latency. Thus, reducing off-chip DRAM accesses can significantly improve the energy efficiency. This thesis compresses the large DNN models to fit in the on-chip SRAM memory to reduce energy cost.

Data Compression. First, a smaller data file is easy to access and store. For example, recently millions of high-dimensional images or videos generated from high resolution mobile phone cameras are uploaded to the Internet every day, which require huge amount of storage space. Compressing large data file to a smaller data file can reduce the storage consumption.

Second, a smaller data file takes less time to transmit. A high resolution image by a recent smart-phone with its main camera at 48 Megapixel (MP) needs about 1.17 Gbps data rate. Such a single high resolution image data will take long time to transmit, *e.g.*, upload or download, with a limited data transfer rate.

Third, a smaller representation of the data helps improve searching speed. The uploaded massive high-dimensional images or videos make it quite difficult to find relevant images according to different requests by users. Even a linear search over the database

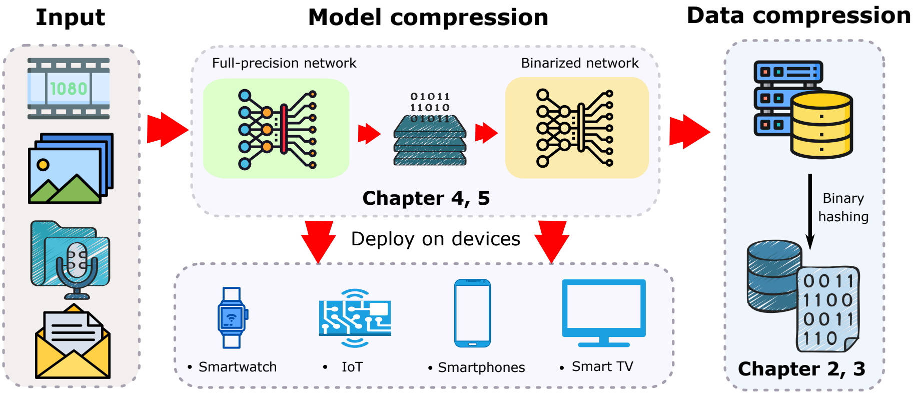


Figure 1.1: Thesis contributions, including model compression in Chapter 4, Chapter 5 and data compression in Chapter 2, Chapter 3.

would cost a huge amount of time and memory. Embedding high-dimensional data to low dimensional binary codes can benefit the low computational cost of approximate nearest-neighbour search in the Hamming space, delivering more effective large-scale data search.

Fourth, a smaller data file helps save the energy cost potentially. Wireless transmission of a single bit requires over 1000 times more energy than a single 32-bit computation operation [9]. In other word, if one bit of the data is compressed by taking less than 1000 computation operations, the energy can be saved.

1.2. CONTRIBUTION AND THESIS OUTLINE

We optimize the efficiency with centering around compression techniques. This thesis proposes the techniques for model compression and data compression, illustrated in Fig. 1.1. This thesis has the following contributions:

- **Data Compression.** This thesis proposes deep hashing algorithms for data compression. Hashing represents high-dimensional data contents to low dimensional binary codes, achieving effective improvement of speed and storage. In this thesis, Chapter 2 and Chapter 3 are on deep hasing, as visualized in Fig. 1.2.

- **Model Compression.** The thesis proposes the DNN model compression techniques that binarize both the weight and activation of DNNs as 1-bit, reaching up to 32× memory saving and 58× inference speed up. A binary network optimizer is proposed which has less hyper-parameters and more intuitive, thus makes the optimization by humans easier. Here, Chapter 4 and Chapter 5 are on binary convolutions, which is visualized in Fig. 1.3.

Chapter 2 describes a deep supervised hashing method which maps high-dimensional images onto compact binary codes with leveraging human-annotated class labels, thus reduces storage and computational cost. We maximize the binary distances between

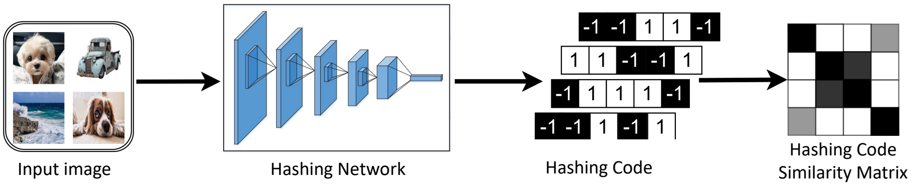


Figure 1.2: Framework of deep hashing. The hash codes are produced by a deep hashing network. We optimize the hash code within same class to be same and be different for different class.

different classes and at the same time minimize the binary distance of images within the same class to learn compact hash codes. We also show how to optimize this under discrete constraints. The content of this chapter is based primarily on [10].

Chapter 3 describes a deep unsupervised hashing method that learns compact binary codes without using any annotated label supervision. A simple parameter-free layer is designed to maximize hash channel information capacity as measured by the entropy [11]. An optimal transport cost as measured by the Wasserstein distance [12] is minimized end-to-end to align continuous features with the optimal discrete distribution. The content of this chapter is based primarily on [13].

Chapter 4 describes the binarization technique to reduce the bit of weight and activation in DNNs to 1-bit. We propose a method to add a hard constraint to binary weight distribution offering precise control for any desired bit ratio including equal prior ratios. We validate the assumption that equal bit ratios are preferable and show its optimization benefits such as search-space reduction and good minima. The content of this chapter is based primarily on [14].

Chapter 5 presents a new understanding of optimization in the context of binary neural network. In specific, we provide an analysis of standard magnitude based hyperparameters such as weight initialization, learning rate and its decay, weight decay, and momentum. The magnitude is interpretable for real-valued weights, but loses its meaning for binary weights. In the context of the latent-weight free interpretation, we interpret

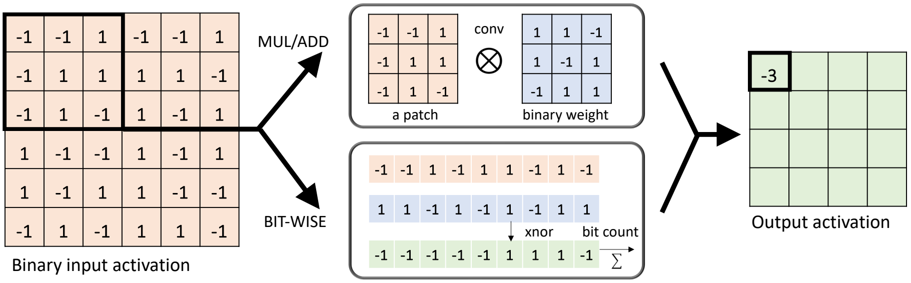


Figure 1.3: A binary convolution involves binary input and binary weights. For typical convolution operation in real-valued CNNs this is implemented by an expensive real value multiplication and addition (top row; center). Yet, when both weight and activation are binary values, the convolution operation can be implemented by a much more efficient bitwise XNOR operation and a bit-count operation (bottom row; center).

these traditional optimizer hyperparameters based on higher-order gradient filtering by an impulse response filters on the gradients during network optimization. This understanding reduces the number of hyperparameters that need to be tuned. The content of this chapter is based primarily on [15].

Chapter 6 concludes the thesis and discusses the possible research directions in the future work.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,* in *Proceedings of the IEEE international conference on computer vision* (2015) pp. 1026–1034.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need,* in *Advances in neural information processing systems* (2017) pp. 5998–6008.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding,* arXiv preprint arXiv:1810.04805 (2018).

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial nets,* in *Advances in neural information processing systems* (2014) pp. 2672–2680.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks,* Advances in neural information processing systems **25**, 1097 (2012).

[6] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, *Scaling vision transformers,* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022) pp. 12104–12113.

[7] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 770–778.

[8] A. Marchisio, M. A. Hanif, M. T. Teimoori, and M. Shafique, *Capstore: Energy-efficient design and management of the on-chip memory for capsulenet inference accelerators,* arXiv preprint arXiv:1902.01151 (2019).

[9] K. C. Barr and K. Asanović, *Energy-aware lossless data compression,* ACM Transactions on Computer Systems (TOCS) **24**, 250 (2006).

[10] Y. Li, W. Pei, J. van Gemert, *et al.*, *Push for quantization: Deep fisher hashing,* BMVC (2019).

[11] C. E. Shannon, *A mathematical theory of communication,* Bell system technical journal **27**, 379 (1948).

[12] C. Villani, *Topics in optimal transportation,* 58 (American Mathematical Soc, 2003).

[13] Y. Li and J. van Gemert, *Deep unsupervised image hashing by maximizing bit entropy,* AAAI (2021).

[14] Y. Li, S. L. Pintea, and J. C. van Gemert, *Equal bits: Enforcing equally distributed binary network weights,* AAAI (2021).

[15] J. Quist, Y. Li, and J. van Gemert., *Understanding weight-magnitude hyperparameters in training binary networks,* in *International Conference on Learning Representations (ICLR)* (2023).

2

Push for Quantization: Deep Fisher Hashing

Current massive datasets demand light-weight access for analysis. Discrete hashing methods are thus beneficial because they map high-dimensional data to compact binary codes that are efficient to store and process, while preserving semantic similarity. To optimize powerful deep learning methods for image hashing, gradient-based methods are required. Binary codes, however, are discrete and thus have no continuous derivatives. Relaxing the problem by solving it in a continuous space and then quantizing the solution is not guaranteed to yield separable binary codes. The quantization needs to be included in the optimization. In this paper we push for quantization: We optimize maximum class separability in the binary space. To do so, we introduce a margin on distances between dissimilar image pairs as measured in the binary space. In addition to pair-wise distances, we draw inspiration from Fisher's Linear Discriminant Analysis (Fisher LDA) to maximize the binary distances between classes and at the same time minimize the binary distance of images within the same class. Experimental results on CIFAR-10, NUS-WIDE and ImageNet100 show that our approach leads to compact codes and compares favorably to the current state of the art.

2.1. INTRODUCTION

Image hashing aims to map high-dimensional images onto compact binary codes where pair-wise distances between binary codes corresponds to semantic image distances, *i.e.,* Similar binary codes should have similar class labels. Binary codes are efficient to store and have low computational cost which is particularly relevant in today's big data age where huge datasets demand fast processing.

A problem in applying powerful deep learning methods for image hashing is that deep nets are optimized using gradient descent while binary codes are discrete and thus have no continuous derivatives and cannot be directly optimized by gradient descent. The current solution [1–6] is to relax the discrete problem to a continuous one, and after optimization in the continuous space, quantize it to obtain discrete codes. This approach, however, disregards the importance of the quantization, which is problematic because image class similarity in the continuous space is not necessarily preserved in the binary space, as illustrated in Fig. 2.1. The quantization needs to be included in the optimization.

In this paper we go beyond preserving semantic distances in the continuous space: We push for quantization by optimizing maximum class separability in the binary space. To do so, we introduce a margin on distances between dissimilar image pairs explicitly measured in the binary space. In addition to pair-wise distances, we draw inspiration from Fisher's Linear Discriminant Analysis (Fisher LDA) to maximize the binary distances between classes and at the same time minimize the binary distance of images within the same class

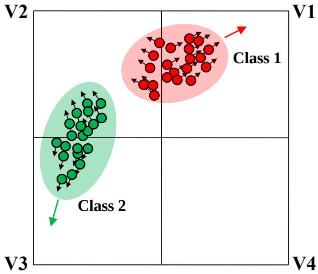


Figure 2.1: Example of two separable classes in a continuous space. After quantization (assign to grid cells) the classes are no longer separable. In this paper we aim for separability in the binary space.

We have the following contributions. 1) Adding a margin to pairwise labels pushes dissimilar samples apart in the binary space; 2) Fisher's criterion to maximize the between-class distance and to minimize the within-class distance leads to compact hash codes; 3) We show how to optimize this under discrete constraints and 4) We outperform state-of-the-art methods on two datasets, being particular advantageous for a small number of hashing bits.

2.2. RELATED WORK

Amount of supervision. Existing hashing methods can be grouped on the amount of prior domain knowledge. Hashing methods without prior knowledge are applicable to any domain and include well-known methods such as Locality-Sensitive Hashing (LSH) [7] and its extensions [8–12]. If some knowledge about the data distribution is known in the form of an unlabeled training set, this knowledge can be advantageously exploited by unsupervised methods [13–19] which learn hash functions by preserving the training set distance distribution. With the availability of additional prior knowledge about how samples should be grouped together, supervised methods [20–26] can leverage such

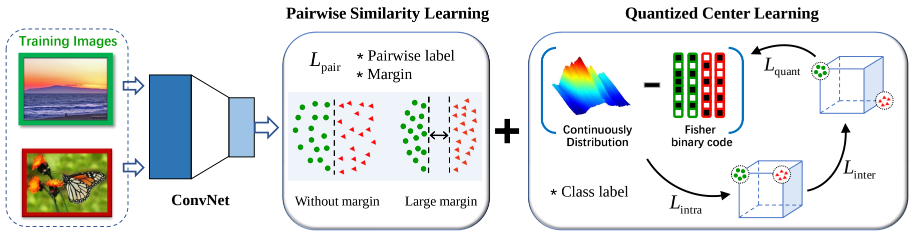


Figure 2.2: Images with class labels (red and green) are input to a CNN which outputs a k-dimensional continues representation **U**. Module 1 maximizes a margin between dissimilar images in binary space (L_{Pair}). Module 2 minimizes binary distances within the same class (L_{Intra}) and pushes different classes away (L_{Inter}) while quantizing **U** as binary codes (L_{Quant}).

label information. Particularly successful supervised hashing methods use deep learning [4, 27–30] to learn the feature representation. Supervision can be in the form of pairwise label information [1, 3, 6, 31, 32] or in the form of class labels [21, 25, 28, 32]. In this paper we exploit both pairwise and class label knowledge, leading to highly compact and discriminative hash codes.

Quantization in hashing. Several methods optimize the continue space and apply the *sign* to obtain binary codes [1–6, 18, 20, 33]. A quantization loss is proposed in deep learning based hashing [1–4, 6, 33] to force the learned continuous representations to approach the desired binary codes. However, optimizing quantization alone may not preserve class separability in the binary space. An elegant solution is to employ *sigmoid* or *tanh* to approximate the non-smooth *sign* function [31, 34], but unfortunately comes with the drawback that such activation functions have difficulty to converge when using gradient descent methods. We circumvent these limitations by imposing the quantization loss in the discrete space, optimizing the separability in the hashing space directly while guiding parameter optimization in the continuous space.

Discrete optimization. Another branch of hashing methods to solve the discrete optimization is to utilize the class information to directly learn the hashing codes. For instance, SDH [25], as well as its extensions such as FSDH [21] and DSDH [32], propose to regress the same-class images to the same binary codes. While this kind of methods encourages a close binary distance between samples from the same class, they cannot guarantee the separability of samples from different classes. In contrast, we propose to explicitly maximize the binary distances between classes and at the same time minimize the binary distances within the same class.

2.3. Deep Fisher Hashing with Pairwise Margin

In Fig. 2.2 we illustrate our model. Two components steer the discrete optimization: 1) A Pairwise Similarity Learning module to preserve semantic similarity between image pairs while using a margin to push similar and non-similar images further apart (L_{pair}). 2) A Quantized Center Learning module inspired by Fisher's linear discriminant that maximizes the distance between different-class images (L_{inter}) whilst minimizing the distance between same-class images (L_{intra}) where the binarization requires minimizing quantization errors L_{quant}. These two modules are optimized jointly on top of a convolutional network (CNN).

For a train set of N images $\mathbf{X} = \{\mathbf{x_i}\}_{i=1}^N$, with M class labels $\mathbf{Y} = \{\mathbf{y}_i\}_{i=n}^N \in \mathbb{R}^{M\times N}$, where $\mathbf{y}_i \in \mathbb{R}^M$ is a vector with all elements ≥ 0 that sums to 1, representing the class proportion of sample $\mathbf{x_i}$. For single-label (multi-class) $\mathbf{y}_i$ reverts to a one-hot encoding $\{0,1\}^M$. If $\mathbf{x}_i$ has m multiple labels, each has a value of $1/m$ in $\mathbf{y}_i$. The last layer of the CNN $\mathbf{U} = \{\mathbf{u}_i\}_{i=1}^N \in \mathbb{R}^{K\times N}$ is the learned representations of $\mathbf{X}$. The output codes $\mathbf{B} = \{\mathbf{b}_i\}_{i=1}^N \in \{-1,1\}^{K\times N}$ are the discretized binary values corresponding to $\mathbf{U}$ with each image encoded by K binary bits.

2.3.1. PAIRWISE SIMILARITY LEARNING

The main goal of hashing is to have small distances between similar image pairs and large distances between dissimilar image pairs in the binary representation. For binary vectors $\mathbf{b_i}, \mathbf{b_j} \in \{-1,1\}^K$, the Hamming distance $D_H(\mathbf{b}_i, \mathbf{b}_j) = \frac{1}{2}(K - \mathbf{b}_i^\intercal \cdot \mathbf{b}_j) = \frac{1}{4} D_E(\mathbf{b}_i, \mathbf{b}_j)$. Since K is a constant, it can be left out and we define the dissimilarity $D(\mathbf{b}_i, \mathbf{b}_j) = -\frac{1}{2}\left(\mathbf{b}_i^\intercal \cdot \mathbf{b}_j\right)$. Note that larger dissimilarity D indicates larger Hamming distance and less similarity.

Similar images should share many binary values while dissimilar images should share few binary values. Given the dissimilarity $D(\cdot,\cdot) \in (-\frac{1}{2}K, \frac{1}{2}K)$, a dissimilarity of 0 between binary vectors $\mathbf{b}_i$ and $\mathbf{b}_j$ means that half of their bits are different. To encourage more overlapping bits for similar images and less overlapping bits for dissimilar images, we add a margin m to a symmetric logistic loss centered at 0:

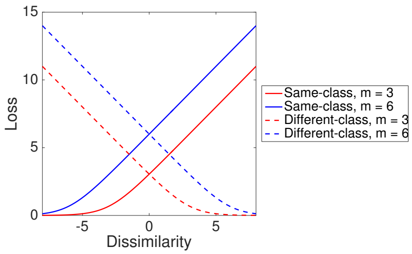


Figure 2.3: Our symmetric large margin logistic loss of both same-class and different-class cases as a function of the dissimilarity with different margin m. Larger m encourages separation.

$$L^S(D) = \log(1+e^{D+m}); L^D(D) = \log(1+e^{-D+m}). \tag{2.1}$$

The hyper-parameter $m \geqslant 0$ controls separation between similar pairs S and dissimilar pairs D. When $m = 0$, our model will turn into the classical way used in [3, 32]. Fig. 2.3 illustrates the loss curves of same-class pairs and different-class pairs as a function of dissimilarity calculated by our dissimilarity measure with various values of m. Larger margin can help to pull same-class pairs together while push different-class pairs far away.

The Pairwise Similarity module minimizes the large margin logistic loss:

$$\begin{aligned} L_{\text{pair}} = \sum_{(i,j)\in\mathscr{S}} L^S\left(D\left(\mathbf{b}_i, \mathbf{b}_j\right)\right) + \sum_{(i,j)\in\mathscr{D}} L^D\left(D\left(\mathbf{b}_i, \mathbf{b}_j\right)\right) \\ s.t.\ \ \mathbf{b}_i, \mathbf{b}_j \in \{-1,1\}^K,\ \ i,j = 1,...,N. \end{aligned} \tag{2.2}$$

Since $\mathbf{b}_i$ and $\mathbf{b}_j$ are discretized hashing codes from the continuous output of the CNN ($\mathbf{u}_i$ and $\mathbf{u}_j$), thus it is hard to back-propagate gradients from L_{pair} to parameters of the CNN. To make the CNN trainable with L_{pair}, we introduce an auxiliary variable $\mathbf{u}_i = \mathbf{b}_i$.

Then we apply Lagrange multipliers to get the Lagrangian:

$$\begin{aligned}\tilde{L}_{\text{pair}} &= \sum_{(i,j)\in\mathscr{S}} L^S(D(\mathbf{u}_i,\mathbf{u}_j)) + \sum_{(i,j)\in\mathscr{D}} L^D(D(\mathbf{u}_i,\mathbf{u}_j)) + \psi\sum_{i=1}^{N}\|\mathbf{u}_i-\mathbf{b}_i\|_2^2,\\ &s.t.\ \mathbf{b}_i,\mathbf{b}_j\in\{-1,1\}^K,\ \ i,j=1,...,N,\end{aligned} \tag{2.3}$$

where ψ is the Lagrange multiplier. The term $\sum_{i=1}^{N}\|\mathbf{u}_i-\mathbf{b}_i\|_2^2$ can be viewed as a constraint to minimize the discrepancy between the binary space and the continuous space.

2.3.2. QUANTIZED CENTER LEARNING

The Quantized Center Learning module, see Fig. 2.4, maximizes the inter-class distances whilst minimizing the intra-class distances in a quantized setting. To represent class-distances we learn a center for each of the M classes: $\mathbf{C}=\{\mathbf{c}_i\}_{i=1}^{M}\in\{-1,1\}^{K\times M}$, where each center $\mathbf{c}$ is encoded by K bits of binary codes. Let $\mathbf{u}$ be the network output representation. We then encourage the learned binary code(vertex) of each representation to be close to the corresponding class center while the distance between different class centers is maximized, taking quantization to binary vectors into account.

Minimizing intra-class distances (L_{intra}). This minimizes the sum of Euclidean distance between the binary codes $\mathbf{b}_i$ of the N training images to their class center:

$$L_{\text{intra}}=\sum_{i=1}^{N}\|\mathbf{b}_i-\mathbf{C}\mathbf{y}_i\|_2^2, \tag{2.4}$$

where all class centers $\mathbf{C}$ are indexed by $\mathbf{b}_i$'s class membership vector $\mathbf{y}_i$.

Maximizing inter-class distances (L_{inter}). We maximize the sum of pairwise Euclidean distance between different class centers to maximize the inter-class distance of training data:

$$\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}\|\mathbf{c}_i-\mathbf{c}_j\|_2^2=\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}(2K-2\mathbf{c}_i^{\mathsf{T}}\mathbf{c}_j). \tag{2.5}$$

Since $\mathbf{c}_i,\mathbf{c}_j\in\{-1,1\}^K$ and $\mathbf{c}_i^{\mathsf{T}}\mathbf{c}_{j\neq i}\geq -K$, maximizing Eq. (2.5) is equivalent to minimizing

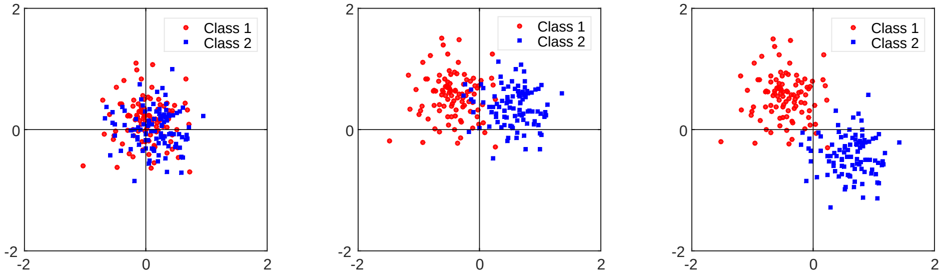
$$\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N}(\mathbf{c}_i^{\mathsf{T}}\mathbf{c}_j-(-K))^2=\|\mathbf{C}^{\mathsf{T}}\mathbf{C}-K(2I-J_K)\|_F^2, \tag{2.6}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, I is the identity matrix and J_K is the all-ones matrix. Simplifying the notation where A replaces $K(2I-J_K)$ yields

$$L_{\text{inter}}=\|\mathbf{C}^{\mathsf{T}}\mathbf{C}-A\|_F^2. \tag{2.7}$$

Minimizing quantization cost (L_{quant}). The Center Learning module exploits label information to learn binary codes by minimizing L_{intra} and L_{inter} simultaneously. We also need to encourage the learned representation to be close to the quantized binary codes. L_{quant} minimizes the total quantization cost in moving representations $\mathbf{u}_i$ towards the desired $\mathbf{b}_i$,

$$L_{\text{quant}}=\sum_{i=1}^{N}\|\mathbf{b}_i-\mathbf{u}_i\|_2^2. \tag{2.8}$$



(a): Input (b): Only L_{intra} (c): $L_{\text{intra}} + L_{\text{inter}}$

Figure 2.4: Illustration of Quantized Center learning. All points denote 2D representations extracted by a CNN model from randomly selected two classes samples of CIFAR-10, for 100 samples per class. Binarization is illustrated by quantization $sgn(\cdot)$ (black lines). *(a):* Inefficient hashing: Binarization will assign same-class points to different bins, while assigning different-class points to the same bins. *(b):* Using L_{intra} clusters classes together and hashing is improved since binarization will assign the classes to different, neighboring bins: class 1 to $[-1,1]$ and class 2 to $[1,1]$. *(c):* Using $L_{\text{intra}} + L_{\text{inter}}$ also pushes the classes away from each other, improving the hashing further since after binarization class 1 is $[-1,1]$ and class 2 is $[1,-1]$ making the difference between class samples two bit flips.

2.4. OPTIMIZATION

Our proposed Pairwise Similarity module and Quantized Center Learning module are optimized jointly in an alternating fashion where their gradients are back-propagated to train the upstream CNN. Combining the loss functions $\tilde{L}_{pair}$ in Eq. (2.3), L_{intra} in Eq. (2.4), L_{inter} in Eq. (2.7) and L_{quant} in Eq. (2.8), the optimization of the whole framework is

$$\begin{aligned}\min_{\mathbf{b}_i,\mathbf{u}_i,\mathbf{C}} \Big[& \varphi\Big(\sum_{(i,j)\in\mathcal{S}} L^S\big(D\big(\mathbf{u}_i,\mathbf{u}_j\big)\big) + \sum_{(i,j)\in\mathcal{D}} L^D\big(D\big(\mathbf{u}_i,\mathbf{u}_j\big)\big)\Big) \\ & + \mu\sum_{i=1}^{N} \|\mathbf{b}_i - \mathbf{C}\mathbf{y}_i\|_2^2 + \nu\|\mathbf{C}^\top\mathbf{C} - A\|_F^2 + \sum_{i=1}^{N}\|\mathbf{b}_i - \mathbf{u}_i\|_2^2\Big], \\ \text{s.t. } & \mathbf{C}\in\{-1,1\}^{K\times M},\ \ \mathbf{b}_i\in\{-1,1\}^K,\ \ i=1,2,\ldots,N,\end{aligned} \tag{2.9}$$

where φ, μ and ν are hyper-parameters that balance the effect of three objective functions.

Optimizing Eq. (2.9) involves the interaction of two types of variables: discrete variables $\{\mathbf{B}=\{\mathbf{b}_i\}_{i=1}^N, \mathbf{C}\}$ and continuous variables $\mathbf{U}=\{\mathbf{u}_i\}_{i=1}^N$. A typical solution to such multi-variable optimization problem is to alternate between two steps. In particular: 1) optimize $\mathbf{U}$ while fixing $\mathbf{B}$ and $\mathbf{C}$ focusing on L_{pair} in the Pairwise Similarity Learning module, 2) fixing $\mathbf{U}$ and optimize discrete variables $\mathbf{B}$ and $\mathbf{C}$ in the Quantized Center Learning.

2.4.1. OPTIMIZING PAIRWISE SIMILARITY LEARNING

Given $\mathbf{B} = \{\mathbf{b}_i\}_{i=1}^N$, it is straightforward to optimize $\mathbf{U} = \{\mathbf{u}_i\}_{i=1}^N$ by minimizing the sub-problem resolved from Eq. (2.9) corresponding to L_{pair} by gradient descent:

$$\min_{\mathbf{U}} \sum_{i=1}^{m} \|\mathbf{b}_i - \mathbf{u}_i\|_2^2 + \varphi\Big(\sum_{(i,j)\in\mathscr{S}} L^S(D\big(\mathbf{u}_i, \mathbf{u}_j\big)) + \sum_{(i,j)\in\mathscr{D}} L^D(D\big(\mathbf{u}_i, \mathbf{u}_j\big))\Big) \tag{2.10}$$

Since $\mathbf{U}$ is the output of the last layer of the upstream CNN, which is denoted as $\mathbf{u}_i = \mathbf{W}^\mathsf{T}\mathscr{F}_{\text{CNNs}}(\mathbf{x}_i;\theta) + \mathbf{v}$. Here $\mathbf{W}$ is the transformation matrix of the last fully connected layer and $\mathbf{v}$ is the bias term. θ is the parameters of CNNs before the last layer. For simplicity, we denote all parameters of CNNs models as $\Theta = \{\mathbf{W}, \mathbf{v}, \Theta\}$. The CNN parameters are optimized by gradient back-propagation: $\frac{\partial L}{\partial \Theta} = \frac{\partial L}{\partial \mathbf{U}}\frac{\partial \mathbf{U}}{\partial \Theta}$, where L is the Loss function corresponding to Eq. (2.10).

2.4.2. OPTIMIZING QUANTIZED CENTER LEARNING

With fixed CNN parameters Θ, we learn $\mathbf{B}$ and $\mathbf{C}$ by optimizing the Quantized Center Learning module, as:

$$\begin{aligned} &\min_{\mathbf{B},\mathbf{C}} \mu \sum_{i=1}^{N} \|\mathbf{b}_i - \mathbf{C}\mathbf{y}_i\|_2^2 + \nu\|\mathbf{C}^\mathsf{T}\mathbf{C} - A\|_F^2 + \sum_{i=1}^{N} \|\mathbf{b}_i - \mathbf{u}_i\|_2^2, \\ &\text{s.t. } \mathbf{C} \in \{-1,1\}^{K\times M},\ \mathbf{B} = \{\mathbf{b}_i\}_{i=1}^N \in \{-1,1\}^{K\times N}. \end{aligned} \tag{2.11}$$

We solve this problem by calling alternating optimization strategy again: optimize variables $\mathbf{B}$ and $\mathbf{C}$ by updating one variable with the other fixed.

Initialization of $\mathbf{b}_i$ and C. Given the representations $\mathbf{u}_i$, we initialize $\mathbf{b}_i$ as $\mathbf{b}_i = \text{sgn}(\mathbf{u}_i)$. In the first iteration we initialize the class centers $\mathbf{C}$ with the class mean of the output representations, later we update $\mathbf{C}$ directly.

Fix $\mathbf{b}_i$, update C. Keeping $\mathbf{b}_i$ fixed in Eq. (2.11) reduces this sub-problem to

$$\begin{aligned} &\min_{\mathbf{C}} \mu \sum_{i=1}^{N} \|\mathbf{b}_i - \mathbf{C}\mathbf{y}_i\|_2^2 + \nu\|\mathbf{C}^\mathsf{T}\mathbf{C} - A\|_F^2, \\ &\text{s.t. } \mathbf{C} \in \{-1,1\}^{K\times M}. \end{aligned} \tag{2.12}$$

Due to the discrete constraints on the class centers $\mathbf{C}$, the minimization of above problem is a discrete optimization problem which is hard to optimize directly. We introduce an auxiliary variable $\mathbf{V}$ with the constrain $\mathbf{C} = \mathbf{V}$, and adding the Lagrange multiplier, the optimization of Eq. (2.12) is:

$$\begin{aligned} &\min_{\mathbf{C},\mathbf{V}} \mu \sum_{i=1}^{N} \|\mathbf{b}_i - \mathbf{V}\mathbf{y}_i\|_2^2 + \nu\|\mathbf{V}^\mathsf{T}\mathbf{V} - A\|_F^2 + \eta\|\mathbf{C} - \mathbf{V}\|_F^2, \\ &\text{s.t.} \quad \mathbf{C} \in \{-1,1\}^{K\times M}. \end{aligned} \tag{2.13}$$

Fixing $\mathbf{V}$, since the optimal solution for $\mathbf{C}$ for minimizing $\|\mathbf{C}-\mathbf{V}\|_F^2$ is $\mathbf{C} = \text{sgn}(\mathbf{V})$, hence $\|\mathbf{C}-\mathbf{V}\|_F^2$ in Eq. (2.13) can be replaced with $\|\text{sgn}(\mathbf{V}) - \mathbf{V}\|_F^2$. Let $\mathscr{L}_2$ denote the loss function after applying Lagrange multipliers, then the gradient w.r.t. $\mathbf{V}$ is calculated as:

$$\frac{\partial \mathscr{L}_2}{\partial \mathbf{V}} = 2\mu(\mathbf{V}\mathbf{Y} - \mathbf{B})\mathbf{Y}^\mathsf{T} + 4\nu\mathbf{V}(\mathbf{V}^\mathsf{T}\mathbf{V} - A) + 2\eta(\mathbf{V} - \text{sgn}(\mathbf{V})), \tag{2.14}$$

approximating the class center **C** with the learned **V**.
Fix C, update $\mathbf{b}_i$. With the variable **C** fixed in Eq. (2.11), we optimize the binary code $\mathbf{b}_i$ with the sub-problem

$$\begin{aligned} &\min_{\mathbf{b}_i} \mu \sum_{i=1}^{N} \|\mathbf{b}_i - \mathbf{C}\mathbf{y}_i\|_2^2 + \sum_{i=1}^{N} \|\mathbf{b}_i - \mathbf{u}_i\|_2^2, \\ &\text{s.t. } \mathbf{b}_i \in \{-1,1\}^K, i = 1,\dots,N. \end{aligned} \tag{2.15}$$

We reformulate the above problem in matrix form as:

$$\begin{aligned} &\min_{\mathbf{B}} \mu\|\mathbf{B} - \mathbf{C}\mathbf{Y}\|_F^2 + \|\mathbf{B} - \mathbf{U}\|_F^2, \\ &\text{s.t. } \mathbf{B} \in \{-1,1\}^{K\times N}, \end{aligned} \tag{2.16}$$

which can be further expanded as:

$$\begin{aligned} &\min_{\mathbf{B}} \mu(\|\mathbf{B}\|_F^2 - 2\text{tr}(\mathbf{B}^\top\mathbf{C}\mathbf{Y}) + \|\mathbf{C}\mathbf{Y}\|_F^2) \\ &\quad + (\|\mathbf{B}\|_F^2 - 2\text{tr}(\mathbf{B}^\top\mathbf{U}) + \|\mathbf{U}\|_F^2) \\ &\text{s.t. } \mathbf{B} \in \{-1,1\}^{K\times N}. \end{aligned} \tag{2.17}$$

Since $\|\mathbf{B}\|_F^2$, $\|\mathbf{C}\mathbf{Y}\|_F^2$ and $\|\mathbf{U}\|_F^2$ are constant, the problem is equivalent to:

$$\begin{aligned} &\min_{\mathbf{B}} -\big(\mu 2\text{tr}(\mathbf{B}^\top\mathbf{C}\mathbf{Y}) + 2\text{tr}(\mathbf{B}^\top\mathbf{U})\big) \\ &\text{s.t. } \mathbf{B} \in \{-1,1\}^{K\times N}. \end{aligned} \tag{2.18}$$

Minimizing above problem is equivalent to maximizing

$$\text{tr}\big(\mathbf{B}^\top(\mu\mathbf{C}\mathbf{Y} + \mathbf{U})\big) = \sum_{i=1}^{N}\sum_{j=1}^{K} B_{ij}\mathscr{F}_{ij}, \tag{2.19}$$

where $\mathscr{F}_{ij}$ is the elements of $\mathscr{F} = \mu\mathbf{C}\mathbf{Y} + \mathbf{U}$. To maximize Eq. (2.19) with respect to **B**, B_{ij} should be optimized to be 1 when $\mathscr{F}_{ij} \geq 0$, and -1 otherwise. Thus, we can derive the closed-form solution of the problem (2.19):

$$\mathbf{B} = \text{sgn}(\mu\mathbf{C}\mathbf{Y} + \mathbf{U}). \tag{2.20}$$

By defining $\mathscr{F} = \mu\mathbf{C}\mathbf{Y} + \mathbf{U}$ as the Fisher's transformed representations, we note that $\mathscr{F}$ is a *translation* transformation of original representations **U** which pushes different-class points to different vertex and pulls same-class points to same vertex, while $\mathscr{F}$ does not change the relative position between same class. The learned center **C** determines where the corresponding class translates to. The 2D example in Fig. 2.4 shows that the shape within a class does not change, yet the classes do translate.

2.4.3. JOINT OPTIMIZATION

We update the two modules jointly, which is shown in Algorithm 1. In each iteration, the Pairwise Similarity Learning module and Quantized Center Learning module are optimized in an alternating way to learn the continuous variable **U** and discrete variables $\{\mathbf{B},\mathbf{C}\}$, respectively.

Algorithm 1: Optimization of our framework

```
Input: Training data: ${\mathbf{x}_i, \mathbf{y}_i}$, iterative number: $R$, parameters: $\mu, \varphi, \nu$, bits length:
    $K$, batch size: $m$;
Output: Optimized network parameters: $\Theta = \{\mathbf{W}, \mathbf{v}, \theta\}$ ;
Randomly initialize $\mathbf{W}, \mathbf{v}$, initialize $\theta$ with pre-trained model, initialize $\mathbf{C}$ with the
  class mean of the output representations;
for $r = 1, 2, ..., R$ do
    Step1:(Pairwise Similarity Learning);
        Optimize CNN parameters in optimizing pairwise similarity learning ;
    Step2:(Quantized Center Learning);
        Initialize: $\mathbf{b}_i = \text{sgn}(\mathbf{u}_i)$, iteration: $iter$ ;
        Fix $\mathbf{b}_i$, update C;
            Introduce an auxiliary variable: V;
            for iter = $1, 2, ..., iter$;
                Obtain the gradients;
            return V;
            Center C is approximated with V, C = V;
        Fix C, update $\mathbf{b}_i$;
            Compute $\mathbf{b}_i$ using Eq. (2.20);
            Fisher binary code, $\mathbf{B} = \text{sgn}(\mu\mathbf{CY} + \mathbf{U})$;
return $\Theta$;
```

2.5. EXPERIMENTS

Datasets. We conduct experiments on three datasets: CIFAR-10, NUS-WIDE and ImageNet100. CIFAR-10 consists of 60k color images with the resolution of 32 × 32 categorized into 10 classes. Each image has a single label. NUS-WIDE is a multi-label dataset, which contains 269,648 color images collected from Flickr. There are 81 classes, where each image is annotated with one or multiple class labels. Following [18, 32, 34], we use a subset of 195,834 images associated with 21 most frequent classes (concepts) for evaluation, among which 105,972 images has more than two labels and 89,862 images have a single label. Each class contains at least 5,000 samples. ImageNet100 consists of 130K single labelled images from 100 categories, which is a subset of the large benchmark ImageNet [35].

Experimental settings. For CIFAR-10 and NUS-WIDE datasets, we conduct experiments under two comparison settings: a small-data setting and a large-data setting. In the *small-data* setting, following previous work [3, 32], 1000 images (100 images per class) in CIFAR-10 are randomly selected to form the test query set and 500 images per class (5,000 images in total) as the training set. For NUS-WIDE, we randomly select 100 images per class (for a total of 2,100 images in 21 classes) as test queries and 500 images per class (10,500 images in total) as the training set. For pairwise ground truth labels we consider two images sharing at least one common label as similar and otherwise dissimilar. In the *large-data* experimental setting, all 1,000 images per class (10,000 images in total) in CIFAR-10 are used as the test query set while all the remaining 50,000 images are used as

Table 2.1: Comparative results for our model with different components of the Quantized Center Learning module on CIFAR-10 and ImageNet100 . We start with the Pairwise Similarity Learning (L_{pair}) and augment incrementally with two components: L_{Intra} in Eq. (2.4) and L_{Inter} in Eq. (2.7). For 24-bits in CIFAR-10 the performance seems already saturated; for all other settings, each added component brings an advantage.

	Components		CIFAR-10		ImageNet100	
Baseline	$L_{\textbf{Intra}}$	$L_{\textbf{Inter}}$	12 Bits	24 Bits	16 Bits	48 Bits
L_{pair}	×	×	0.730	0.787	0.431	0.572
	✓	×	0.746	0.802	0.543	0.696
	✓	✓	0.772	0.809	0.576	0.726

Table 2.2: MAP@1K results on for Different Number of Bits ImageNet100 using AlexNet. Our model achieves the best performance on all bits except for the 16 bits.

	ImageNet100 (mAP@1K)			
Method	16 Bits	32 Bits	48 Bits	64 Bits
CNNH [29]	0.281	0.450	0.525	0.554
NINH [27]	0.290	0.461	0.530	0.565
DHN [6]	0.311	0.472	0.542	0.573
HashNet [31]	0.506	0.630	0.663	0.683
Greedy Hash [38]	**0.625**	0.662	0.682	0.688
Ours	0.590	**0.697**	**0.726**	**0.747**

the training set. In NUS-WIDE, we randomly sample 100 images per class (2,100 images in total) as the test query set and use all remaining 193,734 images as the training set. Following the settings in [31], we sample 100 images per class for ImageNet to construct training set, and all the images in the validation set are used as the test set.

Evaluation metrics. We evaluate retrieval performance using: mean Average Precision (MAP), precision of the top N returned examples (P@N), Precision-Recall curves (PR) and Recall curves (R@N). All compared methods use identical training and test sets for fair comparison. For NUS-WIDE, we adopt MAP@5000 and MAP@50000 for the small-data setting and large-data setting, respectively. We show the results of MAP@1000 for ImageNet100.

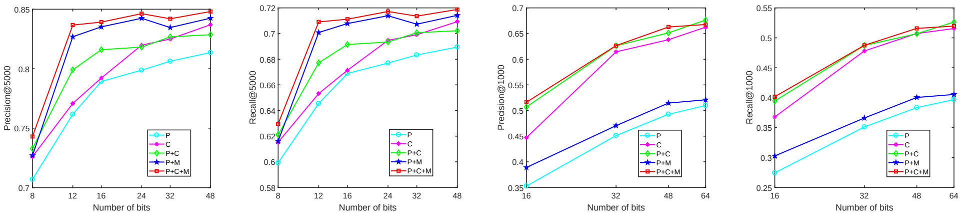
Network and parameter settings. To have a fair comparison with previous methods [3, 32, 36], we fine-tune the VGG-F [3, 32] architecture for the experiments on CIFAR-10 and NUS-WIDE while the AlexNet architecture [37] is fine-tuned for the experments on ImageNet100. Both deep network architectures are pre-trained on ImageNet. The hyper-parameters $\{\varphi, \mu, \eta, \nu, \}$ are tuned by cross-validation on a validation set and the margin m is chosen from $\{0.5, 1, 1.5, 2\}$. Stochastic Gradient Descent (SGD) is used for optimization.

2.5.1. EFFECT OF QUANTIZED CENTER LEARNING

To investigate the effect of L_{Intra} (minimizing intra-class distances) and L_{Inter} (maximizing inter-class distances) in the Quantized Center Learning module, we conduct an ablation study in the small-data setting which starts with the Pairwise Similarity Learning module L_{pair} in Eq. (2.3) in the model and then augment the model incrementally with L_{intra} in Eq. (2.4) and L_{Inter} in Eq. (2.7). In Table 2.1 we show the experimental results. We observe that both L_{Intra} and L_{Inter} contribute substantially to the performance of the whole model.

2.5.2. FUNCTIONALITY OF DIFFERENT MODULES

We evaluate the effect of combining modules on both CIFAR-10 and ImageNet100 datasets using precision and recall curves for top 5,000 returned images for different number of bits. In Fig. 3.5 we compare on CIFAR-10 and ImageNet100. We observe that each mod-



CIFAR-10

ImageNet100

Figure 2.5: Evaluating different modules on two datasets. Herein **P** refers to the Pairwise Similarity Learning module without margin while **C** refers to the Quantized Center Learning module. **P** + **M** denotes the Pairwise Similarity Learning module with tuned margin.

Table 2.3: MAP for various methods for the small-data setting for CIFAR-10 and NUS-WIDE. The best performance is boldfaced. For NUS-WIDE, the top 5,000 is used for the MAP.

Method	CIFAR-10				Method	NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits		12 bits	24 bits	32 bits	48 bits
Ours	**0.803**	**0.825**	**0.831**	**0.844**	Ours	**0.795**	**0.823**	**0.833**	**0.842**
DSDH [32]	0.740	0.786	0.801	0.820	DSDH [32]	0.776	0.808	0.820	0.829
Greedy Hash [38]	0.774	0.795	0.810	0.822	Greedy Hash [38]	–	–	–	–
DPSH [3]	0.713	0.727	0.744	0.757	DPSH [3]	0.752	0.790	0.794	0.812
DQN [39]	0.554	0.558	0.564	0.580	DQN [39]	0.768	0.776	0.783	0.792
DTSH [36]	0.710	0.750	0.765	0.774	DTSH [36]	0.773	0.808	0.812	0.824
NINH [27]	0.552	0.566	0.558	0.581	NINH [27]	0.674	0.697	0.713	0.715
CNNH [29]	0.439	0.511	0.509	0.522	CNNH [29]	0.611	0.618	0.625	0.608

ule adds value. The only exception is Fisher-only, which outperforms the combined Pairwise+Fisher model for a code size of 48. Second, the combined models can get relatively well for fewer bits, while the single models need more bits to achieve the same performance.

The results on ImageNet100 shown in Fig. 3.5 indicate that the Quantized Center Learning module improves the performance substantially. One potential explanation is that the Pairwise Similarity Learning module ($\tilde{L}_{\text{pair}}$) is sensitive to the balance between the positive and negative training sample pairs, which is hard to achieve in the data with large number of classes. In contrast, the Quantized Center Learning module does not suffer from this limitation. The sensitivity of the margin m is in the supplemental.

2.5.3. COMPARISON WITH OTHERS

In Table 2.3 we show results on both CIFAR-10 and NUS-WIDE datasets in the small-data setting. In particular for a few number of bits, our model compares well to others. It is worth noting that the performance comparison among VGG-F and AlexNet networks is considered to be fair [38], since both architectures have the same network composition.

The state-of-the-art DSDH [32] model also uses pairwise labels and classification labels. The major difference between is in using the classification label: DSDH [32] learns hash codes by maximizing the classification performance while our model learns cen-

Table 2.4: MAP for various methods for the large-data setting for CIFAR-10 and NUS-WIDE. The best performance is boldfaced. For NUS-WIDE, top 50,000 returned neighbors are considered for calculating MAP.

Method	CIFAR-10				Method	NUS-WIDE			
	16 bits	24 bits	32 bits	48 bits		16 bits	24 bits	32 bits	48 bits
Ours	**0.948**	**0.949**	**0.949**	**0.951**	Ours	**0.831**	**0.831**	**0.835**	**0.839**
DSDH [32]	0.935	0.940	0.939	0.939	DSDH [32]	0.815	0.814	0.820	0.821
Greedy Hash [38]	0.942	0.943	0.943	0.944	Greedy Hash [38]	–	–	–	–
DTSH [36]	0.915	0.923	0.925	0.926	DTSH [36]	0.756	0.776	0.785	0.799
DPSH [3]	0.763	0.781	0.795	0.807	DPSH [3]	0.715	0.722	0.736	0.741
VDSH [5]	0.845	0.848	0.844	0.845	VDSH [5]	0.545	0.564	0.557	0.570
DRSCH [40]	0.615	0.622	0.629	0.631	DRSCH [40]	0.618	0.622	0.623	0.628
DSCH [40]	0.609	0.613	0.617	0.620	DSCH [40]	0.592	0.597	0.611	0.609
DSRH [41]	0.608	0.611	0.617	0.618	DSRH [41]	0.609	0.618	0.621	0.631

ters to model between-class and between-sample distances. While DSDH performs excellent, our model outperforms DSDH in all experiments.

Another interesting observation is that SDH [25], which is based on sole classification label information, performs competitively on NUS-WIDE but not as good on CIFAR-10. In contrast, our model and DSDH [32] that leverage two types of information, perform much more robust. It reveals the necessity of incorporating the pairwise label information.

Table 2.4 shows results on CIFAR-10 and NUS-WIDE in the large-data setting. Our model performs slightly better then others, the relative improvement is smaller compared to the small-data setting.

We also conduct experiments to compare our method to other baseline models on ImageNet100 and the results are presented in Table 2.2. It is observed that our model achieves the best performance on all bits except for the 16 bits.

2.6. CONCLUSION

We present a supervised deep binary hashing method focusing on binary separability through a pair-wise margin and inspired by Fisher's linear discriminant which minimizes within-class distances while maximizing between-class distances. For medium-sized datasets with much training data –where larger hash codes can be used– our method performs on par or only slightly better than other methods. Our method is most suitable for extremely large datasets with few training data where only tiny bit codes can be used; there our method compares most favorably to others.

REFERENCES

[1] Y. Cao, M. Long, L. Bin, and J. Wang, *Deep cauchy hashing for hamming space retrieval,* in *CVPR* (2018).

[2] Q.-Y. Jiang and W.-J. Li, *Deep cross-modal hashing,* in *CVPR* (2017).

[3] W.-J. Li, S. Wang, and W.-C. Kang, *Feature learning based deep supervised hashing with pairwise labels,* in *IJCAI* (2016).

[4] H. Liu, R. Wang, S. Shan, and X. Chen, *Deep supervised hashing for fast image retrieval,* CVPR (2016).

[5] Z. Zhang, Y. Chen, and V. Saligrama, *Efficient training of very deep neural networks for supervised hashing,* in *CVPR* (2016).

[6] H. Zhu, M. Long, J. Wang, and Y. Cao, *Deep hashing network for efficient similarity retrieval,* in *AAAI* (2016).

[7] A. Gionis, P. Indyk, and R. Motwani, *Similarity search in high dimensions via hashing,* in *Proceedings of International Conference on Very Large Databases* (2000) pp. 518–529.

[8] M. Datar and P. Indyk, *Locality-sensitive hashing scheme based on p-stable distributions,* in *Proceedings of the ACM Symposium on Computational Geometry* (ACM Press, 2004) pp. 253–262.

[9] B. Kulis and K. Grauman, *Kernelized locality-sensitive hashing for scalable image search,* in *ICCV* (2009).

[10] B. Kulis, P. Jain, and K. Grauman, *Fast similarity search for learned metrics,* IEEE Transactions on Pattern Analysis Machine Intelligence **31**, 2143 (2009).

[11] Y. Mu and S. Yan, *Non-metric locality-sensitive hashing,* in *AAAI* (2010).

[12] M. Raginsky, *Locality-sensitive binary codes from shift-invariant kernels,* (2009).

[13] Y. Gong and S. Lazebnik, *Iterative quantization: A procrustean approach to learning binary codes,* in *CVPR* (2011).

[14] K. He, F. Wen, and J. Sun, *K-means hashing: An affinity-preserving quantization method for learning binary compact codes,* in *CVPR* (2013).

[15] Q. Y. Jiang and W. J. Li, *Scalable graph hashing with feature transformation,* in *International Conference on Artificial Intelligence* (2015).

[16] W. Kong and W. J. Li, *Isotropic hashing,* in *NIPS* (2012).

[17] W. Liu, S. Kumar, S. Kumar, and S. F. Chang, *Discrete graph hashing,* in *NIPS* (2014).

[18] W. Liu, J. Wang, and S. fu Chang, *Hashing with graphs,* in *ICML* (2011).

[19] Y. Weiss, A. Torralba, and R. Fergus, *Spectral hashing,* in *NIPS* (2008).

[20] S. F. Chang, *Supervised hashing with kernels,* in *CVPR* (2012).

[21] J. Gui, T. Liu, Z. Sun, D. Tao, and T. Tan, *Fast supervised discrete hashing.* IEEE Transactions on Pattern Analysis Machine Intelligence **PP**, 1 (2018).

[22] G. Lin, C. Shen, Q. Shi, A. V. D. Hengel, and D. Suter, *Fast supervised hashing with decision trees for high-dimensional data,* in *CVPR* (2014).

[23] M. Norouzi and D. J. Fleet, *Minimal loss hashing for compact binary codes,* in *ICML* (2011).

[24] R. Raziperchikolaei and M. A. Carreira-Perpiñán, *Optimizing affinity-based binary hashing using auxiliary coordinates,* in *NIPS* (2016).

[25] F. Shen, C. Shen, W. Liu, and H. T. Shen, *Supervised discrete hashing,* in *CVPR* (2015).

[26] P. Zhang, W. Zhang, W. J. Li, and M. Guo, *Supervised hashing with latent factor models,* in *SIGIR* (2014).

[27] H. Lai, Y. Pan, Y. Liu, and S. Yan, *Simultaneous feature learning and hash coding with deep neural networks.* in *CVPR* (2015).

[28] H. Liu, R. Wang, S. Shan, and X. Chen, *Learning multifunctional binary codes for both category and attribute oriented retrieval tasks,* in *CVPR* (2017).

[29] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, *Supervised hashing for image retrieval via image representation learning,* in *AAAI* (2014).

[30] T. Yao, F. Long, T. Mei, and Y. Rui, *Deep semantic-preserving and ranking-based hashing for image retrieval,* in *IJCAI* (2016).

[31] Z. Cao, M. Long, J. Wang, and P. S. Yu, *Hashnet: Deep learning to hash by continuation, ICCV,* (2017).

[32] Q. Li, Z. Sun, R. He, and T. Tan, *Deep supervised discrete hashing,* in *NIPS* (2017).

[33] F. Zhao, Y. Huang, L. Wang, and T. Tan, *Deep semantic ranking based hashing for multi-label image retrieval,* in *CVPR* (2015).

[34] H. Lai, Y. Pan, Y. Liu, and S. Yan, *Simultaneous feature learning and hash coding with deep neural networks,* in *CVPR* (2015).

[35] J. Deng, W. Dong, R. Socher, and L. J. Li, *Imagenet: A large-scale hierarchical image database,* in *CVPR* (2009).

[36] X. Wang, Y. Shi, and K. M. Kitani, *Deep supervised hashing with triplet labels,* Asian Conference on Computer Vision (2016).

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks,* in *NIPS* (2012).

[38] S. Su, C. Zhang, K. Han, and Y. Tian, *Greedy hash: Towards fast optimization for accurate hash coding in cnn,* in *Advances in Neural Information Processing Systems* (2018) pp. 798–807.

[39] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen, *Deep quantization network for efficient image retrieval,* in *AAAI* (2016).

[40] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang, *Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification,* IEEE Trans. Image Processing **24**, 4766 (2015).

[41] F. Zhao, Y. Huang, L. Wang, and T. Tan, *Deep semantic ranking based hashing for multi-label image retrieval,* in *CVPR* (2015).

3

DEEP UNSUPERVISED IMAGE HASHING BY MAXIMIZING BIT ENTROPY

Unsupervised hashing is important for indexing huge image or video collections without having expensive annotations available. Hashing aims to learn short binary codes for compact storage and efficient semantic retrieval. We propose an unsupervised deep hashing layer called Bi-half Net that maximizes entropy of the binary codes. Entropy is maximal when both possible values of the bit are uniformly (half-half) distributed. To maximize bit entropy, we do not add a term to the loss function as this is difficult to optimize and tune. Instead, we design a new parameter-free network layer to explicitly force continuous image features to approximate the optimal half-half bit distribution. This layer is shown to minimize a penalized term of the Wasserstein distance between the learned continuous image features and the optimal half-half bit distribution. Experimental results on the image datasets Flickr25k, Nus-wide, Cifar-10, Mscoco, Mnist and the video datasets Ucf-101 and Hmdb-51 show that our approach leads to compact codes and compares favorably to the current state-of-the-art.

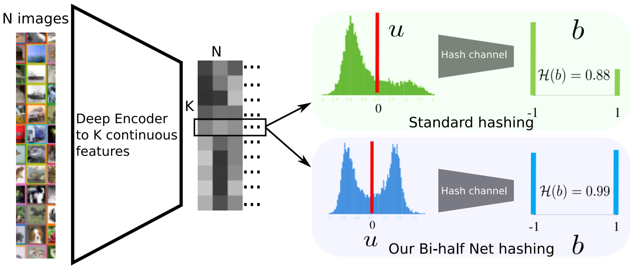


Figure 3.1: Hashing compresses N images to K bits per image. K continuous features are learned, and thresholded to K binary values. We see the continuous to binary transition as a lossy communication channel –a hash channel– between a single continuous value u to a single discrete binary value b. The green histograms (standard hashing) and blue histograms (our approach) show how a single feature is distributed over the N images. Instead of adding an additional loss term, we design a Bi-half layer to explicitly maximizes the bit capacity in the hash channel, leading to more informative hash codes, as measured by the entropy $\mathscr{H}$ of the bits over the images which leads to improved hashing accuracy.

3.1. Introduction

Semantically similar images or videos can be found by comparing their output features in the last layer of a deep network. Such features are typically around 1,000 continuous floating point values [1], which is already too slow and large for moderately sized datasets of a few million samples. Speed and storage are greatly improved by replacing the continuous features with just a small number of bits. Unsupervised hashing aims to learn compact binary codes that preserves semantic similarity without making use of any annotated label supervision and is thus of great practical importance for indexing huge visual collections.

In this paper, as illustrated in Fig. 3.1, we see the transition from a continuous variable to a discrete binary variable as a lossy communication channel. The capacity of a hash bit as measured by the entropy is maximized when it is half-half distributed: Half of the images are encoded with -1 and the other half of the images is encoded with $+1$. We minimize the information loss in the hash channel by forcing the continuous variable to be half-half distributed. Other methods have optimized entropy by adding an additional term to the loss [2–6] which adds an additional hyper-parameter to tune and is difficult to optimize. Instead, we propose Bi-half: A new parameter-free network layer which is shown to minimize the optimal transport cost as measured by the Wasserstein distance. We here explicitly design a new layer to maximize the bit capacity in the hash channel, leading to compact and informative hash codes which yield excellent hashing similarity accuracy.

We have the following contributions. 1) A simple, parameter-free, bi-half layer to maximize hash channel information capacity; 2) A Wasserstein distance is minimized end-to-end to align continuous features with the optimal discrete distribution; 3) We study 2 alternatives to maximizing bit entropy using an additional term in the loss; 4) We show state-of-the-art results for unsupervised hashing on 5 image datasets, and 2 video datasets.

3.2. RELATED WORK

Amount of supervision. Hashing methods can be grouped into data-independent hashing methods and data-dependent hashing methods. Data-independent hashing methods [7–12] design hash function independent of a dataset. In contrast, data-dependent hashing methods can exploit the data distribution. As such, with the availability of labeled training data, supervised hashing methods [13–18] learn hash codes by optimizing class labels. Particularly successful supervised image hashing methods use deep learning [14, 19–26] to learn feature representations and binary codes. Supervised methods work well, yet rely on data annotations done by humans, which are expensive or difficult to obtain. Unsupervised hashing methods [3, 5, 27–31] skip this problem, as they do not rely on annotation labels. Recent unsupervised hashing methods rely on deep learning for representation learning [2, 4, 32–35]. We follow these works and focus on the unsupervised setting.

Quantization from continuous to discrete values. The typical approach for deep learning hashing is to optimize a continuous output and in the last step quantize the continuous values to discrete values. The current approach [36–38] is to apply a sign function, where all negative values are set to -1 and all positive values are set to $+1$. We argue that the sign function is not information efficient. For example, we set the continuous features of one dimension for 4 images to be $[0.2, 0.8, 1.5, 3]$. Passing them through the sign function will binarize them all to same value $+1$ and thus the bit has no discriminative information for these 4 images. In this paper we focus on this loss of information and learn to discretize based on maximum bit capacity over images.

Obtaining gradients for binary codes. A major challenge of learning hash codes with deep nets is that the desired discrete hash output codes have no continuous derivatives and cannot be directly optimized by gradient descent. By the continuous relaxation [19, 23, 39, 40], a continuous space is optimized instead and the continuous values are quantized to binary codes. Such methods are approximations as they do not optimize the binary codes directly. The continuation based hashing methods [20, 41] gradually approximate the non-smooth *sign* function with *sigmoid* or *tanh*, but unfortunately comes with the drawback that such relaxation inevitably becomes more non-smooth during training which slows down convergence, making it difficult to optimize. To overcome these problems, a recent simple and efficient method called greedy hash [38], uses the sign function in the forward pass to directly optimize binary codes. The optimization is done with the straight-through estimator [42] which after quantization computes gradients by simply ignoring the quantization function during training. This optimization is simple and works well in practice. Yet, it ignores bit information capacity and thus may lead to redundant codes. In this work we use the same straight-through estimator to obtain gradients for binary codes while focusing on maximizing bit information capacity to obtain compact and discriminative hash codes.

Information theory in hashing. Many popular unsupervised feature learning methods [43–45] are based on information theory to find good features. In hash learning, some methods [2–6] proposed to add an additional term in the loss function to encourage each bit to have a 50% chance of being one or zero, to maximize bit entropy. It is, however, difficult to balance the added loss term with other terms in the loss, which requires careful hyper-parameter tuning of how much to weight each term in the loss.

Instead of adding an additional loss term and an additional hyper-parameter, we design a new network layer without any additional parameters to explicitly force continuous image features to approximate the optimal half-half bit distribution. Some non-deep learning approaches [46, 47] directly threshold the learned feature vectors at their median point, which have shown excellent performance. Yet, it is a suboptimal solution under deep learning scenario since the median point should be dynamically adapted to random sample statistic computed over each minibatch. We are inspired by their works, and aim to generalize such ideas to an end-to-end deep learnable setting. We cast it into an optimal transport problem and directly quantize the continuous features into half-half distributed binary codes by minimizing the Wasserstein distance between the continuous distribution and a prior half-half distribution.

3.3. Approach

This paper we maximize the hash channel capacity to design a parameter-free bi-half coding layer. We will first introduce some notations. Let $\mathbf{X} = \{\mathbf{x_i}\}_{i=1}^{N}$ denote N training images. The images are encoded to K compact binary codes $\mathbf{B} \in \{1, -1\}^{N \times K}$, which also denotes the output of our hash coding layer. $\mathbf{U} \in \mathbb{R}^{N \times K}$ would be expressed as the continuous feature representations in the last layer of a standard neural network, *e.g.*, an encoder, which serves as the input to our hash coding layer.

3.3.1. Maximizing hash channel capacity

We see the transition from a continuous variable U to a binary code variable B as a lossy communication channel. Per channel, the *maximum* transmitted information from continuous variable U to binary variable B, known as channel capacity [48, 49], is:

$$C = \max_{p(u)} I(U;B), \tag{3.1}$$

where the maximum is taken over all possible input distributions $p(u)$ and $I(U;B)$ denotes mutual information between variable U and binary variable B. We aim to maximize the channel capacity. To maximize channel capacity C we first rewrite the mutual information term $I(U;B)$ in Eq. (3.1) in terms of entropy:

$$I(U;B) = \mathscr{H}(B) - \mathscr{H}(B|U), \tag{3.2}$$

where $\mathscr{H}(B)$ and $\mathscr{H}(B|U)$ denote entropy and conditional entropy respectively. Thus, maximizing channel capacity C in Eq. (3.1) is equivalent to maximizing the entropy $\mathscr{H}(B)$ of B and minimizing the conditional entropy $\mathscr{H}(B|U)$.

The entropy $\mathscr{H}(B)$ in Eq. (3.2) should be maximized. Since B is a discrete binary variable, its entropy is maximized when it is half-half distributed:

$$p(B = +1) = p(B = -1) = \frac{1}{2}. \tag{3.3}$$

The conditional entropy $\mathscr{H}(B|U)$ in Eq. (3.2) should be minimized. Give a certain continuous value u, the transmission probability $p_u(\text{pos})$ is defined as how probable a

+1 binary output value is and the transmission probability $p_u(\text{neg})$ is defined as how probable the binary value -1 is. These are probabilities and thus are non-negative and sum to one as $p_u(\text{pos}) + p_u(\text{neg}) = 1$ and $0 \le p_u(\text{pos}), p_u(\text{neg}) \le 1$. Then the conditional entropy is computed as:

$$\begin{aligned}\mathcal{H}(B|U) &= \int_{u\in\mathcal{U}} p(u)\mathcal{H}(B|U=u)du \\ &= -\int_{u\in\mathcal{U}} p(u)\Big(p_u(\text{pos})\log p_u(\text{pos}) \\ &\qquad + p_u(\text{neg})\log p_u(\text{neg})\Big)du,\end{aligned} \tag{3.4}$$

which is between 0 and 1, and Eq. (3.4) is thus minimized for setting the $\mathcal{H}(B|U=u)$ to 0, *i.e.*: $-\int_{u\in\mathcal{U}} p(u)0\ du = 0$. This minimum is obtained when either $p_u(\text{pos}) = 1$ or $p_u(\text{neg}) = 1$, which means that there is no stochasticity for a certain continuous value u, and its binary value is deterministically transmitted.

In the following, we maximize the entropy of binary variables by encouraging the continuous feature distribution $p(u)$ to align with the ideal half-half distributed distribution $p(b)$ in Eq. (3.3). To minimize Eq. (3.4), we first start with a non-deterministic transmission probability during training, but since we train to align $p(u)$ with the half-half distribution of +1 and −1, this allows us at test time to simply use the sign function as a deterministic function for quantization to guarantee minimizing Eq. (3.4).

3.3.2. BI-HALF LAYER FOR QUANTIZATION

To align the continuous feature distribution with the ideal prior half-half distributed distribution from Eq. (3.3) we use Optimal Transport (OT) [50]. Optimal Transport aims to find a minimal cost plan for moving one unit of mass from one location $\mathbf{x}$ to one other location $\mathbf{y}$ between two probability distributions $\mathbb{P}_r$ and $\mathbb{P}_g$. When $\mathbb{P}_r$ and $\mathbb{P}_g$ are only accessible through discrete samples, the corresponding optimal transport cost can be defined as:

$$\pi_0 = \min_{\pi\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \langle \pi, \mathbf{D}\rangle_F, \tag{3.5}$$

where $\Pi(\mathbb{P}_r,\mathbb{P}_g)$ is the space of joint probability measures with marginals $\mathbb{P}_r$ and $\mathbb{P}_g$, and π is the general probabilistic coupling that indicates how much mass is transported to push distribution $\mathbb{P}_r$ towards distribution $\mathbb{P}_g$. The $\langle .,. \rangle_F$ denotes the Frobenius dot product, and $\mathbf{D} \ge 0$ is the cost function matrix whose element $D(i,j) = d(\mathbf{x},\mathbf{y})$ denotes the non-negative cost to move a probability mass from location $\mathbf{x}$ to location $\mathbf{y}$. When the cost is defined as a distance, OT is referred to as a Wasserstein distance. Specifically, if $d(\mathbf{x},\mathbf{y})$ is the squared Euclidean distance, it is the Earth mover's distance, which is also known as the 1-Wasserstein distance. We optimize the 1-Wasserstein distance because it is flexible and easy to bound.

With a randomly sampled mini-batch of M samples, the corresponding empirical distributions of the continuous variable U and binary variable B, P_u and P_b, can be written as:

$$P_u = \sum_{i=1}^{M} p_i \delta_{u_i}, \quad P_b = \sum_{j=1}^{2} q_j \delta_{b_j}, \tag{3.6}$$

where $\delta_{\mathbf{x}}$ is the Dirac function at location $\mathbf{x}$. The p_i and q_j are the probability mass associated to the corresponding location u_i and b_j, where the total mass is one, *i.e.*: $\sum_{i=1}^{M} p_i = 1$ and $\sum_{j=1}^{2} q_j = 1$. Particularly, a binary variable only has two locations b_1 and b_2, with the corresponding mass q_1 and q_2.

For the ideal prior half-half distribution in Eq. (3.3), the probability mass q_1 at location b_1 is equal to the probability mass q_2 at location b_2 that is $q_1 = q_2 = \frac{1}{2}$. The hash coding strategy is to find the optimal transport coupling π_0 by minimizing the 1-Wasserstein distance $W_1(P_u, P_b)$:

$$\pi_0 = \min_{\pi \in \Pi(P_u, P_b)} \sum_i \sum_j \pi_{ij} (u_i - b_j)^2, \tag{3.7}$$

where $\Pi(P_u, P_b)$ is the set of all joint probability distributions π_{ij}, *i.e.* all probabilistic couplings, with marginals P_u and P_b, respectively.

By optimizing Eq. (3.7), we find an optimal transport plan $\pi_0 \in \Pi(P_u, P_b)$ for one hash bit to quantize the encoded features into half-half distributed binary codes. For a single hash bit in M samples, with a continuous feature vector $\mathbf{u} \in \mathbb{R}^M$, we first simply sort the elements of $\mathbf{u}$ over all mini-batch images, and then assign the top half elements of sorted $\mathbf{u}$ to $+1$ and assign the remaining elements to -1, that is:

$$\mathbf{b} = \pi_0(\mathbf{u}) = \begin{cases} +1, & \text{top half of sorted } \mathbf{u} \\ -1, & \text{otherwise} \end{cases}. \tag{3.8}$$

We implement above equation as a new simple hash coding layer, dubbed ***bi-half layer*** shown in Fig.3.2, to quantize the continuous feature into half-half distributed binary code for each hash channel. The proposed bi-half layer can be easily embedded into current deep architectures to automatically generate higher quality binary codes. During training, the transmission stochasticity introduced by random small batches as shown in Eq. (3.4) can also improve the model generalization capability as the same effect of denoising.

Optimization. The discrete binary codes $\mathbf{B}$ have no continuous derivatives and cannot be directly optimized by gradient descent. Fortunately, some recent works on binarized neural networks (BNNs) have explored to use a proxy derivative approximated by straight through estimator (STE) [42] to avoid the vanishing gradients. We use the same straight-through estimator to obtain the gradients. Specifically, we expect $\mathbf{U}$ and $\mathbf{B}$ have the same update states in backward pass to match the forward goal.

Given time-step t, the current states are denoted as $\mathbf{U}_t$ and $\mathbf{B}_t$. In time-step $t+1$, we force their update states to be same that $\mathbf{U}_{t+1} = \mathbf{B}_{t+1}$. Considering the simplest SGD algorithm, we have $\mathbf{U}_{t+1} = \mathbf{U}_t - lr * \frac{\partial \mathscr{L}}{\partial \mathbf{U}_t}$ and $\mathbf{B}_{t+1} = \mathbf{B}_t - lr * \frac{\partial \mathscr{L}}{\partial \mathbf{B}_t}$ with learning rate lr and loss function $\mathscr{L}$ where $\mathscr{L}$ can be any loss function you need to use, *e.g.* reconstruction loss, cross entropy loss and so on, then the gradient of $\mathbf{U}_t$ is computed as $\frac{\partial \mathscr{L}}{\partial \mathbf{U}_t} = \frac{\partial \mathscr{L}}{\partial \mathbf{B}_t} + \gamma(\mathbf{U}_t - \mathbf{B}_t)$ with $\gamma = \frac{1}{lr}$. The forward pass and backward pass are concluded as:

$$\begin{aligned} \text{Forward:} \quad & \mathbf{B} = \pi_0(\mathbf{U}), \\ \text{Backward:} \quad & \frac{\partial \mathscr{L}}{\partial \mathbf{U}} = \frac{\partial \mathscr{L}}{\partial \mathbf{B}} + \gamma(\mathbf{U} - \mathbf{B}). \end{aligned} \tag{3.9}$$

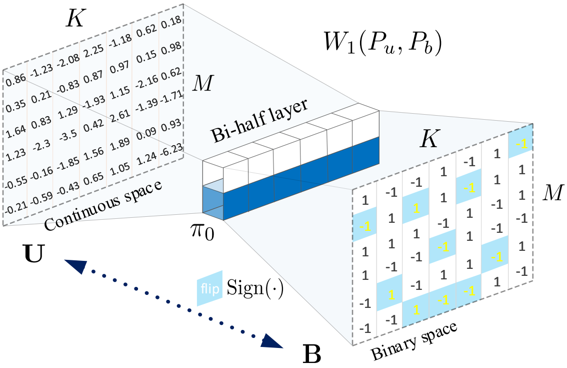


Figure 3.2: The proposed bi-half layer. M is the mini-batch size and K is the feature dimensions. A bi-half layer (middle part in white and blue) is used to quantize continuous features in **U** into binary codes in **B** via minimizing $W_1(P_u, P_b)$ in Eq.(3.7). The assignment strategy is the optimal probabilistic coupling π_0. For each bit, *i.e.*per column of **U**, we first rank its elements and then the top half elements is assigned to +1 and the remaining half elements to −1. In contrast, the commonly used sign function directly during training assigns the continuous features to their nearest binary codes which minimizes the Euclidean distance. The blue boxes indicate where our method differs from the sign function as the code in that position should flip.

In forward pass, the continuous feature is optimally quantized to half-half distributed binary codes. In backward pass, the proposed proxy derivative can automatically encourage the continuous feature distribution to align with the ideal half-half distributed distribution.

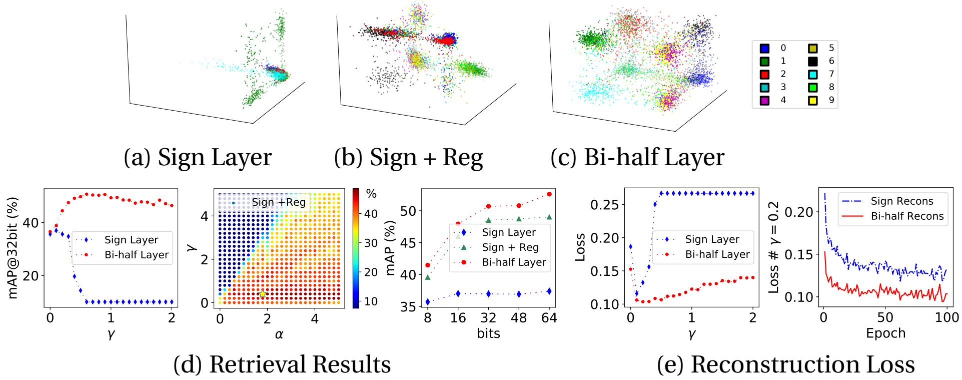
3.4. EXPERIMENTS

Datasets. • *Flickr25k* [51] contains 25k images categorized into 24 classes. Each image is annotated with at least one label. Following [33], 2,000 random images are queries and from the remaining images 5,000 random images are training set.

• *Nus-wide* [52] has around 270k images with 81 classes. To fairly compare with other methods, we consider two versions. Nus-wide(I), following [4], uses the 21 most frequent classes for evaluation. Per class, 100 random images form the query set and the remaining images form the retrieval database and training set. Nus-wide(II), following [33], uses the 10 most popular classes where 5,000 random images form the test set, and the remaining images are the retrieval set. From the retrieval set, 10,500 images are randomly selected as the training set.

• *Cifar-10* [53] consists of 60k color images categorized into 10 classes. In the literature there are also two experimental settings. In Cifar-10(I), following [38], 1k images per class (10k images in total) form the test query set, and the remaining 50k images are used for training. For Cifar-10(II), following [33], randomly selects 1,000 images per class as queries and 500 as training images, and the retrieval set has all images except for the query set.

• *Mscoco* [54] is a dataset for multiple tasks. We use the pruned set as [20] with 12,2218 images from 80 categories. We randomly select 5,000 images as queries with the rest



(a) Sign Layer (b) Sign + Reg (c) Bi-half Layer

(d) Retrieval Results (e) Reconstruction Loss

Figure 3.3: Train an AutoEncoder from scratch on Mnist dataset. The top row (a, b, c) visualizes the continuous feature distributions before binarization over different methods by training the network with 3 hash bits. (d) shows the corresponding retrieval results. We compare bi-half layer with sign layer and sign+reg. In specific, sign+reg uses an additional entropy regularization term to optimize entropy, while it is hard to balance the added term. (e) shows the reconstruction loss curves for sign layer and bi-half layer. Generating informative binary codes in latent space can help to do reconstruction.

used as database, from which 10,000 images are chosen for training.

• *Mnist* [55] contains 70k gray-scale 28 × 28 images of hand written digits from “0" to “9" across 10 classes. 1,000 images per class are randomly selected as queries and the remaining images as training set and database.

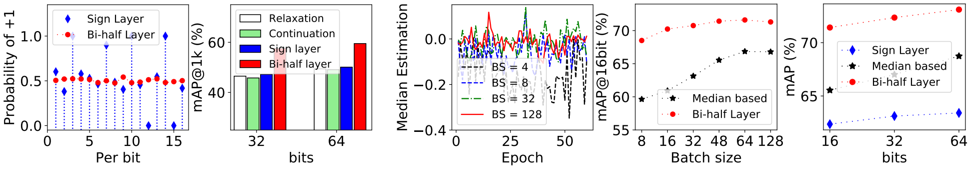
• *Ucf-101* [56] contains 13,320 action instances from 101 human action classes. All the videos are downloaded from YouTube. The average duration per video is about 7 seconds.

• *Hmdb-51* [57] includes 6,766 videos from 51 human action categories. The average duration of each video is about 3 seconds. For both Ucf-101 and Hmdb-51 datasets, we use the provided split 1, where per class 30% of the videos are used for testing and the rest 70% for training and retrieval.

Implementation details. • *Image setup.* For Mnist image dataset, we train an AutoEncoder from scratch. The details will be described in the corresponding subsection. For other image datasets, an ImageNet pre-trained VGG-16 [58] is used as our backbone where following [4, 33, 38] an additional fc layer is used for dimensionality reduction. Our bi-half layer is appended to generate the binary codes. During training, we use Stochastic Gradient Descent(SGD) as the optimizer with a momentum of 0.9 and a weight decay of 5×10^{-4} and a batch size of 32. In all experiments, the initial learning rate is set as 0.0001 and we divide the learning rate by 10 when the loss stop decreasing. The hyper-parameters γ is tuned by cross-validation on training set and set as $\gamma = 3 \times \frac{1}{N \cdot K}$.

• *Video setup.* Two 3D CNNs pre-trained on kinetics [59], ResNet-34 [60] and ResNet-101 [61], are used as backbones where we append bi-half layer to replace the last fc layer. Following the setting of [61], we use SGD as optimizer with a momentum of 0.9 and a weight decay of 0.001. The learning rate starts from 0.1, and is divided by 10 after the validation loss saturates.

Evaluation metrics. We adopt semantic similarity (by labels) as evaluation ground truth,



(a) left two subfigures on Cifar-10 (b) right three subfigures on Flickr25k

Figure 3.4: Empirical analysis on Cifar-10 and Flickr25k datasets. (a) Our bi-half layer can generate informative hash bits and outperforms other coding methods; (b) the alternative median based method performs worse than bi-half layer.

which is widely used in the unsupervised hashing literature, for instance, AGH [31], SADH [4] and DeepBit [35]. Specifically, for multi-label datasets Flickr25k, Nus-wide and Mscoco, the true neighbors are defined based on whether two images share at least one common label. We measure the performance of compared methods based on the standard evaluation metrics: Mean Average Precision (mAP), Precision-Recall curves (PR) and TopN-precision curves with top N returned samples. In our experiments, N is set to 5,000.

3.4.1. TRAINING AN AUTOENCODER FROM SCRATCH

Our bi-half layer can embedded into current deep architectures to learn binary codes from scratch. We train an AutoEncoder with a deep encoder and decoder on Mnist datasets where encoder and decoder consist of two fc layers. We append our bi-half layer after encoder to generate binary code. The reconstruction loss is used as cost function. **We compare to using the sign layer and to adding an additional entropy regularization term in the loss**. For this baseline, as in [2, 4], we use $\mathbf{B}^\top\mathbf{1}$ as regularization term balanced with the *BCE* reconstruction loss through a hyper-parameter α.

In the top row of Fig. 3.3 we train the network with 3 bits and visualize the distributions of the continuous feature $\mathbf{U}$ over 5,000 images. We observe that the features learned by sign layer are seriously tangled with each other. With binarization, most images will be scattered to same binary vertex and thus some bits have no discriminative information. By adding an entropy regularization term, the feature tanglement can be mitigated, but it is suboptimal solution which requires careful hyper-parameter tuning. The proposed bi-half layer can learn evenly distributed features.

Fig. 3.3 (d) shows the retrieval performance where the left two subfigures show the effect of hyper-parameters γ in Eq. (3.9) and α of term $\mathbf{B}^\top\mathbf{1}$. with code length 32 and the right one presents the mAP over different code lengths. Tuning the parameters can effectively improve the performance. For Sign+Reg method, it is a suboptimal solution in optimizing information entropy in comparison with bi-half layer, which can be further demonstrated in the right subfigure of Fig. 3.3 (d). The reconstruction loss for sign layer and bi-half layer is shown in Fig. 3.3 (e). We observe that generating informative binary codes in latent space can effectively minimize the reconstruction loss.

Table 3.1: mAP@1000 results on Cifar-10(I) and mAP@All results on Nus-wide(I). The ⋆ denotes that we run the experiments with the released code by the authors.

Method	Cifar-10(I)			Nus-wide(I)		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
DeepBit [35]	19.40	24.90	27.70	39.22	40.32	42.06
SAH [62]	41.80	45.60	47.40	–	–	–
SADH [4]	–	–	–	60.14	57.99	56.33
HashGAN [34]	44.70	46.30	48.10	–	–	–
GreedyHash⋆ [38]	44.80	47.20	50.10	55.49	57.47	60.93
Ours	**56.10**	**57.60**	**59.50**	**65.12**	**66.31**	**67.26**

3.4.2. EMPIRICAL ANALYSIS

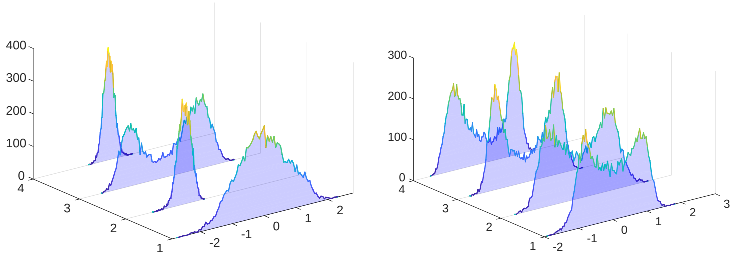
For the pre-trained models, we follow the unsupervised setting in [38] and use $||cos(\mathbf{a}_1,\mathbf{a}_2)-cos(\mathbf{b}_1,\mathbf{b}_2)||_2^2$ as cost function to minimize the difference on the cosine distance relationships, where **a** means the continuous feature extracted from the last layer of a pre-trained network of one sample while **b** means the corresponding binary code.

How are the continuous features distributed? In Fig. 3.5 we train the network on Cifar-10(I) with 4 bits and visualize the histogram distributions of each dimension in the continuous encoded feature **U** over all images. The sign layer [20, 38] does not match an ideal half-half distribution whereas our bi-half method does a better approximation.

How are individual hashing bits distributed? In the left subfigure of Fig. 3.4 (a) we show the per-bit probability of code +1 over all images for 16 bits. Cifar-10(I) dataset is used to generate hash codes. The sign layer gives a non-uniformly distribution, and even for some bits the probability is completely zero or completely one: Those bits never change their value in the entire dataset and can thus safely be discarded. In contrast, our bi-half method approximates a uniform distribution, making good use of full bit capacity.

Other hash coding strategies. The right subfigure of Fig. 3.4 (a) shows the comparison between our bi-half coding method and three other hash coding strategies: continuous relaxation layer [19, 23] ($\mathbf{B} \rightarrow \mathbf{U}$), smoothed continuation layer [20, 41] ($\mathbf{B} \rightarrow \tanh(\beta\mathbf{U})$) and sign layer [38] ($\text{sign}(\mathbf{U})$), respectively. Both 32 and 64 bits are used to generate hash codes on the Cifar-10(I) dataset. From the results, we see that the sign layer method slightly outperforms the other two coding methods which is consistent with [38]. This may be because the sign layer method can effectively keep the discrete constraint in comparison with other two methods. Our bi-half method outperforms other methods for both code sizes.

An alternative variant of bi-half layer. An alternative variant of the bi-half layer is to learn a translation term t added to the sign function $\text{sign}(\mathbf{u}+t)$ for each hash bit to get half-half distributed binary codes. We estimate the median statistic over mini-batches to implement this idea. Specifically, we keep an exponential moving average (EMA) of median points over each mini-batch which is used during inference. We conduct the comparison on Flickr25k dataset in Fig. 3.4 (b). The left subfigure of Fig. 3.4 (b) shows how the EMA estimation of median changes with the training epochs over different batch sizes. We adopt the linear learning rate scaling rule [63, 64] to adapt to batch size. We note that smaller batch size makes the estimation value unstable. The middle subfigure conducts a comparison between bi-half layer method and median translation method over different batch sizes on using 16 bits. Increasing batch size can significantly improve the



Sign layer

Bi-half layer

Figure 3.5: Comparing the distribution of continuous feature **U** for training 4 bits with a sign layer (left) versus our bi-half layer (right) over all images in Cifar-10. The y-axis shows each of the 4 bit dimensions; the x-axis shows the continuous values in **U**; the z-axis presents how many images contain such a continuous value (binned). In contrast to the sign layer, our bi-half method approximates the ideal half-half distribution.

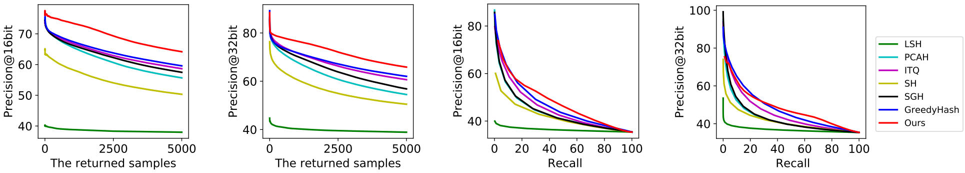


Figure 3.6: Top N precision and precision-recall curves on Mscoco. The proposed bi-half layer performs best.

Table 3.2: mAP@All for various methods on three Flickr25k, Nus-wide(II) and Cifar-10(II) datasets. Our method with 16 bits outperforms others that use 64 bits.

Method	Flickr25k			Nus-wide(II)			Cifar-10(II)		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
LSH + VGG [7]	58.31	58.85	59.33	43.24	44.11	44.33	13.19	15.80	16.73
SH + VGG [5]	59.19	59.23	60.16	44.58	45.37	49.26	16.05	15.83	15.09
ITQ + VGG [27]	61.92	63.18	63.46	52.83	53.23	53.19	19.42	20.86	21.51
DeepBit [35]	59.34	59.33	61.99	45.42	46.25	47.62	22.04	24.10	25.21
SGH [32]	61.62	62.83	62.53	49.36	48.29	48.65	17.95	18.27	18.89
SSDH [65]	66.21	67.33	67.32	62.31	62.94	63.21	25.68	25.60	25.87
DistillHash [33]	69.64	70.56	70.75	66.67	67.52	67.69	28.44	28.53	28.67
GreedyHash* [38]	62.36	63.12	63.41	51.39	55.80	59.27	28.71	31.72	35.47
Ours	**71.42**	**72.35**	**73.10**	**67.12**	**68.05**	**68.21**	**42.87**	**43.29**	**44.13**

performance for median based method. Due to memory limitations, unfortunately, it is difficult to use very large batch sizes. The left subfigure of Fig. 3.4 (b) shows the comparison with greedy hash (sign layer) and the median-based method with code length 16, 32 and 64. As expected, adding median term increases the sign layer baseline and bi-half layer significantly outperforms median-based approach.

Table 3.3: mAP@100 results on two video datasets using kinetics pre-trained 3D ResNet-34 and 3D ResNet-101. The ⋆ denotes we run the experiments with the released code.

Backbone	Method	Ucf-101			Hmdb-51		
		16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
ResNet-34	GreedyHash⋆	45.49	57.24	64.77	30.32	37.55	40.53
	Ours	50.83	60.30	65.89	34.21	38.67	41.74
ResNet-101	GreedyHash⋆	39.29	58.35	67.23	27.60	39.96	42.07
	Ours	**59.30**	**66.13**	**68.47**	**36.68**	**41.48**	**43.03**

3.4.3. COMPARISON WITH STATE-OF-THE ART

We compare our method with previous unsupervised hashing methods, including seven shallow unsupervised hashing methods, *i.e.* LSH [7], SH [5], PCAH, ITQ [27], SGH [29], and eight deep unsupervised hashing methods, *i.e.* DeepBit [35], SGH [32], SSDH [65], DistillHash [33], SAH [62], HashGAN [34], SADH [4], and GreedyHash [38]. To have a fair comparison, we adopt the deep features for all shallow architecture-based baseline methods. For GreedyHash, we run the experiments with the released code by the authors. For other methods, the results are taken from the related literatures.

Table 3.1 shows the mAP@1000 results on Cifar-10(I) and mAP@All results on Nus-wide(I) over three different hash code sizes 16, 32 and 64. The compared greedy hash [38] method which directly uses the sign function as hash coding layer outperforms everything except our method for all code sizes. Greedy hash [38] is effective to solve the vanishing gradient problem and maintain the discrete constraint in hash learning, but it cannot maximize hash bit capacity. In contrast, our method does maximize hash bit capacity and clearly outperforms all other methods on this two datasets.

In Table 3.2, we present the mAP results on three datasets Flickr25k, Nuswide(II), and Cifar-10(II), with hash code length varying from 16 to 64. The experiments are conducted with the same setting as in the compared methods. We do best for all hash bits sizes for all three datasets.

In Fig. 3.6, we conduct experiments on more challenging Mscoco dataset. The left two subfigures present the TopN-precision curves with code lengths 16 and 32. Consistent with mAP results, we can observe that our method performs best. Both mAP and TopN-precision curves are Hamming ranking based metrics where our method can achieve superior performance. Moreover, we plot the precision-recall curves for all methods with hash bit lengths of 16 and 32 in the right two subfigures Fig. 3.6 to illustrate the hash lookup results. From the results, we can again observe that our method consistently achieves the best results among all approaches, which further demonstrates the superiority of our proposed method.

Hashing is about compact storage and fast retrieval, thus we analyze using fewer bits in Table 3.1, Table 3.2 and Fig. 3.6. Only for Nus-wide(II) we perform on par while in all other datasets our method using 16 bits clearly outperforms other methods using 64 bits. This shows a 3 times reduction in storage and speed while even improving accuracy.

Video Retrieval Results: In Table 3.3, we present the mAP@100 results for Ucf-101 and Hmdb-51 datasets with code length 16, 32 and 64. For both datasets and both ResNet models our bi-half method consistently outperforms the sign layer method [38] over all hash bit length, especially for short bits. In hashing, fewer bits is essential to save storage and compute.

3.5. CONCLUSION

We propose a new parameter-free Bi-half Net for unsupervised hashing learning by optimizing bit entropy. Our Bi-half layer has no hyper-parameters and compares favorably to minimizing bit entropy with an additional hyper-parameter in the loss. The designed bi-half layer can be easily embedded into current deep architectures, such as AutoEncoders, to automatically generate higher quality binary codes. The proposed proxy derivative in backward pass can effectively encourage the continuous feature distribution to align with the ideal half-half distributed distribution. One limitation is that the independence between different bits is not considered, which will be investigated in future work. Experiments on 7 datasets show state of the art results. We often outperform other hashing methods that use 64 bits where we need only 16 bits.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 770–778.

[2] V. Erin Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, *Deep hashing for compact binary codes learning,* in *CVPR* (2015) pp. 2475–2483.

[3] W. Liu, S. Kumar, S. Kumar, and S. F. Chang, *Discrete graph hashing,* in *NIPS* (2014).

[4] F. Shen, Y. Xu, L. Liu, Y. Yang, Z. Huang, and H. T. Shen, *Unsupervised deep hashing with similarity-adaptive and discrete optimization,* IEEE TPAMI **40**, 3034 (2018).

[5] Y. Weiss, A. Torralba, and R. Fergus, *Spectral hashing,* in *NIPS* (2008).

[6] B. Xu, J. Bu, Y. Lin, C. Chen, X. He, and D. Cai, *Harmonious hashing,* in *IJCAI* (2013).

[7] M. Datar and P. Indyk, *Locality-sensitive hashing scheme based on p-stable distributions,* in *Proceedings of the ACM Symposium on Computational Geometry* (ACM Press, 2004) pp. 253–262.

[8] A. Gionis, P. Indyk, and R. Motwani, *Similarity search in high dimensions via hashing,* in *Proceedings of International Conference on Very Large Databases* (2000) pp. 518–529.

[9] B. Kulis and K. Grauman, *Kernelized locality-sensitive hashing for scalable image search,* in *ICCV* (2009).

[10] B. Kulis, P. Jain, and K. Grauman, *Fast similarity search for learned metrics,* IEEE Transactions on Pattern Analysis Machine Intelligence **31**, 2143 (2009).

[11] Y. Mu and S. Yan, *Non-metric locality-sensitive hashing,* in *AAAI* (2010).

[12] M. Raginsky, *Locality-sensitive binary codes from shift-invariant kernels,* (2009).

[13] S. F. Chang, *Supervised hashing with kernels,* in *CVPR* (2012).

[14] H. Lai, Y. Pan, Y. Liu, and S. Yan, *Simultaneous feature learning and hash coding with deep neural networks.* in *CVPR* (2015).

[15] G. Lin, C. Shen, Q. Shi, A. V. D. Hengel, and D. Suter, *Fast supervised hashing with decision trees for high-dimensional data,* in *CVPR* (2014).

[16] R. Raziperchikolaei and M. A. Carreira-Perpiñán, *Optimizing affinity-based binary hashing using auxiliary coordinates,* in *NIPS* (2016).

[17] F. Shen, C. Shen, W. Liu, and H. T. Shen, *Supervised discrete hashing,* in *CVPR* (2015).

[18] P. Zhang, W. Zhang, W. J. Li, and M. Guo, *Supervised hashing with latent factor models,* in *SIGIR* (2014).

[19] Y. Cao, M. Long, L. Bin, and J. Wang, *Deep cauchy hashing for hamming space retrieval,* in *CVPR* (2018).

[20] Z. Cao, M. Long, J. Wang, and P. S. Yu, *Hashnet: Deep learning to hash by continuation, ICCV,* (2017).

[21] Q. Li, Z. Sun, R. He, and T. Tan, *Deep supervised discrete hashing,* in *NIPS* (2017).

[22] Y. Li, W. Pei, J. van Gemert, *et al.*, *Push for quantization: Deep fisher hashing,* BMVC (2019).

[23] H. Liu, R. Wang, S. Shan, and X. Chen, *Deep supervised hashing for fast image retrieval,* CVPR (2016).

[24] H. Liu, R. Wang, S. Shan, and X. Chen, *Learning multifunctional binary codes for both category and attribute oriented retrieval tasks,* in *CVPR* (2017).

[25] X. Yuan, L. Ren, J. Lu, and J. Zhou, *Relaxation-free deep hashing via policy gradient,* in *ECCV* (2018) pp. 134–150.

[26] Z. Chen, X. Yuan, J. Lu, Q. Tian, and J. Zhou, *Deep hashing via discrepancy minimization,* in *CVPR* (2018) pp. 6838–6847.

[27] Y. Gong and S. Lazebnik, *Iterative quantization: A procrustean approach to learning binary codes,* in *CVPR* (2011).

[28] K. He, F. Wen, and J. Sun, *K-means hashing: An affinity-preserving quantization method for learning binary compact codes,* in *CVPR* (2013).

[29] Q. Y. Jiang and W. J. Li, *Scalable graph hashing with feature transformation,* in *International Conference on Artificial Intelligence* (2015).

[30] W. Kong and W. J. Li, *Isotropic hashing,* in *NIPS* (2012).

[31] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, *Hashing with graphs,* in *ICML* (2011).

[32] B. Dai, R. Guo, S. Kumar, N. He, and L. Song, *Stochastic generative hashing,* in *ICML* (2017) pp. 913–922.

[33] E. Yang, T. Liu, C. Deng, W. Liu, and D. Tao, *Distillhash: Unsupervised deep hashing by distilling data pairs,* in *CVPR* (2019) pp. 2946–2955.

[34] K. Ghasedi Dizaji, F. Zheng, N. Sadoughi, Y. Yang, C. Deng, and H. Huang, *Unsupervised deep generative adversarial hashing network,* in *CVPR* (2018) pp. 3664–3673.

[35] K. Lin, J. Lu, C.-S. Chen, and J. Zhou, *Learning compact binary descriptors with unsupervised deep neural networks,* in *CVPR* (2016) pp. 1183–1192.

[36] J. Chen, W. K. Cheung, and A. Wang, *Learning deep unsupervised binary codes for image retrieval,* in *IJCAI* (2018).

[37] J. Lu, V. E. Liong, and J. Zhou, *Deep hashing for scalable image search,* IEEE TIP **26**, 2352 (2017).

[38] S. Su, C. Zhang, K. Han, and Y. Tian, *Greedy hash: Towards fast optimization for accurate hash coding in cnn,* in *NIPS* (2018) pp. 798–807.

[39] Q.-Y. Jiang and W.-J. Li, *Deep cross-modal hashing,* in *CVPR* (2017).

[40] F. Zhao, Y. Huang, L. Wang, and T. Tan, *Deep semantic ranking based hashing for multi-label image retrieval,* in *CVPR* (2015).

[41] H. Lai, Y. Pan, Y. Liu, and S. Yan, *Simultaneous feature learning and hash coding with deep neural networks,* in *CVPR* (2015).

[42] Y. Bengio, N. Léonard, and A. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation,* Technical Report (2013).

[43] M. I. Belghazi, A. Baratin, S. Rajeswar, S. Ozair, Y. Bengio, A. Courville, and R. D. Hjelm, *Mine: mutual information neural estimation,* ICML (2018).

[44] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, *Infogan: Interpretable representation learning by information maximizing generative adversarial nets,* in *NIPS* (2016) pp. 2172–2180.

[45] I. T. Jolliffe, *Principal component analysis,* Journal of Marketing Research **87**, 513 (2002).

[46] H. Jegou, M. Douze, and C. Schmid, *Hamming embedding and weak geometric consistency for large scale image search,* in *ECCV* (Springer, 2008) pp. 304–317.

[47] D. Zhang, J. Wang, D. Cai, and J. Lu, *Self-taught hashing for fast similarity search,* in *ACM SIGIR* (2010) pp. 18–25.

[48] T. M. Cover and J. A. Thomas, *Elements of information theory* (John Wiley & Sons, 2012).

[49] C. E. Shannon, *A mathematical theory of communication,* Bell system technical journal **27**, 379 (1948).

[50] C. Villani, *Topics in optimal transportation,* 58 (American Mathematical Soc, 2003).

[51] M. J. Huiskes and M. S. Lew, *The mir flickr retrieval evaluation,* in *ACM Multimedia* (2008) pp. 39–43.

[52] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, *Nus-wide: a real-world web image database from national university of singapore,* in *CIVR* (2009).

[53] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images,* Tech. Rep. (Citeseer, 2009).

[54] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, *Microsoft coco: Common objects in context,* in *European conference on computer vision* (Springer, 2014) pp. 740–755.

[55] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, *Gradient-based learning applied to document recognition,* Proceedings of the IEEE **86**, 2278 (1998).

[56] K. Soomro, A. R. Zamir, and M. Shah, *Ucf101: A dataset of 101 human actions classes from videos in the wild,* CORR (2012).

[57] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, *Hmdb: a large video database for human motion recognition,* in *ICCV* (2011) pp. 2556–2563.

[58] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition,* ICLR (2015).

[59] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, *et al.*, *The kinetics human action video dataset,* CORR (2017).

[60] K. Hara, H. Kataoka, and Y. Satoh, *Learning spatio-temporal features with 3d residual networks for action recognition,* in *ICCV* (2017) pp. 3154–3160.

[61] K. Hara, H. Kataoka, and Y. Satoh, *Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?* in *CVPR* (2018) pp. 6546–6555.

[62] T.-T. Do, D.-K. Le Tan, T. T. Pham, and N.-M. Cheung, *Simultaneous feature aggregating and hashing for large-scale image search,* in *CVPR* (2017) pp. 6618–6627.

[63] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, *Accurate, large minibatch sgd: Training imagenet in 1 hour,* arXiv preprint arXiv:1706.02677 (2017).

[64] A. Krizhevsky, *One weird trick for parallelizing convolutional neural networks,* arXiv preprint arXiv:1404.5997 (2014).

[65] E. Yang, C. Deng, T. Liu, W. Liu, and D. Tao, *Semantic structure-based unsupervised deep hashing.* in *IJCAI* (2018) pp. 1064–1070.

4

Equal Bits: Enforcing Equally Distributed Binary Network Weights

Binary networks are extremely efficient as they use only two symbols to define the network: $\{+1, -1\}$. One can make the prior distribution of these symbols a design choice. The recent IR-Net of Qin *et al.* argues that imposing a Bernoulli distribution with equal priors (equal bit ratios) over the binary weights leads to maximum entropy and thus minimizes information loss. However, prior work cannot precisely control the binary weight distribution during training, and therefore cannot guarantee maximum entropy. Here, we show that quantizing using optimal transport can guarantee any bit ratio, including equal ratios. We investigate experimentally that equal bit ratios are indeed preferable and show that our method leads to optimization benefits. We show that our quantization method is effective when compared to state-of-the-art binarization methods, even when using binary weight pruning.

4.1. Introduction

Binary networks allow compact storage and swift computations by limiting the network weights to only two bit symbols $\{-1,+1\}$. In this paper we investigate weights priors before seeing any data: is there a reason to prefer predominantly positive bit weights? Or more negative ones? Or is equality preferable? Successful recent work [1] argues that a good prior choice is to have an equal bit-ratio: i.e. an equal number of +1 and −1 symbols in the network. This is done by imposing an equal prior under the standard Bernoulli distribution [1–3]. Equal bit distributions minimize information loss and thus maximizes entropy, showing benefits across architectures and datasets [1]. However, current work cannot add a hard constraint of making symbol priors exactly equal, and therefore cannot guarantee maximum entropy.

Here, we propose a method to add a hard constraint to binary weight distribution, offering precise control for any desired bit ratio, including equal prior ratios. We add hard constraints in the standard quantization setting [1, 4, 5] making use of real-valued latent weights that approximate the binary weights. We quantize these real-valued weights by aligning them to any desired prior Bernoulli distribution, which incorporates our preferred binary weight prior. Our quantization uses optimal transport [6] and can guarantee any bit ratio. Our method makes it possible to experimentally test the hypothesis in [1] that equal bit ratios are indeed preferable to other bit ratios. We baptize our approach with equal bit ratios: *bi-half*. Furthermore, we show that enforcing equal priors using our approach leads to optimization benefits by reducing the problem search-space and avoiding local minima.

We make the following contributions: (i) a binary network optimization method based on optimal transport; (ii) exact control over weight bit ratios; (iii) validation of the assumption that equal bit ratios are preferable; (iv) optimization benefits such as search-space reduction and good minima; (v) favorable results compared to the state-of-the-art, and can ensure half-half weight distribution even when pruning is used.

4.2. Related Work

For a comprehensive survey on binary networks, see [7]. In Table 4.1 we show the relation between our proposed method and pioneering methods, that are representatives of their peers, in terms of the binarization choices made. The XNOR method (Table 4.1(a)) was the first to propose binarizing latent real-valued weights using the sign function [5]. Rather than making each binary weight depend only on its associated real-value weight or gradient value, IR-Net [1] (Table 4.1(b)) is a prototype method that uses filter-weight statistics to update each individual binary weight. Here, we also use filter-weight statistics to update the binary weights, however similar to [8] Table 4.1(d)) we do not rely on the sign function for binarization, but instead use binary weight flips. This is a natural choice, as flipping the sign of a binary weight is the only operation one can apply to binary weights.

Sign versus bit flips. The front-runners of binary networks are BinaryConnect [9] and XNOR [5] and rely on auxiliary real weights and the sign function to define binary weights. These works are subsequently extended with focus on the scaling factors in XNOR++ [4]

	Initialization	Binarization, b
ⓐ Sign**, no filter statistics** XNOR-Net [5]	Gradient g; Latent weight w.	$b \leftarrow \text{sign}(w)$
ⓑ Sign**, filter statistics** IR-Net [1]	Gradient g; Latent weight w.	$b \leftarrow \text{sign}\left(\frac{w-\text{avg}(w)}{\text{std}(w-\text{avg}(w))}\right)$
ⓒ Flip, filter statistics Ours	Gradient g, Latent weight w; Threshold dependent on w.	$b \leftarrow \text{flip}(b),\ \text{if} \begin{cases} \text{rank}(w) < \frac{D}{2}, \text{ and rank}(w-\alpha g) \geq \frac{D}{2} \\ \text{rank}(w) \geq \frac{D}{2}, \text{ and rank}(w-\alpha g) < \frac{D}{2} \end{cases}$
ⓓ Flip, no filter statistics Bop [8]	Gradient g; Predefined threshold τ.	$b \leftarrow \text{flip}(b), \text{if} \begin{cases} \tau < \lvert g \rvert, \text{ and} \\ \text{sign}(g) = \text{sign}(b) \end{cases}$

Table 4.1: **Optimization perspectives.** (a) Classical binarization methods tie each binary weight b to an associated real-valued latent variable w, and quantize each weight by only considering its associated real-valued by using the sign function. (b) Rather than updating the weights independent of each other, recent work uses filter-weight statistics when updating the binary weights. (c) Our proposed optimization method does not focus on using the sign function, but rather flips the binary weights based on the distribution of the real weights, thus the binary weight updates depend on the statistics of the other weights through the rank of w. (d) Recent work moves away from using the sign of the latent variables, and instead trains the binary network with bit sign flips, however they still consider independent weight updates.

and BNN+ [10], while HWGQ [11] uses the sign function recursively for binarization. Bi-Real [12] also uses the sign function for binarization and analyzes better approximations of the gradient of the sign function. From a different perspective, recent work tries to sidestep having to approximate the gradient of the sign function, and uses bit flips to train binary networks [8]. The sign of the binary weights can be flipped based on searchable [13] or learnable thresholds [14]. Here, we also rely on bit flips based on a dynamic thresholding of the real weights, entailed by our optimal transport optimization strategy.

Using filter statistics or not. Commonly, binarization methods define each binary weight update by considering only its associated value in the real-valued latent weights [4, 5, 11, 12] or in the gradient vector [8]. However, binary weights can also be updated using explicit statistics of the other weights in the filter [15] or implicitly learned through a function [16]. The real-valued filter statistics are used in IR-Net [1] to enforce a Bernoulli distribution with equal priors. Similarly, our optimal transport optimization leads to ranking the real weights, and therefore making use of the statistics of the real-weights in each filter.

Network pruning. Pruning has been shown to improve the efficiency of deep neural networks [17–21]. However, the reason why pruning can bring improvements remains unclear in real-valued networks. It is commonly believed [22–25] that finding the “important" weight values is crucial for retraining a small pruned model. Specifically, the “important" weight values are inherited [26] or re-winded [23] from a large trained model. In contrast, [27] claims that the selected important weights are typically not useful for the small pruned model, while the pruned architecture itself is more relevant. The lottery-ticket idea has recently been applied to binary networks [28]. Here, we show that having equal +1 and −1 ratios is also optimal when the networks rely on pruning and that our optimal transport optimization can easily be adapted to work with other methods using network pruning.

4.3. BINARIZING WITH OPTIMAL TRANSPORT

4.3.1. BINARY WEIGHTS

We define a binary network where the weights **B** take binary values $\{1,-1\}^D$. The binary weights **B** follow a Bernoulli distribution $\mathbf{B} \sim \text{Be}(p_{pos})$, describing the probabilities of individual binary values $b \in \{-1,1\}$ in terms of the hyperparameters p_{pos} and p_{neg}:

$$p(b) = \text{Be}(b \mid p_{pos}) = \begin{cases} p_{pos} & \text{if } b = +1 \\ p_{neg} = 1 - p_{pos}, & \text{if } b = -1 \end{cases} \tag{4.1}$$

To be consistent with previous work, we follow XNOR-Net [5] and apply the binary optimization per individual filter.

Because the matrix **B** is discrete, we follow [5, 9] by using real-valued latent weights **W** to aid the training of discrete values, where each binary weight in **B** has an associated real-valued weight in **W**. In the forward pass we quantize the real-valued weights **W** to estimate the matrix **B**. Then, we use the estimated matrix **B** to compute the loss, and in the backward pass we update the associated real-valued weights **W**.

4.3.2. OPTIMAL TRANSPORT OPTIMIZATION

The optimization aligns the real-valued weight distribution **W** with the prior Bernoulli distribution in Eq. (4.1) and quantizes the real-valued weights **W** to **B**.

The empirical distribution $\mathbb{P}_w$ of the real-valued variable $\mathbf{W} \in \mathbb{R}^D$ and the empirical distribution $\mathbb{P}_b$ for the discrete variable **B** can be written as:

$$\mathbb{P}_w = \sum_{i=1}^{D} p_i \delta_{w_i}, \quad \mathbb{P}_b = \sum_{j=1}^{2} q_j \delta_{b_j}, \tag{4.2}$$

where $\delta_{\mathbf{x}}$ is the Dirac function at location **x**. The p_i and q_j are the probability mass associated to the corresponding distribution locations w_i and b_j, where $\mathbb{P}_b$ has only 2 possible locations in the distribution space $\{-1,1\}$.

To align $\mathbb{P}_w$ with the Bernoulli prior $\mathbb{P}_b$ in Eq. (4.1) we use optimal transport (OT) [6] which minimizes the cost of moving the starting distribution $\mathbb{P}_w$ to the target distribution $\mathbb{P}_b$. Because $\mathbb{P}_w$ and $\mathbb{P}_b$ are only accessible through a finite set of values, the corresponding optimal transport cost is:

$$\pi_0 = \min_{\pi \in \Pi(\mathbb{P}_w, \mathbb{P}_b)} \langle \pi, \mathscr{C} \rangle_F, \tag{4.3}$$

where $\Pi(\mathbb{P}_w, \mathbb{P}_b)$ is the space of the joint probability with marginals $\mathbb{P}_w$ and $\mathbb{P}_b$, and π is the general probabilistic coupling that indicates how much mass is transported to push distribution $\mathbb{P}_w$ towards the distribution $\mathbb{P}_b$. The $\langle .,. \rangle_F$ denotes the Frobenius dot product, and $\mathscr{C} \geq 0$ is the cost function matrix whose element $\mathscr{C}(w_i, b_j)$ denotes the cost of moving a probability mass from location w_i to location b_j in distribution space. When the cost is defined as a distance, the OT becomes the Wasserstein distance. We minimize the 1-Wasserstein distance between $\mathbb{P}_w$ and $\mathbb{P}_b$. This minimization has an elegant closed-form solution based on simply sorting. For a continuous-valued weights vector

$\mathbf{W} \in \mathbb{R}^D$, we first sort the elements of $\mathbf{W}$, and then assign the top $p_{pos}D$ elements to $+1$, and the bottom $(1-p_{pos})D$ portion of the elements to -1:

$$\mathbf{B} = \pi_0\left(\mathbf{W}\right) = \begin{cases} +1, & \text{top } \mathrm{p_{pos}D} \text{ of sorted } \mathbf{W} \\ -1, & \text{bottom } (1-\mathrm{p_{pos}})\mathrm{D} \text{ of sorted } \mathbf{W} \end{cases} \tag{4.4}$$

Rather than using the sign function to define the binarization, we flip the binary weights based on the distribution of $\mathbf{W}$. Thus the flipping of a binary weight depends on the distribution of the other binary weights through $\mathbf{W}$, which is optimized to be as close as possible to $\mathbf{B}$.

When applying our method in combination with pruning as in [28], we first mask the binary weights $\mathbf{B}' = \mathbf{M} \odot \mathbf{B}$ with a mask $\mathbf{M} \in \{0,1\}^D$. This leads to a certain percentage of the weights being pruned. Subsequently, we apply the Eq. (4.4) to the remaining non-pruned weights, where D in Eq. (4.4) become the L_1 norm of the mask, $|\mathbf{M}|$.

4.3.3. BI-HALF: EXPLICITLY CONTROLLING THE BIT RATIO

Our optimal transport optimization allows us to enforce a hard constraint on precise bit ratios by varying the p_{pos} value. Therefore, we can test a range of prior binary weight distributions.

Following [1], a good prior over the binary weights is one maximizing the entropy. Using optimal transport, we maximize the entropy of the binary weights by setting the bit ratio to half in Eq. (4.4):

$$p^*_{pos} = \operatorname{argmax}_{p_{pos}} H(\mathbf{B} \sim \mathrm{Be}(p_{pos})) = \frac{1}{2}, \tag{4.5}$$

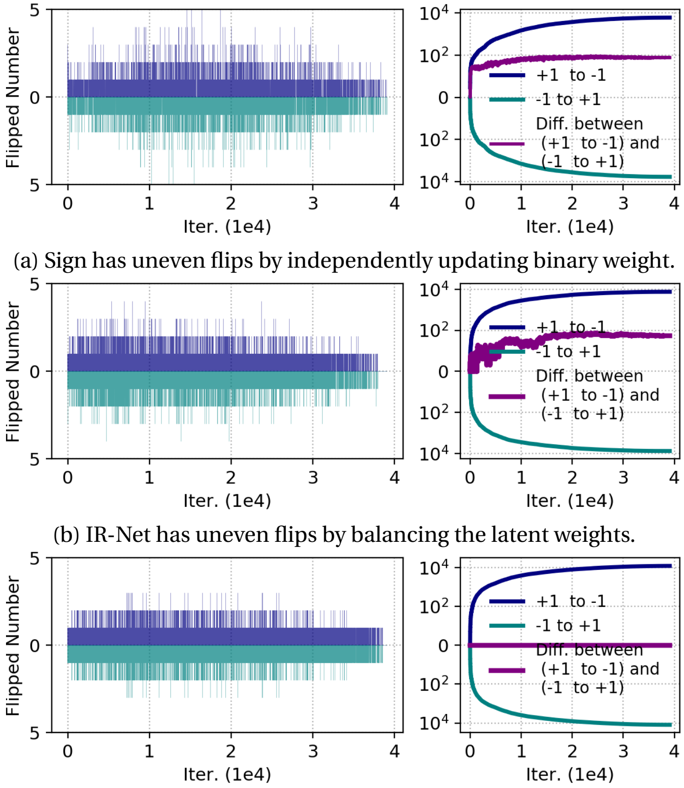
where $H(\cdot)$ denotes the entropy of the binary weights $\mathbf{B}$. We dub this approach *bi-half*. Unlike previous work [1], we can guarantee equal symbol distributions and therefore maximum entropy throughout the complete training procedure.

Initialization and scaling factor. We initialize the real-valued weights using Kaiming normal [29]. The binary weights are initialized to be equally distributed per filter according to Eq. (4.5). To circumvent exploding gradients, we use one scaling factor α per layer for the binary weights to keep the activation variance in the forward pass close to 1. Based on the ReLU variance analysis in [29] it holds that $\frac{1}{2}D \cdot Var(\alpha\mathbf{B}) = 1$, where D is the number of connections and $\mathbf{B}$ are our binary weights. $\mathbf{B}$ is regularized to a *bi-half* distribution, thus $Var(\mathbf{B}) = 1$, which gives $\alpha = \sqrt{2/D}$.

To better clarify, for an L-layer network with input data y_1 standardized to $Var[y_1] = 1$, where the variance of each binary layer l is $Var[\mathbf{B}_l] = 1$, and D_l is the number of connections in that layer: *i)* Without the scaling, the output variance is $Var[y_L] = Var[y_1] \prod_{l=2}^{L} \frac{D_l}{2} Var[\mathbf{B}_l] = \prod_{l=2}^{L} \frac{D_l}{2}$. Typically D_l is large, leading to exploding gradients; *ii)* With the scaling, we scale $\mathbf{B}_l$ by $\alpha = \sqrt{2/D_l}$, leading to $Var[y_L] = Var[y_1] \prod_{l=2}^{L} \frac{D_l}{2} Var(\alpha\mathbf{B}_l) = 1$ which stabilizes learning.

4.4. EXPERIMENTS

Datasets and implementation details. We evaluate on Cifar-10, Cifar-100 [30] and ImageNet [31], for a number of network architectures. Following [23, 32] we evaluate 4

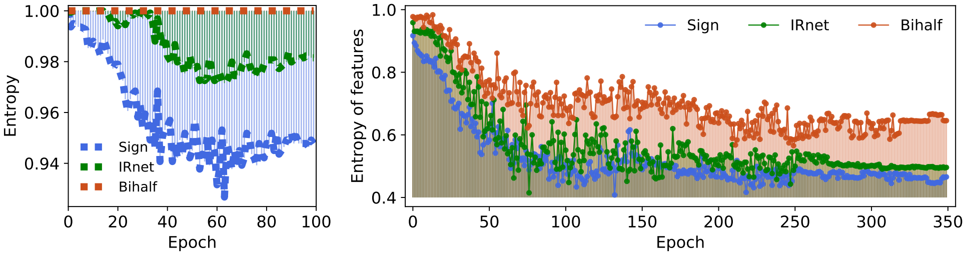


(a) Sign has uneven flips by independently updating binary weight.

(b) IR-Net has uneven flips by balancing the latent weights.

(c) *Bi-half* has even flips by precisely controlling the flipping.

Figure 4.1: **Hypothesis: *bi-half* maximizes the entropy.** We compare the bit flips during training in our *bi-half* with the sign [5] and IR-Net [1] on the Conv2 network on Cifar-10. The x-axis shows the training iterations. *Left:* Bit flips during training to +1 (dark blue) or to -1 (cyan). *Right:* Accumulated bit flips over the training iterations, as well as the difference between the bit flips from $(+1 \text{ to } -1)$ and the ones from $(-1 \text{ to } +1)$. In contrast to sign and IR-Net, our *bi-half* method can guarantee an equal bit ratio.



(a) Binary weight entropy.
On Conv2

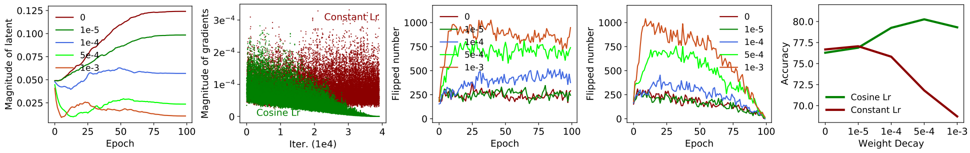
(b) Activation entropy.
On ResNet-18

Figure 4.2: **Hypothesis: bi-half maximizes the entropy.** We compare our bi-half method to sign [5] and IR-Net [1]. (a) Entropy of the binary weights during training for Conv2 on Cifar10. (b) Entropy of the network activations for ResNet-18 on Cifar100. Our *bi-half* model can guarantee maximum entropy during training for the binary weight distribution and it is able to better maximize the entropy of the activations.

shallow CNNs: Conv2, Conv4, Conv6, and Conv8 with 2/4/6/8 convolutional layers. We train the shallow models on Cifar-10 for 100 epochs, with weight decay $1e^{-4}$, momentum 0.9, batch size 128, and initial learning rate 0.1 using a cosine learning rate decay [33]. Following [1] we also evaluate their ResNet-20 architecture and settings on Cifar-10. On Cifar-100, we evaluate our method on 5 different models including VGG16 [34], ResNet18 [35], ResNet34 [35], InceptionV3 [36], ShuffleNet [37]. We train the Cifar-100 models for 350 epochs using SGD with weight decay $5e^{-4}$, momentum 0.9, batch size 128, and initial learning rate 0.1 divided by 10 at epochs 150, 250 and 320. For ImageNet we use ResNet-18 and ResNet-34 trained for 100 epochs using SGD with momentum 0.9, weight decay $1e^{-4}$, and batch size 256. Following [1, 12], the initial learning rate is set as 0.1 and we divide it by 10 at epochs 30, 60, 90. All our models are trained from scratch without any pre-training. For the shallow networks we apply our method on all layers, while for the rest we follow [1, 12], and apply it on all convolutional and fully-connected layers except the first, last and the downsampling layers.

4.4.1. HYPOTHESIS: BI-HALF MAXIMIZES THE ENTROPY

Here we test whether our proposed *bi-half* model can indeed guarantee maximum entropy and therefore an exactly equal ratio of the -1 and $+1$ symbols. Fig. 4.1 shows the bit flips performed in our proposed *bi-half* method during training when compared to two baselines: sign [5] and IR-Net [1]. We train a Conv2 network on Cifar-10 and plot the flips of binary weights in a single binary weight filter during training. The binary weights are initialized to be equal distributed (half of the weights positive and the other half negative). The classical sign method [5] in Fig. 4.1(c) binarizes each weight independent of the other weights, therefore during training the flips for ($+1$ to -1) and (-1 to $+1$) are uneven. The recent IR-Net [1] in Fig. 4.1(b) balances the latent weights by using their statistics to obtain evenly distributed binary weight values. However, it can not guarantee evenly distributed binary weights throughout training. Our *bi-half* model in Fig. 4.1(c) updates the binary weight based on the statistics of the other weights. For our



(a) Latent weights (b) Gradient (c) Constant lr (d) Cosine lr (e) Accuracy

Figure 4.3: **Empirical analysis (a): Effect of hyper-parameters.** We show the effect of weight decay and learning rate decay on binary weights flips using the Conv2 network on Cifar-10. Carefully tuning these hyper-parameters is important for adequately training the binary networks.

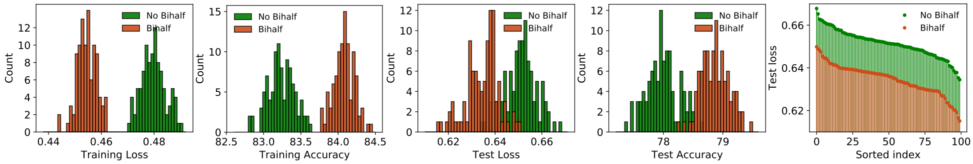


Figure 4.4: **Empirical analysis (c): Optimization benefits.** We train our *bi-half* model 100 times on Cifar-10 and plot the distribution of the losses and accuracies over the 100 repetitions. We compare our results using optimal transport to the results using the standard sign function. On average our *bi-half* model tends to arrive at better losses and accuracies than the baseline.

method the binary weights are evenly flipped during training, offering exact control of bit weight ratios.

Fig. 4.2(a) shows the binary weights entropy changes during training on Conv2 when compared to sign [5] and IR-Net [1]. IR-Net aims to maximize entropy by subtracting the mean value of the weights, yet, this is not exact. In contrast, we maximize the information entropy by precisely controlling the binary weight distribution. In Fig. 4.2(b) we show the entropy of the binary activations. Adjusting the distribution of binary weights retains the information in the binary activation. For our *bi-half* method, the binary activation of each channel is close to the maximum information entropy under the Bernoulli distribution.

4.4.2. EMPIRICAL ANALYSIS

(a) Effect of hyper-parameters. In Fig. 4.3 we study the effectiveness of the commonly used training techniques of varying the weight decay and learning rate decay, when training the Conv2 network on Cifar-10. Fig. 4.3(a) shows that using a higher weight decay reduces the magnitude of latent weights during training and therefore the magnitude of the cut-off point (threshold) between the positive and negative values. Fig. 4.3(b) compares the gradient magnitude of two different learning rate (lr) schedules: “constant lr" and “cosine lr". The magnitude of the gradients reduces during training when using the cosine learning rate. In Fig. 4.3(c) we find that increasing the weight decay for binary network with a constant learning rate schedule, increases binary weights flips. Fig. 4.3(d) shows that decaying the learning rate when using a cosine learning rate schedule gradually decreases the number of flipped weights. Fig. 4.3(e) shows that the choice of weight decay and learning rate decay affect each other. Our bi-half method uses the rank of la-

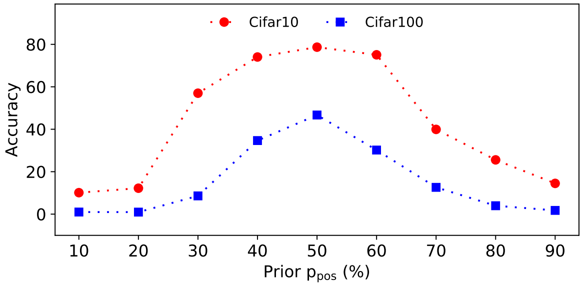


Figure 4.5: **Empirical analysis (b): Which bit-ratios are preferred?** We test on Cifar-10 and Cifar-100 using Conv2 the choice of the priorp_{pos} under the Bernoulli distribution. The x-axis is the probability of the +1 connections denoted by p_{pos} in the Bernoulli prior distribution, while the y-axis denotes the top-1 accuracy values. Results are in agreement with the hypothesis of Qin *et al.* [1] that equal priors as imposed in our *bi-half* model are preferable.

tent weights to flip the binary weights. A proper tuned hyper-parameter of weight decay and learning rate decay will affect the flipping threshold. Therefore in the experiments, we carefully tune the hyper-parameters of weight decay and learning rate decay to build a competitive baseline.

(b) Which bit-ratio is preferred? In Fig. 4.5, we evaluate the choice of the prior p_{pos} in the Bernoulli distribution for Conv2 on Cifar-10 and Cifar-100. By varying the bit-ratio, the best performance is consistently obtained when the negative and positive symbols have equal priors as in the *bi-half* model. Indeed, as suggested in [1], when there is no other a-priori reason to select a different p_{pos}, having equal bit ratios is a good choice.

(c) Optimization benefits with bi-half. The uniform prior over the -1 and $+1$ under the Bernoulli distribution regularizes the problem space, leading to only a subset of possible weight combinations available during optimization. We illustrate this intuitively on a $2D$ example for a simple fully-connected neural network with one input layer, one hidden layer, and one output layer in a two-class classification setting. We consider a $2D$ binary input vector $\mathbf{x} = [x_1, x_2]^\top$, and define the network as: $\sigma(\mathbf{w_2}^\top \sigma(\mathbf{w_1}^\top \mathbf{x} + \mathbf{b_1}))$, where $\sigma(\cdot)$ is a sigmoid nonlinearity, $\mathbf{w_1}$ is a $[2 \times 3]$ binary weight matrix, $\mathbf{b_1}$ is $[3 \times 1]$ binary bias vector, and $\mathbf{w_2}$ is a $[3 \times 1]$ binary vector. We group all 12 parameters as a vector $\mathbf{B}$. We enumerate all possible binary weight combinations in $\mathbf{B}$, *i.e.* $2^{12} = 4096$, and plot all decision boundaries that separate the input space into two classes as shown in Fig. 4.7(a). All possible 4096 binary weights combinations offer only 76 unique decision boundaries. In Fig. 4.7.(b) the Bernoulli distribution over the weights with equal prior (*bi-half*) regularizes the problem space: it reduces the weight combinations to 924, while retaining 66 unique solutions, therefore the ratio of the solutions to the complete search spaces is increased nearly 4 times. Fig. 4.7.(c) shows in a half-log plot how the numbers of weight combinations and unique network solutions change with varying bit-ratios. Equal bit ratios is optimal.

In Fig. 4.4 we train the Conv2 networks 100 times on Cifar-10 and plot the distribution of the training and test losses and accuracies. We plot these results when using the *bi-half* model optimization with optimal transport and by training the network using the standard sign function. The figure shows the *bi-half* method consistently finds better

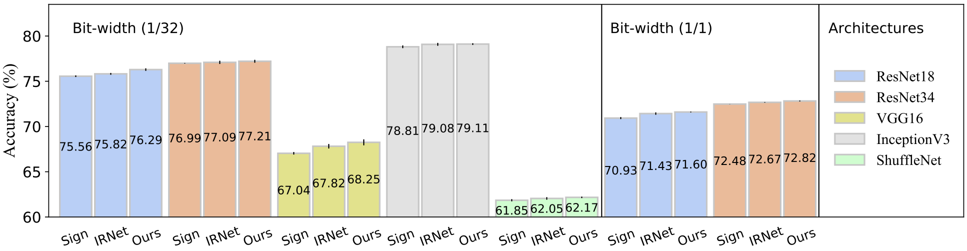
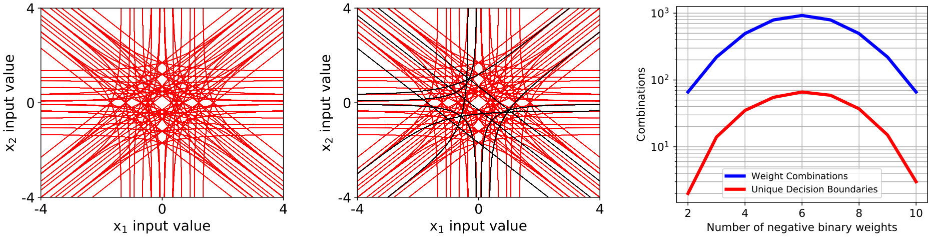


Figure 4.6: **Architecture variations: Different architectures on Cifar-100.** We evaluate on Cifar-100 over 5 different architectures: VGG16 [34], ResNet18 [35], ResNet34 [35], InceptionV3 [36], ShuffleNet [37]. We compare sign [5], IR-Net [1] and our *bi-half*. The 1/32 and 1/1 indicate the bit-width for weights and for activations, where 1/1 means we quantize both the weights and the activations to binary code values. Our method achieves competitive accuracy across different network architectures.



Weight combinations: 100% (4096).

Unique solutions: 100% (76).

(a) Full binary network

Weight combinations: 22.5% (924).

Unique solutions: 86.8% (66).

(b) *Bi-half* network

(c) Solution space vs bit-ratios

Figure 4.7: **Empirical analysis (c): Optimization benefits.** *Bi-half* regularization: 2D example for a 12-parameter fully connected binary network $\sigma(\mathbf{w_2}^\top \sigma(\mathbf{w_1}^\top \mathbf{x} + \mathbf{b_1}))$, where $\sigma(\cdot)$ is a sigmoid nonlinearity. Weights are in $\{-1, 1\}$. (a) Enumeration of all decision boundaries for 12 binary parameters ($4096 = 2^{12}$ combinations). (b) Weight combinations and unique solutions when using our *bi-half* constraint. (c) The weight combinations and unique decision boundaries for various bit-ratios. When the number of negative binary weights is 6 on the x-axis, we have equal bit-ratios, which is the optimal ratio. Using the bi-half works as a regularization, reducing the search-space while retaining the majority of the solutions.

solutions with lower training and test losses and higher training and test accuracy. To better visualize this trend we sort the values of the losses for our *bi-half* and the baseline sign method over the 100 repetitions and plots them next to each other. On average the *bi-half* finds better optima.

4.4.3. Architecture variations

In Table 4.2 we compare the Sign [5], IR-Net [1] and our *bi-half* on four shallow Conv2/4/6/8 networks on Cifar-10 (averaged over 5 trials). As the networks become deeper, the proposed *bi-half* method consistently outperforms the other methods.

In Fig. 4.6, we further evaluate our method on Cifar-100 over 5 different architectures: VGG16 [34], ResNet18 [35], ResNet34 [35], InceptionV3 [36], ShuffleNet [37]. Our method is slightly more accurate than the other methods, especially on the VGG16 architecture, it never performs worse.

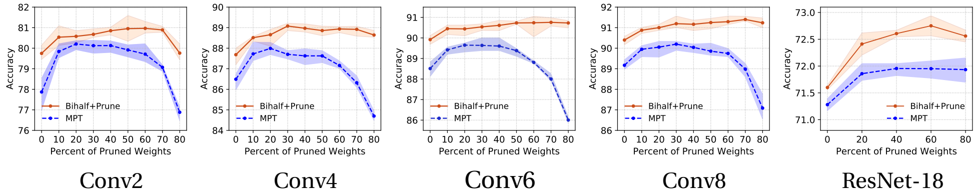


Figure 4.8: **Comparison with state-of-the-art (b): Pruned networks.** Test accuracy of Conv2/4/6/8 on CIFAR-10, and ResNet-18 on CIFAR-100 when varying the % pruned weights. We compare with the MPT baseline [28] using binary weight masking and the sign function. Having equal +1 and −1 ratios is also optimal when the networks rely on pruning and that our optimal transport optimization can easily be adapted to work in combination with pruning.

Table 4.2: **Architecture variations.** Accuracy comparison of sign [5], IR-Net [1] and our *bi-half* on Conv2/4/6/8 networks using Cifar-10, over 5 repetitions. As the depth of the network increases, the accuracy of our method increases.

Method	Conv2	Conv4	Conv6	Conv8
Sign	77.86± 0.69	86.49± 0.24	88.51± 0.35	89.17±0.26
IR-Net	78.32± 0.25	87.20± 0.26	89.61± 0.11	90.06±0.06
Bi-half (ours)	79.25± 0.28	87.68± 0.32	89.92± 0.19	90.40±0.17

4.4.4. COMPARISON WITH STATE-OF-THE-ART

(a) Comparison on ImageNet. For the large-scale ImageNet dataset we evaluate a ResNet-18 and ResNet-34 backbone [35]. Table 5.3 shows a number of state-of-the-art quantization methods over ResNet-18 and ResNet-34, including: ABC-Net [15], XNOR [5], BNN+ [10], Bi-Real [12], RBNN [38], XNOR++ [4], IR-Net [1], and Real2binary [39]. Of all the methods, RBNN is the closest in accuracy to our *bi-half* model. This is because RBNN relies on the sign function but draws inspiration from hashing, and adds an activation-aware loss to change the distribution of the activations before binarization. On the other hand, our method uses the standard classification loss but outperforms most other methods by a large margin on both ResNet-18 and ResNet-34 architectures.

(b) Comparison on pruned networks. In Fig. 4.8 we show the effect of our *bi-half* on pruned models. Following the MPT method [28] we learn a mask for the binary weights to prune them. However, in our *bi-half* approach for pruning we optimize using optimal transport for equal bit ratios in the remaining unpruned weights. We train shallow Conv2/4/6/8 networks on CIFAR-10, and ResNet-18 on CIFAR-100 while varying the percentage of pruned weights. Each curve is the average over five trials. Pruning consistently finds subnetworks that outperform the full binary network. Our *bi-half* method with optimal transport retains the information entropy for the pruned subnetworks, and consistently outperforms the MPT baseline using the sign function for binarization.

4.5. CONCLUSION

We focus on binary networks for their well-recognized efficiency and memory benefits. To that end, we propose a novel method that optimizes the weight binarization by align-

Table 4.3: **Comparison with state-of-the-art (a): ImageNet results.** We show Top-1 and Top-5 accuracy on ImageNet for a number of state-of-the-art binary networks. Sign is our baseline by carefully tuning the hyper-parameters. Our proposes *bi-half* model consistently outperforms the other binarization methods on this large-scale classification task.

Backbone	Method	Bit-width (W/A)	Top-1(%)	Top-5(%)
ResNet-18	FP	32/32	69.3	89.2
	ABC-Net	1/1	42.7	67.6
	XNOR	1/1	51.2	73.2
	BNN+	1/1	53.0	72.6
	Least-squares	1/1	58.9	81.4
	XNOR++	1/1	57.1	79.9
	IR-Net	1/1	58.1	80.0
	RBNN	1/1	59.9	81.9
	Sign (Baseline)	1/1	59.98	82.47
	Bi-half (ours)	1/1	**60.40**	**82.86**
ResNet-34	FP	32/32	73.3	91.3
	ABC-Net	1/1	52.4	76.5
	Bi-Real	1/1	62.2	83.9
	IR-Net	1/1	62.9	84.1
	RBNN	1/1	63.1	84.4
	bi-half (ours)	1/1	**64.17**	**85.36**

ing a real-valued proxy weight distributions with an idealized distribution using optimal transport. This optimization allows us to test which prior bit ratio is preferred in a binary network, and we show that the equal bit ratios as advertised by [1] do indeed work better. Additionally, we show that our optimal transport binarization has optimization benefits such as: reducing the search space and leading to better local optima. Finally, we demonstrate competitive performance when compared to state-of-the-art, and show improved accuracy on 3 different datasets, using a variety of different architectures. We also show accuracy gains when using network pruning.

REFERENCES

[1] H. Qin, R. Gong, X. Liu, M. Shen, Z. Wei, F. Yu, and J. Song, *Forward and backward information retention for accurate binary neural networks,* in *CVPR* (2020).

[2] J. W. Peters and M. Welling, *Probabilistic binary neural networks,* CoRR (2018).

[3] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, *Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,* CoRR (2016).

[4] A. Bulat and G. Tzimiropoulos, *Xnor-net++: Improved binary neural networks,* (2019), arXiv:1909.13863 [cs.CV] .

[5] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *Xnor-net: Imagenet classification using binary convolutional neural networks,* in *European conference on computer vision* (Springer, 2016) pp. 525–542.

[6] C. Villani, *Topics in optimal transportation,* 58 (American Mathematical Soc, 2003).

[7] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, *Binary neural networks: A survey,* Pattern Recognition (2020).

[8] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder, *Latent weights do not exist: Rethinking binarized neural network optimization,* in *Advances in neural information processing systems* (2019) pp. 7533–7544.

[9] M. Courbariaux, Y. Bengio, and J.-P. David, *Binaryconnect: Training deep neural networks with binary weights during propagations,* in *NeurIPS* (2015).

[10] S. Darabi, M. Belbahri, M. Courbariaux, and V. P. Nia, *Bnn+: Improved binary network training,* ICLR (2019).

[11] Z. Li, B. Ni, W. Zhang, X. Yang, and W. Gao, *Performance guaranteed network acceleration via high-order residual quantization,* in *Proceedings of the IEEE International Conference on Computer Vision* (2017) pp. 2584–2592.

[12] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, *Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm,* in *ECCV* (2018).

[13] Z. Yang, Y. Wang, K. Han, C. Xu, C. Xu, D. Tao, and C. Xu, *Searching for low-bit weights in quantized neural networks,* in *NeurIPS* (2020).

[14] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, *Reactnet: Towards precise binary neural network with generalized activation functions,* in *European Conference on Computer Vision* (Springer, 2020) pp. 143–159.

[15] X. Lin, C. Zhao, and W. Pan, *Towards accurate binary convolutional neural network,* in *NeurIPS* (2017).

[16] K. Han, Y. Wang, Y. Xu, C. Xu, E. Wu, and C. Xu, *Training binary neural networks through learning with noisy supervision,* in *International Conference on Machine Learning* (PMLR, 2020) pp. 4017–4026.

[17] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, *Pruning neural networks at initialization: Why are we missing the mark?* CoRR (2020).

[18] Q. Huang, K. Zhou, S. You, and U. Neumann, *Learning to prune filters in convolutional neural networks,* in *WACV* (2018).

[19] J. Lin, Y. Rao, J. Lu, and J. Zhou, *Runtime neural pruning,* in *NeurIPS* (2017).

[20] X. Xiao, Z. Wang, and S. Rajasekaran, *Autoprune: Automatic network pruning by regularizing auxiliary parameters,* in *NeurIPS* (2019).

[21] M. Ye, C. Gong, L. Nie, D. Zhou, A. Klivans, and Q. Liu, *Good subnetworks provably exist: Pruning via greedy forward selection,* ICML (2020).

[22] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, *Rigging the lottery: Making all tickets winners,* ICML (2020).

[23] J. Frankle and M. Carbin, *The lottery ticket hypothesis: Finding sparse, trainable neural networks,* ICLR (2020).

[24] E. Malach, G. Yehudai, S. Shalev-Shwartz, and O. Shamir, *Proving the lottery ticket hypothesis: Pruning is all you need,* CoRR (2020).

[25] H. Zhou, J. Lan, R. Liu, and J. Yosinski, *Deconstructing lottery tickets: Zeros, signs, and the supermask,* in *NeurIPS* (2019).

[26] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,* (2016), arXiv:1510.00149 [cs.CV] .

[27] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, *Rethinking the value of network pruning,* ICLR (2019).

[28] J. Diffenderfer and B. Kailkhura, *Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network,* ICLR (2021).

[29] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,* in *Proceedings of the IEEE international conference on computer vision* (2015) pp. 1026–1034.

[30] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images,* Tech. Rep. (Citeseer, 2009).

[31] J. Deng, W. Dong, R. Socher, and L. J. Li, *Imagenet: A large-scale hierarchical image database,* in *CVPR* (2009).

[32] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari, *What's hidden in a randomly weighted neural network?* in *CVPR* (2020).

[33] I. Loshchilov and F. Hutter, *Sgdr: Stochastic gradient descent with warm restarts,* arXiv preprint arXiv:1608.03983 (2016).

[34] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition,* ICLR (2015).

[35] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 770–778.

[36] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision,* in *CVPR* (2016) pp. 2818–2826.

[37] X. Zhang, X. Zhou, M. Lin, and J. Sun, *Shufflenet: An extremely efficient convolutional neural network for mobile devices,* in *CVPR* (2018) pp. 6848–6856.

[38] M. Lin, R. Ji, Z. Xu, B. Zhang, Y. Wang, Y. Wu, F. Huang, and C.-W. Lin, *Rotated binary neural network,* ECCV (2020).

[39] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, *Training binary neural networks with real-to-binary convolutions,* arXiv preprint arXiv:2003.11535 (2020).

5

UNDERSTANDING WEIGHT MAGNITUDE HYPERPARAMETERS IN TRAINING BINARY NETWORKS

Binary Neural Networks (BNNs) are compact and efficient by using binary weights instead of real-valued weights. Current BNNs use latent real-valued weights during training, where hyper-parameters are inherited from real-valued networks. The interpretation of several of these hyperparameters is based on the magnitude of the real-valued weights. For BNNs, however, the magnitude of binary weights is not meaningful, and thus it is unclear what these hyperparameters actually do. One example is weight-decay, which aims to keep the magnitude of real-valued weights small. Other examples are latent weight initialization, the learning rate, and learning rate decay, which influence the magnitude of the real-valued weights. The magnitude is interpretable for real-valued weights, but loses its meaning for binary weights. In this paper we offer a new interpretation of these magnitude-based hyperparameters based on higher-order gradient filtering during network optimization. Our analysis makes it possible to understand how magnitude-based hyperparameters influence the training of binary networks which allows for new optimization filters specifically designed for binary neural networks that are independent of their real-valued interpretation. Moreover, our improved understanding reduces the number of hyperparameters, which in turn eases the hyperparameter tuning effort which may lead to better hyperparameter values for improved accuracy.

5.1. INTRODUCTION

A Binary Neural Network (BNN) weight is a single bit: -1 or $+1$, which is compact and efficient, enabling applications on, for example, edge devices. Yet, training BNNs using gradient descent is difficult because of the discrete binary values. Thus, BNNs are often [1–3] optimized with so called 'latent', real-valued weights, which are discretised to -1 or $+1$ by, *e.g.*, taking the positive or negative sign of the real value.

The latent weight optimization depends on several essential hyperparameters, such as their initialization, learning rate, learning rate decay, and weight decay. These hyperparameters are important for BNNs, as shown for example in [3], and also by [4], who both improve BNN accuracy by better tuning these hyperparameters.

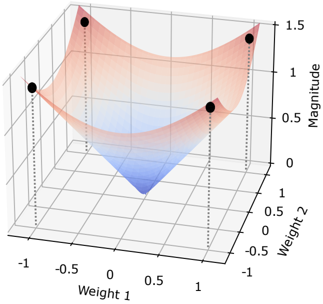


Figure 5.1: Changes in real-valued weights change their magnitude. For binary weights, however, the magnitude will never change and magnitude-based hyperparameters need reinterpretation.

In this paper we investigate the latent weight hyperparameters used in a BNN, including initialization, learning rate, learning rate decay, and weight decay. All these hyperparameters influence the magnitude of the latent weights. Yet, as illustrated in Fig 5.1, in a BNN, the *binary weights are* -1 *or* $+1$*, which always have a constant magnitude and thus magnitude-based hyperparameters lose their meaning.* We draw inspiration from the seminal work of [5], who reinterpret latent weights from an inertia perspective and state that latent weights do not exist. Thus, the magnitude of latent weights also does not exist. Here, we investigate what latent weight-magnitude hyperparameters mean for a BNN, how they relate to each other, and what justification they have. We provide a gradient-filtering perspective on latent weight hyperparameters which main benefit is a simplified setting: fewer hyperparameters to tune, achieving similar accuracy as current, more complex methods.

5.2. RELATED WORK

Latent weights in BNNs. By tying each binary weight to a latent real-valued weight, continuous optimization approaches can be used to optimize binary weights. Some methods minimize the quantization error between a latent weight and its binary variant [6, 7]. Others focus on gradient approximation [8–10], or on reviving dead weights [11, 12], or on entropy regularization [13] or a loss-aware binarization [14, 15]. These works directly apply traditional optimization techniques inspired by real-valued network such as weight decay, learning rate and its decay, and optimizers. The summary of [16] gives a good overview of these training techniques in BNNs. Recently, some papers [3, 4, 17, 18] noticed that the interpretation of these optimization techniques does not align with the binary weights of BNNs [19, 20]. Here, we aim to shed light on why, by explicitly analyzing latent weight-magnitude hyperparameters in a BNN.

Latent weight magnitudes. Several techniques exploit the magnitude of the latent weights during BNN optimization. Latent weights clipping is proposed in [21] and followed by its extensions [22, 23] to clip the latent weights within a $[-1,1]$ interval to prevent the magnitude of latent weights from growing too large. Gradient clipping [21, 24, 25] stops gradient flow if the magnitude of latent weight is too large. Work on latent weight scaling [25, 26] standardizes the latent weights to a pre-defined magnitude. Excellent results are achieved by a two-step training strategy [2, 4] that in the first step trains the network from scratch using only binarizing activations with weight decay, and then in the second step they fine-tune by training without weight decay. Our method reinterprets the meaning of the magnitude based weight decay hyperparameter in optimizing BNNs from a gradient filtering perspective, offering similar accuracy as two step training with a simpler setting, using just a single step.

Optimization by gradient filtering. Gradient filtering is a common approach used to tackle the noisy gradient updates caused by minibatch sampling. Seminal algorithms including Momentum [27] and Adam [28] which use a first order infinite impulse response filter (IIR), *i.e.* exponential moving average (EMA) to smooth noisy gradients. [29] takes this one step further and introduces the Filter Gradient descent Framework that can use different types of filters on the noisy gradients to make a better estimation of the true gradient. In binary network optimization, Bop [5] and its extension [30] introduce a threshold to compare with the smoothed gradient by EMA to determine whether to flip a binary weight. In our paper, we build on second order gradient filtering techniques to reinterpret the hyperparameters that influence the latent weight updates.

Sound optimization approaches. Instead of using heuristics to approximate gradient descent on discrete binary values, several works take a more principled approach. [31] propose a probabilistic training method for BNN, and [32] present a theoretical understanding of straight through estimators (STE) [33]. [34] propose a Bayesian perspective and [35] formulate a noisy quantizer. Even though these approaches provide more theoretical justification in optimizing BNNs, they are more complex by either relying on stochastic settings or discrete relaxation training procedures. Moreover, these methods do not (yet) empirically reach a similar accuracy as current mainstream heuristic methods [2, 8]. In our paper, we build on the mainstream approaches, to get good empirical results, but add a better understanding of their properties, taking a step towards better theoretical understanding of empirical approaches.

We start with a latent weights BNN and convert it to an equivalent latent-weight free setting, as in [5]. To do this, we use a magnitude independent setting, which means that no gradient-clipping or scaling based on the channel-wise mean of the latent-weights is used.

BNN setup. We use Stochastic Gradient Descent (SGD) with weight decay and momentum as a starting point, as this is a commonly used setting, see [6], [8], [25]. Our setup is as follows:

$$w_0 = \text{init}() \tag{5.1}$$

$$m_i = (1-\gamma)m_{i-1} + \gamma\nabla_{\theta_i}, \tag{5.2}$$

$$w_i = w_{i-1} - \epsilon(m_i + \lambda w_{i-1}) \tag{5.3}$$

$$\theta_i = \text{sign}(w_i) \tag{5.4}$$

$$\text{sign}(x) = \begin{cases} -1, & \text{if } x < 0; \\ +1, & \text{if } x > 0; \\ \text{random}\{-1,+1\} & \text{otherwise.} \end{cases} \tag{5.5}$$

Here, w_i is a latent weight at iteration i which is initialized at w_0. θ_i is a binary weight, ϵ is the learning rate, λ is the weight decay factor, m_i is the momentum exponentially weighted moving average with $m_{-1} = 0$ and discount factor γ, ∇_{θ_i} is the gradient over the binary weight and random$\{-1,+1\}$ is a uniformly randomly sampled -1 or +1.

We then convert to the latent-weight free setting of [5] where latent weights are interpreted as accumulating negative gradients. We introduce $g_i = -w_i$, which allows working with gradients instead of with latent weights. We can then write (5.3) as follows

$$g_i = g_{i-1} + \epsilon(m_i - \lambda g_{i-1}). \tag{5.6}$$

Latent weight initialization. To investigate latent weight initialization we unroll the the recursion in (5.6) by writing it out as a summation:

$$g_i = (1-\epsilon\lambda)g_{i-1} + \epsilon m_i \quad = \quad \epsilon \sum_{r=0}^{i} (1-\epsilon\lambda)^{i-r} m_r. \tag{5.7}$$

Latent-weights are typically initialized using real-valued weight initialization techniques [36, 37]. However, since we now interpret latent weights as accumulated gradients, we argue to also initialize them as gradient accumulation techniques such as Momentum [27] and simply initialize $w_0 = g_0 = 0$, because at initialization there is no preference for negative or positive gradients, and their expectation is 0. We do not use a bias-correction as done in [28] because in practice we noticed that gradient magnitudes are large in the first few iterations. Applying bias correction increases this effect, which had a negative effect on training. To prevent all binary weights θ to start at the same value, we use the stochastic sign function in (5.5) that randomly chooses a sign when the input is exactly 0.

Learning rate and weight decay. The learning rate ϵ appears in two places in (5.7): once outside the summation, and once inside the summation. The ϵ outside the summation can only scale the latent weight and will not influence outcome of the sign in (5.4) as

$$\text{sign}\left(\epsilon \sum_{r=0}^{i} (1-\epsilon\lambda)^{i-r} m_r\right) = \text{sign}\left(\sum_{r=0}^{i} (1-\epsilon\lambda)^{i-r} m_r\right). \tag{5.8}$$

Thus, the leftmost ϵ can be removed, or set randomly without influencing the training process.

For the ϵ inside the summation of (5.7), it appears together with the weight decay term λ. Thus, there are two free hyperparameters that only control one factor, therefore one of them is redundant and can use a single combined hyperparameter $\alpha = \epsilon\lambda$. Instead of setting a value for the learning rate ϵ, and setting a value for the weight decay λ, we now only have to set a single value for α. Since (5.8) shows us that we can freely scale the sum with any constant factor, we scale it with α, as

$$g_i = \alpha \sum_{r=0}^{i} (1-\alpha)^{i-r} m_r, \tag{5.9}$$

which allows us to re-write the sum with a recursion, as an exponential moving average (EMA) as

$$g_i = (1-\alpha) g_{i-1} + \alpha m_i, \tag{5.10}$$

where $g_{-1} = 0$. This shows that for BNNs under magnitude independent conditions, SGD with weight decay is just a exponential moving average. This gives a magnitude-free justification for using weight decay since its actual role is to act as the discount factor in an EMA. Note that it is no longer possible to set α to 0 since then there are no updates anymore, but setting to a small (10^{-20}) number will essentially work the same. The meaning of α is now clear, as in the EMA it controls how much to take the past into account.

Learning rate decay. There no longer is a learning rate to be decayed, however, since learning rate decay scales the learning rate and $\alpha = \epsilon\lambda$, now the learning rate decay directly scales α, so from now on we apply it to alpha and will refer to it as α-decay. This also helps better explain its function: α-decay increases the window size during training, causing the filtered gradient to become more stable and allowing the training to converge.

Momentum. Now adding back the momentum term of (5.2) in the original setup yields

$$m_i = (1-\gamma) m_{i-1} + \gamma \nabla_{\theta_i}, \tag{5.11}$$

$$g_i = (1-\alpha) g_{i-1} + \alpha m_i, \tag{5.12}$$

$$\theta_i = -\text{sign}(g_i). \tag{5.13}$$

Thus, SGD with weight decay and momentum is smoothing the gradient twice with an EMA filter.

Latent weight optimization as a second order linear infinite impulse response filter. EMAs are a specific type of linear Infinite Impulse Response (IIR) Filter [38]. Linear filters are filters that compute an output based on a linear combination of current and past inputs and past outputs. The general definition is given as a difference equation:

$$y_i = \frac{1}{a_0}(b_0 x_i + b_1 x_{i-1} + \ldots + b_P x_{i-P} - a_1 y_{i-1} - a_2 y_{i-2} - \ldots - a_P y_{i-Q}), \tag{5.14}$$

where i is the time step, y_i are the outputs, x_i are the inputs, a_j and b_j are the filter coefficients and P and Q are the maximum of iterations the filter looks back at the inputs and outputs to compute the current output. The maximum of P and Q defines the order of

	Learning rate	Learning rate decay	Init	Momentum	Weight decay	Scaling	Clipping
Latent:	ϵ	ϵ-decay	w_0	γ	λ	✓	✓
Filtered:	–	α-decay	–	γ	α	–	–

Table 5.1: Hyperparameters used in the latent weight view versus our filtered gradients perspective. Our filtered perspective reduces the number of hyperparameters from 7 to 3.

the filter. An EMA only looks at the previous output and the current input, so is therefore a first order IIR filter. Expressing an EMA as a filter looks as follows:

$$y_i = (1-\alpha)y_{i-1} + \alpha x_i = \frac{1}{a_0}(b_0 x_i - b_1 \cdot x_{i-1} - a_1 y_{i-1}), \quad b = \begin{bmatrix}\alpha\\0\end{bmatrix}, a = \begin{bmatrix}1\\\alpha-1\end{bmatrix}. \tag{5.15}$$

In our optimizer we have a cascade of two EMAs applied in series to the same signal which can be represented by a filter with the order being the sum of the orders of the original filters. To get the new a and b vectors the original ones are convolved with each other. In our case this gives:

$$b = \begin{bmatrix}\gamma\\0\end{bmatrix} \star \begin{bmatrix}\alpha\\0\end{bmatrix} = \begin{bmatrix}\alpha\gamma\\0\\0\end{bmatrix}, \quad a = \begin{bmatrix}1\\\gamma-1\end{bmatrix} \star \begin{bmatrix}1\\\alpha-1\end{bmatrix} = \begin{bmatrix}1\\(\alpha-1)+(\gamma-1)\\(\alpha-1)\cdot(\gamma-1)\end{bmatrix}, \tag{5.16}$$

when applied to our gradient filtering setting in (5.12) gives the difference equation:

$$g_i = \alpha\gamma\nabla_{\theta_i} - (\alpha+\gamma-2)g_{i-1} - (\alpha-1)(\gamma-1)g_{i-2} \tag{5.17}$$

Thus, in a magnitude independent setting, SGD with weight decay and momentum is equivalent to a 2nd order linear IIR filter. Note that α and γ have the same function: Without α decay, the values for α and γ can be swapped. This filtering perspective opens up new methods of analysis for optimizers.

Main takeaway. Our re-interpretations reduces the 7 hyper parameters in the latent weight view with SGD, to only 3 hyperparameters in our filtering view, see Table 5.1.

5.3. Experiments

We empirically validate our analysis on CIFAR-10, using the BiRealNet-20 architecture [8]. Unless mentioned otherwise the networks were optimized using SGD for both the real-valued and binary parameters with as hyperparameters: learning rate=0.1, momentum with $\gamma = (1-0.9)$, weight decay=10^{-4}, batch size=256 and cosine learning rate decay and cosine alpha decay. We analyze the weight flip ratio at every update, which is also known as the FF ratio ([4]).

$$\mathbf{I}_{\text{FF}} = \frac{|\text{sign}(w_{i+1}) - \text{sign}(w_i)|_1}{2} \qquad \mathbf{FF}_{\text{ratio}} = \frac{\sum_{l=1}^{L}\sum_{w\in W_l}\mathbf{I}_{\text{FF}}}{N_{\text{total}}} \tag{5.18}$$

where w_i is a latent weight at time i, L the number of layers, W_l the weights in layer l, and N_{total} the total number of weights.

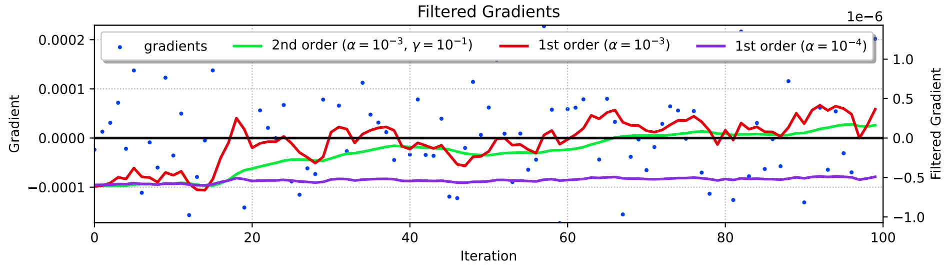
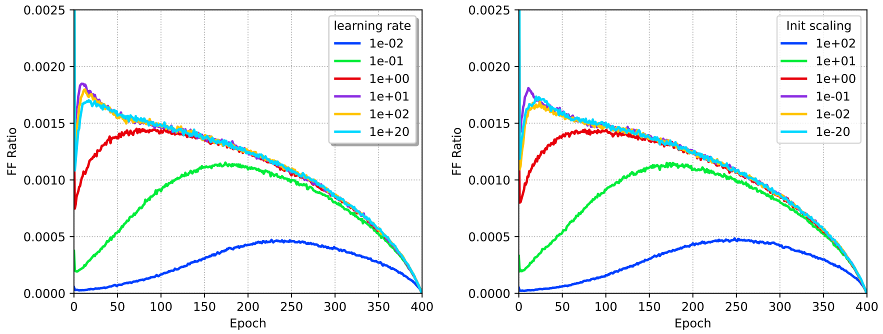


Figure 5.2: Gradients and filtered gradients for a first and second order filter in a single epoch on CIFAR-10. For better visualisation, the filter outputs are scaled up to a similar range as the unfiltered gradients. It can be seen that the unfiltered gradients are noisy and that the filtered outputs are smoother. The second order filter reduces the noise even further compared to the first order filter.



(a) Varying learning rates, constant init.

(b) Vary init, constant learning rate (ϵ=1.0).

Figure 5.3: In the magnitude independent setting, scaling the learning rate has the exact same effect on the flipping ratio as scaling the initial latent-weights by the inverse.

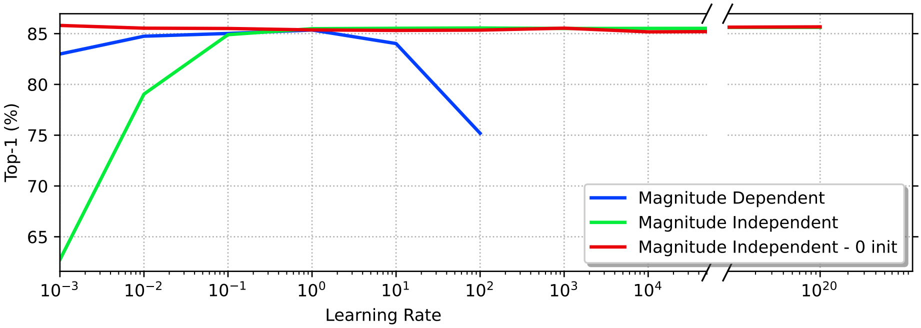


Figure 5.4: Learning rate effect on accuracy for three settings. (a). standard SGD. (b) Magnitude independent in (5.3), by removing clipping and scaling. (c) Magnitude independent with latent weight initializes of 0 in (5.7). Setting (a) has just a single optimum. The accuracy in setting (b) is sensitive to small learning rates. For setting (c), accuracy is independent of the learning rate.

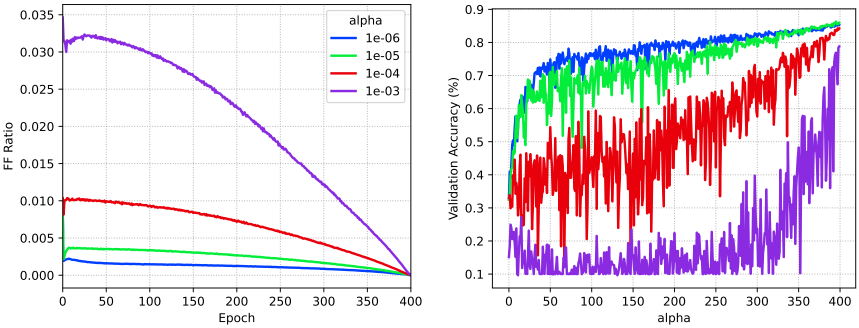


Figure 5.5: Bit flipping (FF) ratio and accuracy for varying alphas from eq 5.17. BNNs are sensitive to alpha, similar to how they are sensitive to weight decay. Tuning alpha is essential.

1st order vs 2nd order We visually compare filter orders by sampling real gradients from a single binary weight trained on CIFAR-10 in Figure 5.2. For the same α, a 1st order filter is more noisy than a 2nd order filter. This may cause the binary weight to oscillate, even though the larger trend is that it should just flip once. To reduce these oscillations with a 1st order filter requires a smaller alpha. This, however, causes other problems because α determines the window size of past gradients and with a smaller α many more gradients are used. This means that it takes much longer for a trend in the gradients to effect the binary weight. Instead, the 2nd order filter has both benefits: it can filter out high frequency noise while still able to react quicker to changing trends.

Magnitude independent learning rate vs initialization In Figure 5.3 we show the bit flipping ratio for the learning rate ϵ and the initialization g_0. Multiplying ϵ with some scaling factor s is the same as dividing g_0 by s: $\text{sign}\left((1-\epsilon\lambda)^i g_0 + s\epsilon \sum_{r=1}^{i}(1-\epsilon\lambda)^{i-r} m_r\right) =$

sign$\left((1-\epsilon\lambda)^i \frac{g_0}{s} + \epsilon\sum_{r=1}^{i}(1-\epsilon\lambda)^{i-r} m_r\right)$, because the magnitude has no effect on the sign. Larger ϵ in Figure 5.3(a) and smaller g_0 in Figure 5.3(b) are independent to scaling and have similar flipping ratios. A too small ϵ or too large g_0 do not reach the same flipping ratios, because their ratio is insufficient to update the binary weights. For sufficiently large ratios it means that scaling both ϵ and g_0 has no effect on training, but also that scaling the ϵ or scaling g_0 with the inverse is identical: as seen by comparing the two plots in Figure 5.3.

Sensitivity to hyperparameters for weight magnitude (in)dependence We evaluate SGD in the standard magnitude dependent setting with clipping and scaling vs a magnitude independent setting with initializing the latent-weights to zero. To keep the effect of weight decay constant, we scale the weight decay factor inversely with the learning rate. Results in Figure 5.4 show that for the standard magnitude dependent setting the learning rate ϵ has to be carefully balanced. A too small ϵ w.r.t. to the initial weights inhibits learning; while a too large ϵ will push latent-weights to the clipping region and will stop updating. In the magnitude independent setting there is no clipping. A too small ϵ, however is still problematic because the magnitudes of the gradients are smaller when not using the scaling factor and the accuracy drops significantly. When initializing to zero, as we propose, this problem disappears, because there is no initial weight to hinder training and all learning rates perform equal.

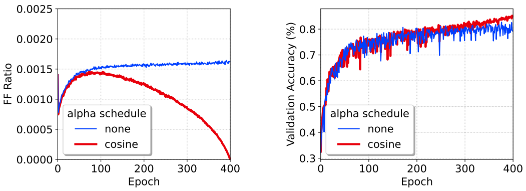


Figure 5.6: Flipping rate and accuracy for alpha decay. With decay the FF ratio goes to zero. Without decay, the flipping will continue, preventing convergence, reducing accuracy.

Alpha In Fig 5.5 we vary α from eq 5.17. Alpha strongly influences training. A too large α causes too many binary weight flips per update, which hinders converging. A too small α makes the network converge too quickly, which hurts the end result. Tuning α is essential.

Alpha decay For proper convergence the flipping (FF) ratio should go to zero. We transform learning rate decay to alpha decay. When α becomes smaller, the gradients will change less, causing fewer flips, forcing the network to converge.

See the plots in Figure 5.6 where one network has been trained with cosine alpha decay and one without alpha decay. With and without alpha decay both seem to perform

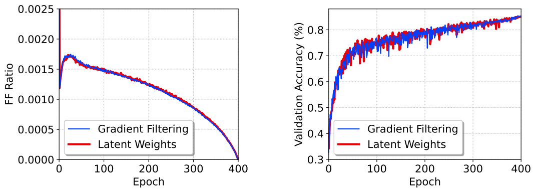


Figure 5.7: Equivalence of latent weights ((5.3)) and our gradient filtering ((5.17)). They are empirically equivalent.

Table 5.2: State-of-the-art on CIFAR-10. The * denotes that we re-ran these experiments ourselves.

Method	Training Strategy	Bit-width (W/A)	Top-1 Acc(%)
FP		32/32	91.7
DoReFa-Net [39]	One step	1/1	79.3
DSQ [40]		1/1	84.1
IR-Net [25]		1/1	**86.5**
Bi-Real* [8]		1/1	85.0
Bi-Real + Our filtering optimizer		1/1	**86.5**
Bi-Real* [8]	Two step	1/1	**86.7**

Table 5.3: Comparison with state-of-the-art on Imagenet.

Method	Training Strategy	Top-1 Acc(%)	Top-5 Acc(%)
CI-BCNN [41]	One step	59.9	84.2
Binary MobileNet [42]		60.9	82.6
MoBiNet [43]		54.4	77.5
EL [17]		56.4	–
MeliusNet29 [44]		65.8	–
ReActNet-A + Our filtering optimizer		**69.7**	**88.9**
StrongBaseline [3]	Two step	60.9	83.0
Real-to-Binary [3]		65.4	86.2
ReActNet-A [2]		69.4	88.6
ReActNet-A-AdamBNN [4]		**70.5**	**89.1**

well at the start of training, however, the variant without alpha decay plateaus at the end of training while the BNN with alpha decay converges better and continues improving, leading to a better end result.

Equivalent interpretation Figure 5.7 shows empirical validation with matching hyperparameters that the SGD setting using latent weights in (5.3) is equivalent to our gradient filtering interpretation in (5.17) that no longer uses latent weights,

5.3.1. VALIDATION OF EQUIVALENCE TO THE CURRENT STATE OF THE ART

We validate on for CIFAR-10 and Imagenet that our filtering-based optimizer is similar to the current state of the art. Several current methods use an expensive two-step optimization step. We aim to show the value of our re-interpretation by showing similar accuracy but only in a single step.

CIFAR-10: We train all networks for 400 epochs. As data augmentation we use padding of 4 pixels, followed by a 32x32 crop and random horizontal flip. We use Bi-RealNet-20, and for the real-valued parameters and latent-weights when used, we use SGD with a learning rate of 0.1 with cosine decay, momentum of 0.9 and on the non-BN parameters a weight decay of 10^{-4}. For our filtering-based optimizer we used an alpha of 10^{-3} with cosine decay and a gamma of 10^{-1}. Results in Table 5.2 show that our re-interpretation achieves similar accuracy.

Imagenet: We follow [4]: We train for 600K iterations with a batch size of 510. For the real-valued parameters we use Adam with a learning rate of 0.0025 with linear learning rate decay. For the binary parameters we use our 2nd order filtering optimizer with $\alpha = 10^{-5}$, which we decay linearly and $\gamma = 10^{-1}$. We do not use two-step training to pre-train the latent-weights. Results in Table 5.3, show that ReActNet-A with our optimizer compares well to other one step training methods. It approaches the accuracy of two step training approaches, albeit without an additional expensive second step of training.

5.3.2. EMPIRICAL ADVANTAGE OF HAVING FEWER HYPERPARAMETER TO TUNE

Our filtering perspective significantly reduces the number of hyperparameters (Table 5.1). Here, we empirically verify the computational benefit of having fewer hyperparameters

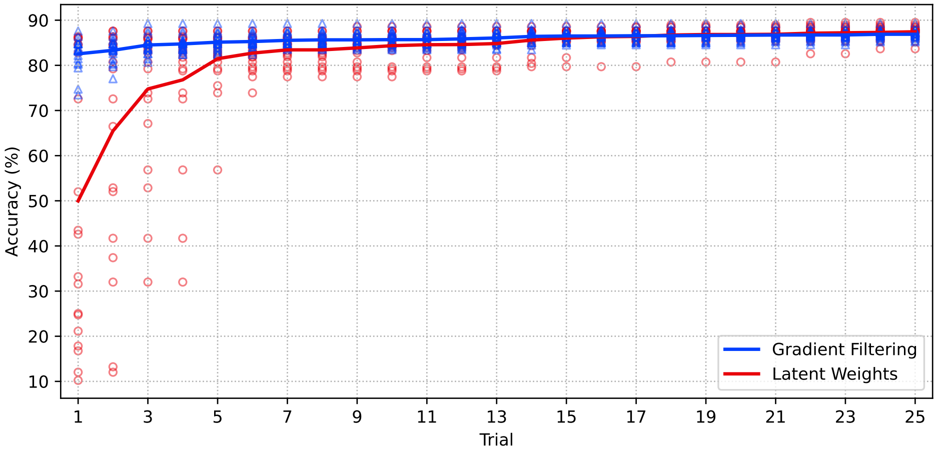


Figure 5.8: Bayesian hyperparameter optimization for our gradient filtering vs. latent weights. We optimize the hyperparameters of Table 5.1, except learning rate decay and alpha decay, as they had no influence. The scatter plot show the best achieved result so far at each trial for all runs. The lines show the average best result, up to the current trial. Both methods perform equally for good hyperparameters. Our gradient filtering view needs much less trials.

to tune when applied in a setting where the hyperparameters are unknown. To show generalization to other architectures and modality we use an audio classification task [45] with a fully connected network. Specifically, we use 4 layers with batch normalization of which the first layer is not binarized. For the latent weights we optimize 6 hyperparameters, and for our filtering perspective we optimize two hyperparameters, see Table 5.1. We did not tune learning rate decay as this had no effect on both methods. To fairly compare hyperparameter search we used Bayesian optimization [46]. The results for 25 trials of tuning, averaged over 20 independent runs are in Figure 5.8. We confirm that both perspectives achieve similar accuracy when their hyperparameters are well tuned. Yet, for the latent weights, it takes on average around 10 more trials when compared to the gradient filtering. This means that the latent weights would have to train many more networks, which on medium-large datasets such as Imagenet would already take several days to converge. In contrast, the gradient filtering requires much less time and energy to find a good model.

5.4. DISCUSSION AND LIMITATIONS

One limitation of our work is that we do not achieve “superior performance” in terms of accuracy. Our approach merely matches the state of the art results. Note, however, that our goal is to provide insight into how SGD and its hyperparameters behave, not to improve accuracy. Our analysis ended up with an optimizer with less hyperparameters, that also have a better explanation in the context of BNN optimization leading to simpler, more elegant methods. Our main empirical contribution is in the significant computational reduction in hyperparameter tuning.

Another perceived limitation is that our new proposed optimizer can be projected back to a specific setting within the current SGD with latent-weights interpretation. Thus, our analysis might not be needed. While it is true that latent-weights can also be used, we argue that there is no disadvantage to switching to the filtering perspective, because the options are the same, but the benefit is that our hyperparameters make more sense. The option to project back to latent-weights also works the other way around and for those who already have a well tuned SGD optimizer could use it to make it easier to switch to our filtering perspective. The benefit of our interpretations is having fewer hyperparameters to set.

Its also true that our method cannot use common techniques based on the magnitude such as weight clipping or gradient clipping. Yet, we do not really think these techniques are necessary. We see such methods as heuristics to reduce the bit flipping ratio over time, which helps with convergence. However, in our setting, this can also be done using a good α decay schedule without reverting to such heuristics, making the optimization less complex.

We did not yet have the opportunity to test the filtering-based optimizer on more architectures and datasets. However, since our optimizer is equivalent to a specific setting of SGD, we would argue that architectures that have been trained with SGD will probably also work well with our optimizer. This is also a reason why we chose to use ReActNet-A, since it was trained using Adam in both in the original paper [2] and in [4]. The latter specifically argues that Adam works better for optimizing BNNs, but we suspect that the advantages of Adam are decreased because it might not work in the same way in the magnitude invariant setting, as we see a smaller difference in accuracy. Introducing this normalizing aspect into the filtering-based perspective is an interesting topic for future research.

One last point to touch upon is soundness. Even though the filtering perspective provides a better explanation to hyperparameters, it does not provide understanding on why optimizing BNNs with second-order low pass filters works as well as it does. Whereas stochastic gradient descent has extensive theoretical background, this does not exist for current mainstream BNN methods. Fully understanding BNN optimization is an interesting direction for future research and our hope is that this work takes a step in that direction.

Ethics Statement We believe that this research does not bring up major new potential ethical concerns. Our work makes training BNNs easier, which might increase their use in practice.

Reproducibility Statement Two important things for better reproducing our results rely on the GPUs and the dataloader. The reproduction of our ImageNet experiments is not trivial. First, as the teacher-student model is used in our ImageNet experiments, it will occupy a lot of GPU memory. We trained on 3 NVIDIA A40 GPUs, each A40 has 48 GB of GPU memory, with a batch size of 170 per GPU for as much as ten days. Second, for faster training on ImageNet, we used NVIDIA DALI dataloader to fetch the data into GPUs for the image pre-processing. This dataloader could effect training as it uses a slightly different image resizing algorithm than the standard PyTorch dataloader. To keep results consistent with other methods, we do the inference with the standard PyTorch dataloader.

REFERENCES

[1] H. Kim, J. Park, C. Lee, and J.-J. Kim, *Improving accuracy of binary neural networks using unbalanced activation distribution,* in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2021) pp. 7862–7871.

[2] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng, *Reactnet: Towards precise binary neural network with generalized activation functions,* in *European Conference on Computer Vision* (Springer, 2020) pp. 143–159.

[3] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, *Training binary neural networks with real-to-binary convolutions,* arXiv preprint arXiv:2003.11535 (2020).

[4] Z. Liu, Z. Shen, S. Li, K. Helwegen, D. Huang, and K.-T. Cheng, *How do adam and training strategies help bnns optimization?* ICML (2021).

[5] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder, *Latent weights do not exist: Rethinking binarized neural network optimization,* in *Advances in neural information processing systems* (2019) pp. 7533–7544.

[6] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *Xnor-net: Imagenet classification using binary convolutional neural networks,* in *European conference on computer vision* (Springer, 2016) pp. 525–542.

[7] A. Bulat and G. Tzimiropoulos, *Xnor-net++: Improved binary neural networks,* (2019), arXiv:1909.13863 [cs.CV] .

[8] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, *Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm,* in *ECCV* (2018).

[9] J. Lee, D. Kim, and B. Ham, *Network quantization with element-wise gradient scaling,* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021) pp. 6448–6457.

[10] Y. Zhang, Z. Zhang, and L. Lew, *Pokebnn: A binary pursuit of lightweight accuracy,* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022) pp. 12475–12485.

[11] Z. Xu, M. Lin, J. Liu, J. Chen, L. Shao, Y. Gao, Y. Tian, and R. Ji, *Recu: Reviving the dead weights in binary neural networks,* ICCV (2021).

[12] Z. Liu, Z. Shen, S. Li, K. Helwegen, D. Huang, and K.-T. Cheng, *How do adam and training strategies help bnns optimization?* in *International Conference on Machine Learning* (PMLR, 2021).

[13] Y. Li, S.-L. Pintea, and J. C. van Gemert, *Equal bits: Enforcing equally distributed binary network weights,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36 (2022) pp. 1491–1499.

[14] L. Hou, Q. Yao, and J. T. Kwok, *Loss-aware binarization of deep networks,* ICLR (2017).

[15] D. Kim, J. Lee, and B. Ham, *Distance-aware quantization,* in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021) pp. 5271–5280.

[16] F. De Putter and H. Corporaal, *How to train accurate bnns for embedded systems?* arXiv preprint arXiv:2206.12322 (2022).

[17] J. Hu, Z. Wu, V. Tan, Z. Lu, M. Zeng, and E. Wu, *Elastic-link for binarized neural networks,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36 (2022) pp. 942–950.

[18] W. Tang, G. Hua, and L. Wang, *How to train a compact binary neural network with high accuracy?* in *AAAI* (2017) pp. 2625–2631.

[19] X. Lin, C. Zhao, and W. Pan, *Towards accurate binary convolutional neural network,* in *NeurIPS* (2017).

[20] M. Lin, R. Ji, Z. Xu, B. Zhang, Y. Wang, Y. Wu, F. Huang, and C.-W. Lin, *Rotated binary neural network,* ECCV (2020).

[21] M. Courbariaux, Y. Bengio, and J.-P. David, *Binaryconnect: Training deep neural networks with binary weights during propagations,* in *NeurIPS* (2015).

[22] M. Alizadeh, J. Fernández-Marqués, N. D. Lane, and Y. Gal, *An empirical study of binary neural networks' optimisation,* in *International conference on learning representations* (2018).

[23] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, *Binarized neural networks,* Advances in neural information processing systems **29** (2016).

[24] Z. Cai, X. He, J. Sun, and N. Vasconcelos, *Deep learning with low precision by half-wave gaussian quantization,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017) pp. 5918–5926.

[25] H. Qin, R. Gong, X. Liu, M. Shen, Z. Wei, F. Yu, and J. Song, *Forward and backward information retention for accurate binary neural networks,* in *CVPR* (2020).

[26] T. Chen, Z. Zhang, X. Ouyang, Z. Liu, Z. Shen, and Z. Wang, *" bnn-bn=?": Training binary neural networks without batch normalization,* in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2021) pp. 4619–4629.

[27] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, *On the importance of initialization and momentum in deep learning,* in *International conference on machine learning* (2013).

[28] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization,* ICLR (2015).

[29] X. Yang, *Stochastic gradient variance reduction by solving a filtering problem,* arXiv preprint arXiv:2012.12418 (2020).

[30] C. D. Suarez-Ramirez, M. Gonzalez-Mendoza, L. Chang, G. Ochoa-Ruiz, and M. A. Duran-Vega, *A bop and beyond: a second order optimizer for binarized neural networks,* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021) pp. 1273–1281.

[31] J. W. Peters and M. Welling, *Probabilistic binary neural networks,* CoRR (2018).

[32] A. Shekhovtsov and V. Yanush, *Reintroducing straight-through estimators as principled methods for stochastic binary networks,* in *DAGM German Conference on Pattern Recognition* (Springer, 2021) pp. 111–126.

[33] Y. Bengio, N. Léonard, and A. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation,* Technical Report (2013).

[34] X. Meng, R. Bachmann, and M. E. Khan, *Training binary neural networks using the bayesian learning rule,* in *International conference on machine learning* (PMLR, 2020) pp. 6852–6861.

[35] C. Louizos, M. Reisser, T. Blankevoort, E. Gavves, and M. Welling, *Relaxed quantization for discretized neural networks,* arXiv preprint arXiv:1810.01875 (2018).

[36] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks,* in *AISTATS* (2010) pp. 249–256.

[37] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,* in *Proceedings of the IEEE international conference on computer vision* (2015) pp. 1026–1034.

[38] J. G. Proakis, *Digital signal processing: principles algorithms and applications* (Pearson Education India, 2001).

[39] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, *Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,* CoRR (2016).

[40] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, *Differentiable soft quantization: Bridging full-precision and low-bit neural networks,* in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019) pp. 4852–4861.

[41] Z. Wang, J. Lu, C. Tao, J. Zhou, and Q. Tian, *Learning channel-wise interactions for binary convolutional neural networks,* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019) pp. 568–577.

[42] H. Phan, Z. Liu, D. Huynh, M. Savvides, K.-T. Cheng, and Z. Shen, *Binarizing mobilenet via evolution-based searching,* in *CVPR* (2020).

[43] H. Phan, Y. He, M. Savvides, Z. Shen, *et al.*, *Mobinet: A mobile binary network for image classification,* in *WACV* (2020).

[44] J. Bethge, C. Bartz, H. Yang, Y. Chen, and C. Meinel, *Meliusnet: Can binary neural networks achieve mobilenet-level accuracy?* arXiv preprint arXiv:2001.05936 (2020).

[45] S. Becker, M. Ackermann, S. Lapuschkin, K.-R. Müller, and W. Samek, *Interpreting and explaining deep neural networks for classification of audio signals,* arXiv preprint arXiv:1807.03418 (2018).

[46] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, *BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization,* in *Advances in Neural Information Processing Systems 33* (2020).

6

CONCLUSION AND FUTURE WORK

6.1. CONCLUSION

The whole dissertation focuses on quantizing full-precision representation to a single bit representation for improving efficiency in terms of speed and storage.

- Chapter 2 and chapter 3 proposed two different hashing methods to convert the complex raw data such as images and videos into single bit binary code representation. The results showed that the proposed supervised hashing method in chapter 2, which used high quality human annotated labels as supervision, can achieve better accuracy performance than the unsupervised hashing method as in chapter 3. The fine-grained human annotations in supervised hashing is quite expensive. It remains future work to investigate semi-supervised hashing or self-supervised hashing.

- Chapter 4 and chapter 5 proposed different approaches for quantizing full-precision weights and activations of deep neural networks into single bit values to exploit fast bit-wise operators and meanwhile save storage cost. Chapter 5 proposed a new optimization method for BNNs which has fewer hyperparameters to tune comparing with the general used optimization approach as in chapter 4. The proposed technique by maximizing bit entropy in chapter 4 may lead to further performance improvement when applied on the new optimization approach proposed in chapter 5.

- Both chapter 3 and chapter 4 are based on entropy theory. Chapter 3 proposed a parameter-free bi-half layer to minimize the information loss when quantizing real-valued features into binary code representations. Chapter 4 proposed to impose a Bernoulli prior distribution on weights to learn binary weights. Both chapters shown information entropy theory can help the learning of binary codes.

- Optimization plays an important role in training DNNs. The magnitude-based training hyperparameters such as weight-decay, learning rate, and learning rate decay are well interpreted in real-valued weights based networks, which were adopted in chapter 2, chapter 3 and chapter 4. In chapter 5 we found the magnitude of binary weights is not meaningful thus we offered a new interpretation of these magnitude-based hyperparameters based on higher-order gradient filtering which allows for new optimization filters specifically designed for binary neural networks.

Limitations: We have explored the extreme case of quantizing full-precision data, such as images, to 1 bit to achieve memory savings and faster computation with little accuracy degradation. However, the restricted binary format data compression leads to a large loss of information, which may encounter problems such as in medical imaging analysis. In future work we propose to explore adaptive quantization [1] that dynamically adjusts the quantization levels to preserve more information.

6.2. FUTURE WORK

This thesis provides fruitful ground for future research, and some directions are explored below.

Optimizing Binary Neural Network by Explicit Weight Flip Ratios: Flipping the binary weight states is of core practical importance in training Binarized Neural Networks (BNNs). However, current optimization methods can only implicitly control the flips with a thresholding of the associated latent weights or gradient values, making it difficult to tune and even preventing important flipping options. As a next step, it will be interesting to develop a new optimizer to directly targeting at bit flipping for BNNs optimization. A hyper-parameter, flipping ratio, would explicitly control the binary weight flips. The hyper-parameter has an intuitive interpretations and typically require less tuning. optimization options.

Parameter-free Optimizer for Binarized Network: The existing optimization algorithms of binary neural networks (BNNs) highly rely on tuning the hyper-parameters such as learning rate, weight decay [2, 3] to achieve good accuracy results. Tuning the hyperparameter is difficult and inefficient for training. Binary networks are a discrete optimization problem where each binary weight only has two states, -1 or +1. This provides us an opportunity to design a new parameter-free optimizer based on discrete programming to search a good binary weight solution. Enumerating all possible combinations is infeasible in practice. The calculated partial derivatives with respect to binary weights could provide a good search direction. Going along this direction, we could constrain the solution space from 2^N to $N+1$, where N is the binary weights number. For the large network, the N could be millions. Searching over the solution space is infeasible. One possible approach is to use a line search [4] method with Wolf conditions to further reduce the search space. In particular, the Taylor's formula could be explored to estimate the loss values over the solution space and the estimated loss is used to measure when we stop flipping the binary weight. Another potential solution is to use Bayesian optimization to estimate the loss function with sampling limited points from the whole $N+1$ solutions.

Learnable Optimizer for Binarized Network: Traditional optimization methods of BNNs such as STE [5] and Bop [6] are designed by hand. Both optimization methods rely on a predefined or dynamic threshold to determine whether the binary weight flips or not, but the threshold is given based on human experiences. For example, the Bop [6] method needs to carefully tune the predefined threshold to improve the performance which is difficult to optimize. The more efficient approach is to propose a learnable optimizer to learn how to flip binary weights. This can be implemented by training a RNN [7] or a MLP [8] jointly with training binary network to learn how to flip. Specifically, the hand-designed flipping rule used in STE [5] or Bop [6] is replaced by a learned flipping rule, which we name learnable bit-flip optimizer parameterized via a small network. The learnable optimizer based on RNN or MLP is not used for inference, thus it will not affect the efficiency.

Accelerating Pruned Binarized Network: In chapter 4, we show that applying a pruning method on binary network, as in [9], could further compress the model and speed up inference. However, pruning binary weights may require storing a binary mask and the unstructured pruned binary network has an irregular computation pattern and is hard to parallelize [10, 11]. The work of Kwon S J et al.(2020) [12] attempted to study the weight representation scheme to support unstructured pruning method. They can represent the sparse weights of unstructured pruning by only 0.28 bits/weight for 1-bit quantization and 91% pruning rate with a fixed decoding rate and full memory bandwidth usage. Meanwhile, ESE [13] optimizes LSTM computation across algorithm, software and hardware stack that efficiently deals with the irregularity caused by unstructured pruning. It still remains an interesting investigation of co-designing software-hardware in terms of data flow scheduler and hardware accelerator to accelerate the pruned binary network.

Learning Binary Hash Codes by Forcing Bit Independent: We proposed an unsupervised hashing method in chapter 3 that could learn binary codes with maximum bit entropy. One limitation is that the independence between different bits is not considered. To reduce the information redundancy between different hash bits, we expect to benefit, as in previous methods [14, 15], by forcing bits to be independent with each other. The independence can be defined by the probability densities. We could formulate a term loss to force bit independence.

A Framework to Integrate Model and Data Compression: The efficiency could be further improved by exploiting the compressed DNN model to compress the high dimensional data. For the image retrieval task, a query image is first input to the trained model to learn binary codes, then the generated binary codes of query image are used to compute Hamming distance with binary codes of database where the nearest image is returned. Using a compressed model to generate binary codes could accelerate the retrieval process which is more efficient.

As a concluding remark, we have demonstrated that encoding full-precision complex representations into binary code values can significantly enhance the efficiency in terms of computation and storage. Throughout our thesis, we have effectively applied the proposed methodologies to various medium-sized models and databases. Yet, additional research is essential for large-scale models and databases.

REFERENCES

[1] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, *Adaptive quantization for deep neural network,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32 (2018).

[2] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, *Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm,* in *ECCV* (2018).

[3] Z. Liu, Z. Shen, S. Li, K. Helwegen, D. Huang, and K.-T. Cheng, *How do adam and training strategies help bnns optimization?* ICML (2021).

[4] J. J. Moré and D. J. Thuente, *Line search algorithms with guaranteed sufficient decrease,* ACM Transactions on Mathematical Software (TOMS) **20**, 286 (1994).

[5] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *Xnor-net: Imagenet classification using binary convolutional neural networks,* in *European conference on computer vision* (Springer, 2016) pp. 525–542.

[6] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder, *Latent weights do not exist: Rethinking binarized neural network optimization,* in *Advances in neural information processing systems* (2019) pp. 7533–7544.

[7] S. Hochreiter and J. Schmidhuber, *Long short-term memory,* Neural computation **9**, 1735 (1997).

[8] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators,* Neural networks **2**, 359 (1989).

[9] J. Diffenderfer and B. Kailkhura, *Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network,* ICLR (2021).

[10] T. Gale, E. Elsen, and S. Hooker, *The state of sparsity in deep neural networks,* arXiv preprint arXiv:1902.09574 (2019).

[11] A. Buluc and J. R. Gilbert, *Challenges and advances in parallel sparse matrix-matrix multiplication,* in *2008 37th International Conference on Parallel Processing* (IEEE, 2008) pp. 503–510.

[12] S. J. Kwon, D. Lee, B. Kim, P. Kapoor, B. Park, and G.-Y. Wei, *Structured compression by weight encryption for unstructured pruning and quantization,* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020) pp. 1909–1918.

[13] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, *et al.*, *Ese: Efficient speech recognition engine with sparse lstm on fpga,* in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2017) pp. 75–84.

[14] V. Erin Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, *Deep hashing for compact binary codes learning,* in *CVPR* (2015) pp. 2475–2483.

[15] Y. Weiss, A. Torralba, and R. Fergus, *Spectral hashing,* in *NIPS* (2008).

ACKNOWLEDGEMENTS

"One cup of dutch tomato soup, one sandwich, one roasted potato bowl with meatball and one banana", this is a simple lunch for a family of three at Aula, TUDelft after picking up kid from preschool! To be precise, this is a family with my wife, my daughter and me! I am very thankful to my wife, Yongni, for your love, accompany and support. We go together from China to Netherlands, from the beginning to the end of my Ph.D. journey, from two person to three person! May 23, 2020, the happiest day in my life, hmm, I actually cried when I saw my little cutie, Lezhi, slowly open her eyes which was full of curiosity about this world. Being a father also made me be in a spin. I learnt to change diaper, learnt to tell bedtime stories, learnt to become her "superhero", the most important thing I learnt is getting to understand my parents better. I put my mind to feel love from my family, fight hesitation because of responsibility, find interests from piece of life. This is how a father perusing his PhD degree peruses his happiness in the years' challenging and memorable PhD Journey.

"Simple is elegant!" told by my supervisor Jan van Gemert in a research project meeting which impressed me a lot along my whole PhD journey. I would like to express my sincere gratitude to Jan van Gemert. Thank you for your professional guidance and smart thinking way of doing research. Actually, I was a big fun of “complex" in my first year of PhD, where I kept adding more points to show more contributions in a single research project. After going through a lot of helpful and inspiring discussion between us, I understand simple is not trivial, instead, it first needs to dig very deep into the idea, and then summarize, revise, tune the idea repeatedly, to finally arrive at a perceived simplicity. I believe once I can arrive at such simplicity in digging into a whole research field, it must be a completely original scientific research. I have got a lot of valuable advises from you in both my research and life, many thanks, my Advisor!

I would like to give my full gratitude to my promoter, Marcel Reinders. Thank you for taking time to revise my dissertation carefully. During my PhD period, I observe your boundless enthusiasm to science and life which also gradually affects my attitude towards science and life. I deeply appreciate your continuous support and guidance over the years, and I thank you for your valuable advice.

I would also like to express my gratitude to all members of my thesis committee. Thank you for taking time for reading this thesis.

Many thanks to Silvia, a senior researcher whose assistance and valuable suggestions have been instrumental in my progress. Our collaborative efforts have been fruitful, and I have gained a wealth of knowledge from our association. I extend special thanks to Wenjie for being an excellent mentor when I arrived in the Netherlands. I cherish the memories of our small group of four with Yazhou, Xiangrong, you and me, often playing cards all night during the weekends, which brought warmth to my new Dutch life.

My thanks also go to my colleagues in computer vision lab: Amogh, Attila, Burak, Chengming, Giorgio, Hadi, Marian, Nergis, Niek, Nikolaas, Ombretta, Osman, Robert-

Jan, Seyran, Xiangwei, Xin, Xucong, Yancong, Yeshwanth, Ziqi.

I express my sincere gratitude to the other members of the PRB group for their assistance and friendship: Alexander, Arman, Bernd, Bob, Chengyao, Chirag, David, Ekin, Hayley, Jesse, Jin, Jing, Jose, Laura, Mahdi, Marco, Meng, Ramin, Rickard, Ruud, Saskia, Skander, Stephan, Stephanie, Taylan, Tom, Yanxia, Yuko.

Special thanks to my good friend Ruisheng Su. Thanks for your sharing and invitation for all interesting events covering career vision and life pursuit. Also many thanks to my other Chinese friends: Xiuxiu, Li Zhou, Guosheng, Yuan Chen.

This dissertation, as a gift, I would like to give it to my father, mother, father-in-law, mother-in-law and my elder brother. Thank you for your infinite love and support all the time in my life.

Last, I would like to convey my profound sense of missing my great grandma–my dad's grandma, who given me a happy childhood with her deep love. Due to illness, you have forgotten everyone but the only thing you remembered, "Beneath my bed, there are some cashes that I stored for buying snacks for my kid". I regret to not believe you at that time, while we did see them afterwards. I miss you a lot, my great grandma! May you rest in peace in heaven.

Yunqiang LI
Delft, Netherlands

CURRICULUM VITAE

Yunqiang Li was born at 12/08/1993 in a small village affiliated to Shangqiu, Henan Province of China. He was a second child in a peasant family. His parents have devoted most of their time to field works, growing and harvesting corn, wheat and apples. They always taught Yunqiang – What you plant now, you will harvest later– while Yunqiang was a little kid. What a strange brain kid has, it can quickly filter out parents' words! Yunqiang was still very naughty, catching fish in the river and skipping class occasionally along with his elder brother and friends. Naughty time also brought the most innocent happiness. Years passed, Yunqiang started to think about what he could plant for his future. During the next few years, he would often help his parents to do farm work in the fields, experiencing the hard but joy of harvest. He dreamed about obtaining a machine to do all these field kind of works itself. He was eager to gain more knowledge to change his life, and the fastest path is to go to college.

September 2010, Yunqiang began his first year as a Bachelor student in Air Force Engineering University, majored in Information and Communication Engineering. In 4 years' time Yunqiang completed his studies and gained basic knowledge to become a computer engineer. Between September 2014 and December 2016, he became a joint master student in Xi'an Jiaotong University and touched the research field of computer vision and artificial intelligence for the first time. Yunqiang was quite excited about his research topics. Afterwords, he moved to Delft University of Technology to further pursuing the doctoral education in artificial intelligence, working with Dr. Jan C. van Gemert. Yunqiang will continue to follow his dream to apply artificial intelligence to save the efforts of human like his parents in the future!

List of Publications

Thesis Research:

- **Yunqiang Li**, Wenjie Pei, Yufei Zha and Jan van Gemert. *Push for quantization: Deep fisher hashing*, Published in British Machine Vision Conference (BMVC), Oral, 2019, **Chapter 2**.
- **Yunqiang Li** and Jan van Gemert. *Deep Unsupervised Image Hashing by Maximizing Bit Entropy*, Published in Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2021, **Chapter 3**.
- **Yunqiang Li**, Silvia Laura Pintea and Jan van Gemert. *Equal Bits: Enforcing Equally Distributed Binary Network Weights*, Published in Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2022, **Chapter 4**.
- Joris Quist, **Yunqiang Li***, Jan van Gemert (* daily supervisor). *Understanding weight-magnitude hyperparameters in training binary networks*, Published in International Conference on Learning Representations (ICLR), 2023, **Chapter 5**.

Other Research:

- **Yunqiang Li**, Jan van Gemert, Torsten Hoefler, Bert Moons, Evangelos Eleftheriou, Bram-Ernst Verhoef. *Differentiable Transportation Pruning*, Published in International Conference on Computer Vision (ICCV), 2023.
- Xiangwei Shi*, **Yunqiang Li***, Xin Liu* and Jan van Gemert (* equal contribution), *WeightAlign:Normalizing Activations by Weight Alignment*, Published in International Conference on Pattern Recognition (ICPR), 2020.
- Xiangwei Shi, Seyran Khademi, **Yunqiang Li**, Jan van Gemert. *Zoom-CAM: Generating Fine-grained Pixel Annotations from Image Labels*, Published in International Conference on Pattern Recognition (ICPR), 2020.
- Yufei Zha, Tao Ku, **Yunqiang Li**, Peng Zhang. *Deep Position-Sensitive Tracking*, Published in IEEE Transactions on Multimedia (TMM), 2019.