xAILAB
EXPLAINABLE MACHINE LEARNING

Exploring and Evaluating Deep Hashing Methods within Vision Foundation Model Feature Spaces for Similarity Search and Privacy Preservation

Bachelor Thesis

Bachelor of Science in Information Systems

Peiyao Mao

September 4, 2024

Supervisor:

1st: Prof. Dr. Christian Ledig
2nd: Francesco Di Salvo, M.Sc.

Chair of Explainable Machine Learning
Faculty of Information Systems and Applied Computer Sciences
Otto-Friedrich-University Bamberg

Abstract

In the digital age, the burgeoning use of images across various applications, from social media to medical diagnostics, necessitates advancements in efficient data retrieval and robust privacy preservation techniques. This thesis investigates the efficacy of hashing methods applied to image embeddings derived from state-of-the-art Vision Transformer (ViT) models, focusing on both the semantic preservation between original image embeddings and hashed image embeddings and the strengthening of data privacy.

The study begins by generating high-dimensional image embeddings using a ViT-B/16 encoder, known for its effectiveness in capturing complex visual patterns. These embeddings serve as a baseline to assess the initial retrieval performance using the K-Nearest Neighbors (KNN) algorithm. Subsequent experiments involve the application of various deep supervised hashing techniques, including both pairwise and triplet methods, to transform these embeddings into compact, hashed representations. The performance of these hashed embeddings is again evaluated using KNN to draw comparisons with the baseline results, providing a comprehensive analysis of each method's impact on accuracy and efficiency.

A key aspect of this research is the emphasis on privacy preservation. By converting raw image data into hashed forms, this work explores how advanced hashing techniques can obscure original data features, thereby enhancing privacy without substantially compromising the utility for tasks such as medical image retrieval. This dual focus addresses the critical challenge of using sensitive image data in environments where privacy concerns are important.

Abstract

Im digitalen Zeitalter erfordert die zunehmende Verwendung von Bildern in verschiedenen Anwendungen, von sozialen Medien bis hin zur medizinischen Diagnostik, Fortschritte bei der effizienten Datenabfrage und robusten Techniken zum Schutz der Privatsphäre. Diese Arbeit untersucht die Wirksamkeit von Hashing-Methoden, die auf Bildeinbettungen angewandt werden, die von modernen Vision-Transformer-Modellen (ViT) abgeleitet sind, und konzentriert sich dabei sowohl auf die semantische Bewahrung zwischen ursprünglichen Bildeinbettungen und gehashten Bildeinbettungen als auch auf die Stärkung des Datenschutzes.

Die Studie beginnt mit der Erzeugung hochdimensionaler Bildeinbettungen unter Verwendung eines ViT-B/16-Encoders, der für seine Effektivität bei der Erfassung komplexer visueller Muster bekannt ist. Diese Einbettungen dienen als Ausgangsbasis für die Bewertung der anfänglichen Abrufleistung mit dem K-Nächste-Nachbarn-Algorithmus (KNN). Bei den anschließenden Experimenten werden verschiedene überwachte Hashing-Verfahren angewendet, darunter sowohl paarweise als auch triolische Methoden, um diese Einbettungen in kompakte, gehashte Darstellungen umzuwandeln. Die Leistung dieser gehashten Einbettungen wird wiederum mit KNN bewertet, um Vergleiche mit den Basisergebnissen zu ziehen und eine umfassende Analyse der Auswirkungen jeder Methode auf Genauigkeit und Effizienz zu erhalten.

Ein Schlüsselaspekt dieser Forschung ist die Betonung der Wahrung der Privatsphäre. Durch die Umwandlung von Rohbilddaten in gehashte Formen wird in dieser Arbeit untersucht, wie fortschrittliche Hash-Verfahren die ursprünglichen Datenmerkmale verschleiern können, wodurch die Privatsphäre verbessert wird, ohne dass der Nutzen für Aufgaben wie die Suche nach medizinischen Bildern wesentlich beeinträchtigt wird. Mit diesem doppelten Schwerpunkt wird die kritische Herausforderung der Verwendung sensibler Bilddaten in Umgebungen angegangen, in denen der Schutz der Privatsphäre eine wichtige Rolle spielt.

Acknowledgements

I would like to extend my heartfelt gratitude to Prof Dr.Christian Ledig for the opportunity to complete my bachelor thesis at his chair. My initial attendance at his seminar introduced me to the intriguing field of Explainable AI (xAI), sparking a profound interest that has significantly influenced my academic journey.

I am especially indebted to my supervisor, Francesco Di Salvo, whose invaluable advice, guidance, and insights have been instrumental in the successful completion of this thesis. His expertise and thoughtful mentorship have not only helped me navigate this complex topic but have also greatly enriched my learning experience.

Lastly, I must express my deepest appreciation to my parents, whose endless support and encouragement have been my anchor throughout this journey. Thank you for believing in me and standing by my side every step of the way.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AI	Artificial Intelligence
CNN	Convolutional neural network
DNN	Deep neural network
KNN	K-nearest neighbors algorithm
ViT	Vision Transformer
DSH	Deep supervised hashing
CNNH	Convolutional neural network hashing
DPSH	Deep pairwise supervised hashing
DHN	Deep hashing network
DSDH	Deep supervised discrete hashing
DNNH	Deep neural network Hashing
GAN	Generative Adversarial network
DSHGAN	Deep semantic hashing with GAN
TCL	Triplet center loss
xAI	Explainable AI
ML	Machine learning
DL	Deep learning

Notation

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathbf{A}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathbf{A}$
$\mathbf{A}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Linear Algebra Operations

$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^+$	Moore-Penrose pseudoinverse of $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$
$\det(\boldsymbol{A})$	Determinant of $\boldsymbol{A}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tensor containing derivatives of y with respect to $\mathbf{X}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$\mathrm{a} \perp \mathrm{b}$	The random variables a and b are independent
$\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$	They are conditionally independent given c
$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E}f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \Vert Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1 + \exp(-x)}$
$\zeta(x)$	Softplus, $\log(1 + \exp(x))$
$\Vert \boldsymbol{x} \Vert_p$	L^p norm of $\boldsymbol{x}$
$\Vert \boldsymbol{x} \Vert$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\mathrm{condition}}$	is 1 if the condition is true, 0 otherwise

Sometimes we use a function f whose argument is a scalar but apply it to a vector, matrix, or tensor: $f(\boldsymbol{x})$, $f(\boldsymbol{X})$, or $f(\mathbf{X})$. This denotes the application of f to the array element-wise. For example, if $\mathbf{C} = \sigma(\mathbf{X})$, then $C_{i,j,k} = \sigma(X_{i,j,k})$ for all valid values of i, j and k.

Datasets and Distributions

p_{data}	The data generating distribution
$\hat{p}_{\text{data}}$	The empirical distribution defined by the training set
$\mathbb{X}$	A set of training examples
$\boldsymbol{x}^{(i)}$	The i-th example (input) from a dataset
$y^{(i)}$ or $\boldsymbol{y}^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\boldsymbol{X}$	The $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$

1 Introduction

1.1 Context and Motivation

In recent years, the field of computer vision has experienced a remarkable surge in both research and practical applications, driven by advancements in deep learning techniques and the increasing availability of large-scale visual data. Among the applications of computer vision, one of the fundamental tasks is similarity search, where the goal is to retrieve images similar to a query from vast database. Whether it's for image retrieval in search engines, content-based recommendation systems, or object recognition in surveillance systems, efficient similarity search plays a significant roll in enhancing user experience and enabling various downstream applications.

Simultaneously, the importance of privacy preservation when dealing with visual data cannot be overstated. As healthcare systems increasingly rely on visual information for diagnosis and treatment monitoring, the need to protect patients' privacy has become more critical. With the proliferation of medical technologies and electronic health record systems, concerns regarding the protection of individuals' sensitive visual data have increased. Personal images, biometric data, and proprietary medical records must be shielded from unauthorized access, misuse, or exploitation to uphold patient confidentiality and trust in healthcare services.

Hashing methods emerge as a promising solution to address the challenges of similarity search and privacy preservation in computer vision applications. By mapping high-dimensional feature representations extracted from images into compact binary codes, hashing enables efficient similarity retrieval while significantly reducing storage and computational requirements (Redaoui et al. (2024)). For example, 80 million tiny images (32×32 pixels, of double type) cost around 600 Giga bytes, but can be compressed into 64 bit binary codes requiring only 600 Megabytes (Wang et al. (2015)). Moreover, hashing techniques offer inherent privacy benefits by obscuring the original data while still allowing for meaningful comparisons based on similarity metrics. Within the context of vision foundation models, such as convolutional neural networks (CNNs), hashing methods play a crucial roll in leveraging the learned feature representations for similarity search tasks. These models, trained on large-scale visual datasets, capture complex hierarchical patterns and semantic information, which can be utilized to generate discriminative and compact hash codes (Sarker (2021)). By exploring and evaluating various hashing methods within the feature spaces of vision foundation models, we endeavor to enhance the advancement of both theoretical understanding and practical implementations of similarity search while ensuring robust privacy preservation.

1.2 Related Work

In recent years, similarity search, also known as nearest neighbor search, has become a prominent area of research to effectively process the ever-increasing amount of data in real-world applications [2]. It involves a collection of objects that are characterized by a collection of relevant features and represented as points in a high-dimensional attribute space. Given queries in the form of points in this space, we are required to find the nearest object to the query (Gionis et al. (1999)). In the case that the reference database is very large or that the distance computation between the query items and the database item is costly, it is often computationally infeasible to find the exact nearest neighbor (Wang et al. (2014)). Thus, hashing has emerged effectively popular reducing the dimensionality and storage requirements, making retrieval processes faster and more scalable (Gong et al. (2012)). In general, a hashing model takes an input data point (images, document, etc.) and outputs a sequence of bits or hash code representing that data point. These hash codes, which is typically irreversible, are designed so that similar items in the original space are mapped to similar or identical hash codes in the hashed space. It means that the original data cannot easily be reconstructed from the hash code, thereby also preserving the confidentiality of original data.
The first step of computing binary codes usually involves finding an intermediate continuous embedding of the original data (Gong (2014)). In the scope of this thesis, we involve pre-trained models such as VGG (Simonyan and Zisserman (2014)), ResNet (He et al. (2016)), and transformers (Vaswani et al. (2017)), as they can provide a robust feature extraction mechanism , allowing systems to capture subtle nuances in images that might be missed when training from scratch. This transfer learning from large-scale pre-trained models to specific task such as similarity search can results in higher accuracy because the model uses sophisticated, pre-learned features as building blocks, improving the relevance of results by understanding and comparing deep features, which can also reduce the computational cost and time required for training (Han et al. (2021), Paaß and Giesselbach (2023)). Other prior approaches such as methods based on randomized embeddings, which tend to be too noisy for a smaller number of bits (Kulis and Grauman (2009), Raginsky and Lazebnik (2009), Rahimi and Recht (2007)). For example, a popular hash function for Locality Sensitive Hashing (LSH) that preserves dot-product similarity uses random projections drawn from a Gaussian distribution (Andoni and Indyk (2008)). Another choice would be use Principal Component Analysis (PCA) to reduce the dimensions and bring out strong patterns of a dataset (Gordo et al. (2013), Jegou et al. (2010)). Liu et al. (2012) has also developed a principled formulation for learning hash codes using kernels and category labels, which requires a limited amount of supervised information, i.e., similar and dissimilar data pairs, and a feasible training cost in achieving high quality hashing.

[2] See `https://pub.towardsai.net/deep-hashing-for-similarity-search-9273aac054db`

After the embedding, we can use hashing to generate binarized data. General classification of hashing algorithm can be broadly divided into two categories: data independent methods and data dependent method. If a hashing function is defined independently of the data to be processed without involving a training process from the data, we refer to such a hashing technique as data independent hashing (Chi and Zhu (2017)). A hashing function is often prespecified, although some of them may learn data distributions to improve hashing results such as locality-sensitive hashing (LSH). LSH involves designing different hashing functions with randomized projections or permutations, which has a very high probability to return the same bit for similar data items (Singh and Gupta (2022); Gionis et al. (1999)). It offers sub-linear time search by hashing highly similar examples together in a hash table. LSH functions that accommodate Hamming distance (Indyk and Motwani (1998)), inner products (Charikar (2002)), l_p norms (Datar et al. (2004)), and normalized partial matching (Grauman and Darrell (2007)) have been developed in prior work, which have shown effectiveness on various image search applications. However, often long hash codes are used in LSH-based randomized techniques because of which recall decreases. Building more than one hash tables can remove this problem, but it increases the storage requirements and the time of query in the meantime (Wang et al. (2015)). Additionally, as LSH does not take the properties of data into account, it shows insufficient performance in different real world applications.

To address the shortcomings of data independent methods, researchers try to get high-quality hashing codes by learning good hash functions. As two pioneering methods, i.e., spectral hashing (Weiss et al. (2008)) and semantic hashing (Salakhutdinov and Hinton (2009)) have been proposed, learning to hash has sparked considerable academic interest in both machine-learning and data mining. With the development of deep learning, obtaining hash codes through deep learning is becoming more and more because of the ability to learn powerful representation of very complex hash functions and achieve end-to-end hash codes, which is very useful in many applications. These data dependent methods exploit data distributions and information on class labels from the dataset so that the nearest neighbor of any pattern in the hash coding space is as close as possible to its neighbors in the original space (Singh and Gupta (2022)), which can be broadly divided into two main sub categories: deep supervised hashing and deep unsupervised hashing. In deep supervised hashing, labeled information about similarity between pairs of images is leveraged to generate hash codes while learning model parameters to design advanced hash function (Zhu et al. (2016)). The design of the deep supervised hashing method mainly includes two parts: the design of network structure and the design of loss function (Luo et al. (2023)). For small datasets like MNIST (LeCun et al. (1998)) and CIFAR-10 (Krizhevsky et al. (2009)), shallow architecture such as CNN-F (Chatfield et al. (2014)) and AlexNet (Krizhevsky et al. (2017)) are widely used. While for complex datasets such as NUS-WIDE (Chua et al. (2009)), deeper architecture such as ResNet50 (He et al. (2016)) can be applied. The loss functions are designed with the objective of maintaining similarity structures. These methods primarily focus on minimizing the discrepancies between similarity structures in the original and

Hamming spaces (Cao et al. (2018)). In supervised scenarios, researchers typically derive similarities in the original space by utilizing label information. Consequently, determining how to obtain similarities from learned hash codes is crucial for different algorithms. The deep supervised hashing algorithms can be further classified based on their approaches to measuring the similarities of learned hash codes into four categories: pairwise methods, ranking-based methods, point-wise methods and quantization (Luo et al. (2023)). Another area of research in this field is deep unsupervised hashing, which does not require any label information. In unsupervised learning, the semantic information is typically extracted from relationships in the original spaces. Depending on the approach to learning this semantic information, deep unsupervised hashing algorithms can be categorized into three types: pseudo-label-based methods, similarity reconstruction-based methods and prediction-free self-supervised learning methods (Luo et al. (2023)). These methods harness the inherent structure of the data to generate meaningful hash codes without relying on external labels, thereby facilitating the efficient retrieval and categorization of large-scale data sets in various applications.

At retrieval time, when a new query image is given, it is necessary to calculate the distances from that query to every image in the database. Most methods achieve this by directly computing the Hamming distance, which counting the number of positions at which the corresponding bits are different. It is highly efficient for large-scale image retrieval because it requires simple bitwise operations that are computationally inexpensive. Another computation would be asymmetric distance, which efficiently deals with the problem when the query item might not undergo the same transformation or might be partially transformed, if database items are stored in a reduced dimensional space to speed up retrieval operations for instance (Gordo et al. (2013)). One typical approach in asymmetric distance calculations is to use different functions or metrics for the database and the query sides. For example, the query might be processed using a more complex function due to fewer constraints on query processing time compared to the need for extremely fast processing on the database side.

1.3 Contribution

- The contribution of the experiment involves the transformation of image embeddings to hashed image embeddings using various deep supervised hashing methods, making sure the semantic similarities are preserved and data privacy is enhanced. By comparing various deep supervised hashing methods, including pairwise and triplet methods, the experiment try to understand and evaluate how different approaches perform under the same conditions. It provides insights into which methods are most effective in retaining important data characteristics after hashing. The systematic evaluation of hashing techniques across different metrics like accuracy and computational efficiency helps in benchmarking current methods.

- We innovatively apply Triplet Center Loss (TCL) in the domain of deep hashing, aiming to achieve both high performance and computational efficiency.

2 Theoretical Foundations

2.1 Deep Neural Networks

Deep neural networks (DNNs) (Samek et al. (2016)) have emerged as a cornerstone in the field of machine learning, particularly in tasks requiring complex pattern recognition and data representation. Originating from the idea of artificial neural networks, DNNs consist of multiple layers of neurons, each designed to process aspects of the input data differently, yet collectively contributing to the final input (LeCun et al. (2015)). This multi-layer architecture allows DNNs to learn hierarchical representations, making them exceptionally good at managing complex and voluminous datasets typical in various computational tasks. The transformative impact of DNNs in computer vision began with the introduction of architectures like AlexNet (Krizhevsky et al. (2017)), which in 2012 demonstrated a profound leap in performance on the ImageNet challenge. AlexNet's success was attributed to its deep structure (five convolutional layers followed by three fully connected layers) and the use of rectified linear units, which helped the network train faster and more effectively. Subsequently, architectures such as VGGNet (Simonyan and Zisserman (2014)) and ResNet (He et al. (2016)) further advanced the field. VGGNet is noted for its simplicity and depth, utilizing small receptive fields in convolutional layers to build deeper networks. ResNet, introduced by Microsoft, brought the novel concept of residual learning, tackling the problem of vanishing/exploding gradients in building very deeper networks by using shortcut connections to jump over some layers.

In the realm of hashing within vision foundation models, these deep architectures have been foundational. Their ability to encode input data into compact binary hash codes through end-to-end learning revolutionizes similarity search and privacy preservation tasks (Wang et al. (2021)). This is particularly evident with the introduction of Vision Transformer (ViT) (Dosovitskiy et al. (2020)), which deviate from traditional convolutional approaches by applying self-attention mechanisms to process data. This shift has allowed for even more nuanced understanding and representation of image data, enhancing both the efficiency and accuracy of generating hash codes. For deep supervised hashing methods, the hashing network is usually modified from these aforementioned standard networks by replacing the classification head with a fully-connected layer containing L units for hash code learning (Luo et al. (2023)). To be more precise, after feature extraction, the data is flattened (if necessary) and passed through one or more dense layers which consolidate the features into a format suitable for the final hashing layer, which is typically a fully connected layer with a specific number of neurons corresponding to the designed length of the hash code. For instance, if a generation of 64-bit hash code is wanted,

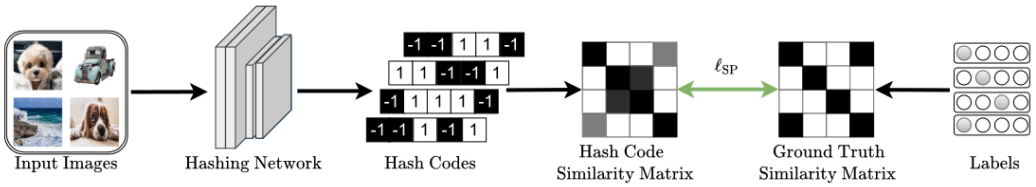

Figure 1: Basic Framework of Deep Supervised Hashing with Pairwise Similarity Measurement.The hash codes are produced by a hashing network. Afterwards, the pairwise similarity information of hash codes and ground truth is matched and thus get similarity preserving loss. (Luo et al. (2023))

this layer would have 64 neurons. The outputs from the hashing layer are then passed through a binary activation function, such as a sigmoid function followed by a thresholding step to convert each output to 0 or 1, thereby generating a binary hash code. The final output is a compact binary hash code representing the input data, ready for use in similarity searches or other applications. Figure 1 shows a representative framework of deep supervised hashing.

The architectural design of a hashing network is important in deep supervised hashing, significantly influencing the accuracy of the search results and the inference speed (Mehta and Sahni (2004)). The architecture's complexity and depth play dual roles: while a deeper network can increase the accuracy of hashing, it usually adds to the computational burden, thus slowing down the inference process. This trade-off highlights the importance of selecting an architecture that balances both accuracy and efficiency (Luo et al. (2023)). Moreover, if the network architecture is overly simplified—reverting to basic multilayer perceptrons (MLPs) or simple linear projections—the advantages of deep learning are negated, reducing the approach to traditional hashing methods. Such basic approaches do not leverage the rich, hierarchical feature representations that more sophisticated deep learning models offer, which are essential for effective hashing in complex, high-dimensional data spaces. It is also crucial to align the network architecture with the specific characteristics and complexity of the dataset. A more complex dataset, featuring high variability in data patterns, may need a deeper and more complex network to capture the nuanced features essential for accurate hashing. Conversely, for simpler datasets, a less complex model might suffice, optimizing both performance and computational efficiency (Luo et al. (2023)).

2.2 K-Nearest Neighbors Algorithm(KNN)

K-Nearest Neighbors (KNN) serves as a fundamental metric in the evaluation of hash codes generated by deep neural networks (Xu et al. (2018)), particularly within the domain of similarity search. As a non-parametric, instance-based learning algo-

rithm, KNN plays a crucial role in determining how effectively the embedded feature spaces preserves neighborhood relationships (Mitchell (1997)). This preservation is essential for maintaining data privacy and improving the performance of retrieval systems. By applying KNN, we can directly assess how closely the hash codes replicate the spatial relationships found in the original high-dimensional feature space. The basic concept of KNN is to identify the k most similar data points to a new data point, referred to as the query data point. It predicts the label of the query point by utilizing the labels of these k nearest neighbors. Similarity between two data points in a vector space is typically determined using a distance metric, such as Euclidean, Cosine and Manhattan distance. Euclidean distance is the most common choice:

$$EuclideanDistance = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

where x_i and y_i are components of vectors x and y respectively.

The k data points closest to the query point are called the k-nearest neighbors (Doerrich et al. (2024)). The choice of K is crucial as it determines the frontier of classification and how granular the classification is. A smaller K can make the noise have a higher influence on the result, while a larger K makes boundaries between classes less distinct. Research has also shown that a small number of neighbors are most flexible fit which will have low bias but the high variance and a large number of neighbors will have a smoother decision boundary which means lower variance but higher bias (Navlani (2018)).

A majority vote among these neighbors is then used to predict the label of the query point [1]. Optionally, the contributions of neighbors can also be weighted, which can be done by the inverse of their distance, giving nearer neighbors greater influence on the classification:

$$y_{pred} = argmax_j(\sum_{u=1}^{k} \omega_i \times \mathbf{1}(y_i = j))$$

where $\omega_i = \frac{1}{d(x,x_i)^\alpha}$. $d(x, x_i)$ is the distance from the test point x to each neighbor x_i, and $\mathbf{1}$ is the indicator function that is 1 if $y_i = j$ and 0 otherwise (Mitchell (1997)).

The process is illustrated in Figure 2.

KNN on Hashed Embeddings To apply the KNN algorithm to hashed embeddings, which are typically binary codes, we may need to use a distance metric that is suitable for comparing binary data, such as Hamming distance, instead of the Euclidean distance commonly used with continuous data.

[1] See `https://prof-frenzel.medium.com/dear-friends-854ba66361d7`

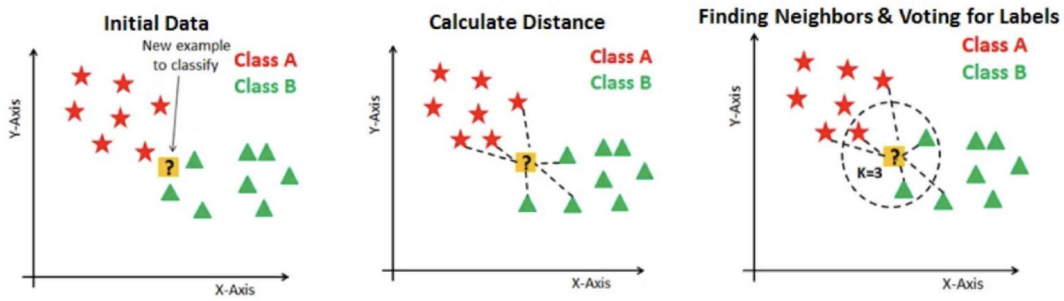

Figure 2: Visualization of KNN. (Navlani (2018))

Hamming distance The Hamming distance is a metric used to measure the difference between two strings of equal length (Federici et al. (2021)). It quantifies the number of positions at which the corresponding symbols are different, or, in simpler terms, it counts the number of substitutions required to change one string into the other (Waggener and Waggener (1995)). The concept was introduced by Richard Hamming, an American mathematician and computer scientist, in 1950. Hamming initially developed this distance metric to detect and correct errors in data transmitted over noisy communication channels. Today, it is widely used in various fields such as information theory, particularly in scenarios involving error detection and correction algorithms (Robinson (2003)). Given two strings, s and t, of equal length, the hamming distance between them, denoted as $d_H(s,t)$, is calculated as the number of positions i at which the corresponding symbols s_i and t_i differ:

$$d_H(s,t) = \sum_{i=1}^{n} [s_i \neq t_i]$$

where $[s_i \neq t_i]$ is an indicator function that is 1 if $s_i \neq t_i$ and 0 otherwise, and n is the length of the strings. It provides a quick and efficient way to measure similarity between encoded instances, making it particularly useful for applications involving binary hash codes, where it helps in assessing the closeness of data points in a feature space.

Here is the pseudocode for applying KNN to hashed embeddings, using the hamming distance:

Algorithm 1 KNN on Hashed Image Embeddings

```
procedure KNN_HASHED(X_train_hashed, y_train, X_q_hashed, k)
    for each x_q_hashed in X_q_hashed do
        Initialize an empty list distances
        for each x_train_hashed in X_train_hashed do
            distances.append(calculate_hamming_distance(x_q_hashed,
x_train_hashed))
        end for
        indices ← argsort(distances)[:k]
        k_nearest_labels ← select(y_train, indices)
        y_pred.append(mode(k_nearest_labels))
    end for
    return y_pred
end procedure
```

1. Calculate Hamming Distance. For each new data point's hashed embedding, calculate the Hamming distance to each training point's hashed embedding.

2. Sort and Select Neighbors. After calculating the distances, sort them and select the indices of the smallest distances to identify the k nearest neighbors.

3. Determine the Output. For each query data point, the output label is determined by the mode of the labels associated with the k nearest training instances.

2.3 Deep Hashing Algorithms

Learning to Hash The objective of learning-based hashing is to design hash functions that can convert high-dimensional data into compact binary codes while preserving the intrinsic similarities among the data points (Wang et al. (2017)). Let's denote the dataset $D = x_1, ..., x_n$ where each $x_i \in \mathbf{R}^d$ represents a high-dimensional input vector. The goal is to learn a hash function $h : \mathbf{R}^d \longrightarrow 0, 1^k$ that maps each input vector to a k-dimensional binary vector for the convenience of the nearest neighbor search (Mehta and Sahni (2004)). In the context of this article, we focus on deep hashing methods, which is an extension of learning to hash paradigm that incorporates deep learning techniques. In deep hashing, the hash functions are learned using deep neural networks. The integration of deep learning allows the hash function to not only learn from that data but also to perform feature extraction and transformation in an end-to-end manner. This means that deep hashing networks are capable of processing raw input data (like images or text), extracting relevant features using layers of the network, and then transforming these features into compact binary codes.

2.3.1 Overview

When designing deep supervised hashing, we mainly focuses on these key processes to ensure the system is efficient, effective, and capable of handling large-scale datasets (Luo et al. (2023)):

1. Choosing the deep neural network architecture.
2. Designing the loss function for preserving the similarity structure.
3. Optimizing the deep neural network with the discretization problem.
4. Additional techniques to enhance performance.

As discussed above, the choice of network architecture should be capable of extracting robust and discriminative features from the data that are suitable for generating binary hash codes. For image data, CNNs are typically preferred because of their proven ability in extracting hierarchical visual features from images, while Recurrent Neural Networks (RNNs) or Transformers may be more appropriate for text data due to their effectiveness in handling sequential data. Additionally, auto-encoders (Kramer (1991)) can be used for unsupervised feature extraction followed by a supervised fine-tuning for hashing.

Similarity Measurement (Luo et al. (2023)) Firstly, A clarifies the formal notations and symbols. $X = \{\mathbf{x}_i\}_{i=1}^N$ is denoted as the training set. The outputs of the hashing network are represented as $H = \{\mathbf{h}_i\}_{i=1}^N$, where $h_i = \Psi(x_i)$. The resulting binary codes are expressed as $B = \{\mathbf{b}_i\}_{i=1}^N$. We define the similarity between pairs of items $(\mathbf{x}_i, \mathbf{x}_j)$ in the input space and Hamming space as s_{ij}^o and s_{ij}^h respectively. In the input space, similarity is the ground truth, primarily based on sample distance d_{ij}^o and semantic labels. The sample distance, such as the Euclidean distance $||\mathbf{x}_i - \mathbf{x}_j||^2$, can be computed using functions like the Gaussian function, $exp(-\frac{(d_{ij}^o)^2}{2\sigma^2})$, or the characteristic function $I_{d_{ij}^o < \tau}$, where τ is a predetermined threshold. Cosine similarity is another popular metric. Semantic similarity, which is more relevant in deep supervised hashing, assigns a value of 1 if two examples share a common label and 0 otherwise.
In the Hamming space, the pairwise distance d_{ij}^h is naturally the Hamming distance. For binary codes valued at 1 and 0,

$$d_{ij}^h = ||\mathbf{b}_i - \mathbf{b}_j||_1$$

varies from 0 to L, the similarity is expressed as:

$$s_{ij}^h = \frac{L - d_{ij}^h}{L}$$

If two items have identical hash codes, their Hamming distance is 0, making $s_{ij}^h = \frac{L-0}{L} = 1$, indicating maximum similarity. Conversely, if all bits are different (worst case), $d_{ij}^h = L$ and $s_{ij}^h = \frac{L-L}{L} = 0$, indicating no similarity.

If the codes are valued at 1 and -1,

$$d_{ij}^h = \frac{1}{2}(L - \mathbf{b}_i^T \mathbf{b}_j)$$

The inner product $\mathbf{b}_i^T \cdot \mathbf{b}_j$ sums the products of corresponding bits. When all corresponding bits in the vectors match perfectly, the product will be high (maximum L); otherwise $-L$ when the bits are completely opposite. $L - \mathbf{b}_i^T \mathbf{b}_j$ calculates the "difference" score, which is then halved by $\frac{1}{2}$ to convert any pair of different bits into a single count of dissimilarity, fitting the typical range of a Hamming distance for binary codes valued between 0 and 1. This adjustment is necessary because each pair of mismatched bits (-1 in the product) represents two bits different, hence dividing by 2 aligns it with the traditional Hamming count where each different bit is counted once.

The similarity is then calculated using the inner product:

$$s_{ij}^h = \frac{\mathbf{b}_i^T \cdot \mathbf{b}_j + L}{2L}$$

The formula leverages the inner product of the codes. By shifting and scaling, it converts the inner product into a similarity measure. If all bits match perfectly (all pairs of corresponding bits are either both 1 or both -1), the sum $\mathbf{b}_i \cdot \mathbf{b}_j$ equals L, and the similarity $s_{ij}^h = \frac{L+L}{2L} = 1$. If all bits are opposite, the sum equals $-L$ and similarity is 0.

This framework can be extended to a weighted scenario, where each bit is associated with a weight λ_l, and for codes valued at 1 and -1, the similarity becomes

$$s_{ij}^h = \frac{\mathbf{b}_i^T \Lambda \mathbf{b}_j + tr(\Lambda)}{2tr(\Lambda)}$$

where $\Lambda = diag(\lambda_1, ..., \lambda_l)$ is diagonal and $tr(\cdot)$ denotes matrix trace. In the inner product $\mathbf{b}_i^T \Lambda \mathbf{b}_j$, each bit comparison $b_{il} \cdot b_{jk}$ (for each bit position l) is scaled by the weight λ_l. The normalization is then conducted by first adding $tr(\Lambda)$, which shifts the weighted sum to ensure that the resulting similarity score is non-negative. Then the step dividing by $2tr(\Lambda)$ normalizes the similarity score to be within range $[0, 1]$. Normalization is crucial for standardizing the output of the similarity measure, making it easier to interpret and compare across different pairs of items and different settings.

This weighted similarity measure is particularly useful in scenarios where not all the bits in a hash code have equal significance. For example, in image retrieval, certain features extracted and encoded into the hash might be more crucial for identifying similarities due to their higher discriminative power. By assigning higher weights to these bits, the similarity measure can more effectively reflect the true likeness between images.

Loss Function After defining the similarity measurement, we continue with the loss function in deep supervised hashing. Loss functions are central to the performance of deep supervised hashing. Their primary role is to ensure that the hash codes generated by the model accurately preserve the similarity structure of the original data space (Athitsos et al. (2007)). This means that the algorithm should minimize the discrepancy between how similar (or dissimilar) items are in the original input space and their representations in the Hamming space (where hash codes reside). The typical loss functions are pairwise loss and ranking-based loss. The pairwise loss is often used to ensure that pairs of similar items result in similar hash codes (small Hamming distance) and vice versa. The ranking-based loss is used to maintain the relative order of distances in the hash code space, ensuring that an anchor item is closer to positive items than to negatives.

Besides of maintaining similarity structures, label information can also be integrated in multiple ways. The first way is to regress hash codes with labels (Liu et al. (2019)). Here, the label is encoded into one-hot format matrix and regression loss, i.e.,

$$||Y - WH||_F$$

are added into the loss function, where Y is the matrix of label vectors, W is a transformation matrix, and H are the hash codes. The second way is adding a classification layer after the hashing network involves using a supervised loss function, such as cross-entropy, to make the hash codes discriminative with respect to the labels (Lin et al. (2015)).

Quantization loss plays a crucial role in controlling the quality of hash codes in deep hashing models, particularly when transforming the continuous outputs of neural networks into discrete, binary hash codes. It minimizes the differences between the continuous outputs of the network and their nearest binary values, helping to reduce the gap between the ideal binary representation and the actual outputs of the hashing network. This is crucial for the practical deployment of hashing systems where binary codes are required for storage and retrieval efficiency. Another important term is bit balance loss. It addresses the distribution of hash bits across the dataset, penalizing scenarios where certain bits are always 1 or always -1, which can lead to uninformative bits in the hash codes. Some regularization losses, such as L2 regularization, can also be added to prevent overfitting.

2.3.2 Pairwise Loss Functions

The core principle of pairwise loss is to bring the hash codes of similar class patterns closer together while pushing those of different classes further apart. Figure 3 depicts this fundamental concept of the pairwise approach. In this context, the pattern of interest is termed the "anchor." A pattern belonging to the same class as the anchor is referred to as "positive," whereas a pattern from a different class is labeled "negative" (Singh and Gupta (2022)). The loss function processes pairs formatted as $\langle anchor, positive \rangle$ to decrease the distance between their hash codes in the hash

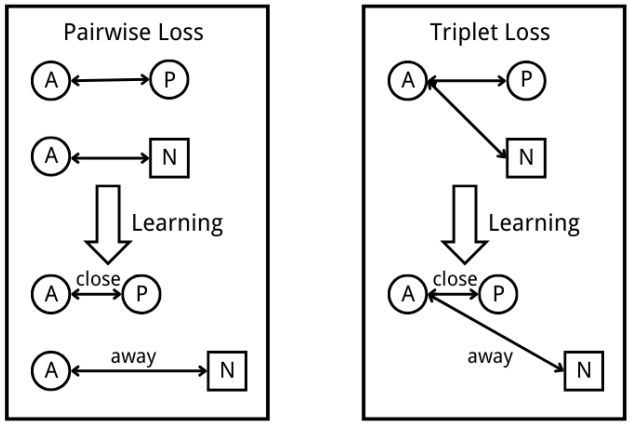

Figure 3: The basic idea of pairwise loss and triplet loss (Singh and Gupta (2022)). A: Anchor; P: Positive examples; N: Negative examples. Pairwise loss considers the distance between $\langle A, P\rangle$ and $\langle A, N\rangle$ separately, while triplet loss treat $\langle A, P, N\rangle$ as a whole.

space, or as $\langle anchor, negative\rangle$ to increase the distance, thereby segregating them distinctly within the hash code space (Singh and Gupta (2022)).

$$L = \sum_{(i,j)\in P} (y_{ij}||h_i - h_j||^2 + (1 - y_{ij})max(0, m - ||h_i - h_j||^2))$$

where h_i and h_j are the hash codes for the anchor and either a positive or negative example, respectively. y_{ij} is a binary label indicating if the pair is similar ($y_{ij} = 1$ for anchor-positive pairs) or dissimilar ($y_{ij} = 0$ for anchor-negative pairs). m is a margin that defines how far apart the dissimilar items' hash codes should be - a hyperparameter that can be fine-tuned.

For similar pairs ($\langle anchor, positive\rangle$), the goal is to minimize the distance between their hash codes ($||h_i - h_j||^2$). This encourages the model to map similar patterns (same class) closer in the hash code space (Jiao et al. (2023)), enhancing retrieval performance by making similar items more accessible via smaller Hamming distances. For dissimilar pairs ($\langle anchor, negative\rangle$), the model is trained to ensure that the distance between their hash codes is at least m. This is accomplished via the term $max(0, m - ||h_i - h_j||^2)$, which penalizes the model if the distance between dissimilar pairs is less than m. This helps in mapping different class patterns away from each other, reducing false positives in retrieval tasks.

Pairwise loss function typically falls into two categories: difference-based minimization and likelihood minimization (Luo et al. (2023)). Each approach targets a slightly different aspect of learning how to encode similarities and dissimilarities between pairs of data points.

Difference-Based Minimization Difference-based minimization is a critical approach in deep hashing, aimed at reducing the discrepancies between the similarities of data points in the original input space and the encoded Hamming space:

$$min \sum_{(i,j)\in P} (s^o_{i,j} - s^h_{i,j})^2$$

However, a direct optimization of binary hash codes poses significant challenges due to the non-differentiable nature of binary operations. Do et al. (2016) replaces actual binary codes with continuous outputs from neural networks before applying a hard thresholding sign function.

$$s^h_{i,j} = \frac{h_i^T h_j}{L}$$

where h_i and h_j are continuous vectors from the neural network. Backpropagation can then be used effectively, as the gradients can be calculated for the optimization process. Another asymmetric similarity method involved in Wu et al. (2019) calculates similarities using one binary and one continuous representation of data.

$$s^h_{i,j} = \frac{b_i^T h_j}{L}$$

This approach helps mitigate the harsh effects of quantization (the process of converting continuous output to binary) by reducing the error that typically arises when a continuous value is forced into a binary state, providing a buffer against the loss of information that occurs during binarization.

By transforming the difference losses into a product form and minimizing a product-based loss that ties the original space similarities to Hamming distances, the model directly encourages a smaller Hamming distance for items are similar in the original space (Erin Liong et al. (2015)):

$$min \sum_{(i,j)\in P} s^o_{i,j} d^h_{i,j}$$

emphasizing that higher similarities in the original space should correspond to smaller distances in the Hamming space. Incorporating a margin within these losses can provides a buffer that enhances the model's ability to differentiate between different classes of data more distinctly, thus further facilitating the relaxation and convergence of the optimization process.

Likelihood Loss Minimization Likelihood loss minimization in deep hashing involves modeling the similarity between items in a dataset probabilistically. This approach not only considers how closely the hash codes resemble each other but also integrates prior knowledge about the distribution of hash codes and the expected similarities based on data characteristics. This probabilistic framework is essential

for understanding and optimizing the semantic relationships captured by the binary hash codes. The process starts by considering a similarity matrix S and a matrix of binary codes B. The matrix

$$S = s^o_{i,j}{}_{(i,j)\in P}$$

represents known similarities between items. The relationship between the observed similarities S and the binary hash codes B is captured through a posterior estimation (Luo et al. (2023)):

$$p(B|S) \propto p(S|B)p(B) = \prod_{(i,j)\in P} p(s^o_{i,j}|B)p(B)$$

where $p(B)$ represents a prior distribution over the hash codes, and $p(S|B)$ is the likelihood of observing the similarity matrix given the hash codes. The conditional probability $p(s^o_{i,j}|B)$ can be expressed in terms of the hash code derived similarities $s^h_{i,j}$:

$$p(s^o_{i,j}|B) = p(s^o_{i,j}|s^h_{i,j}) = \begin{cases} \sigma(s^h_{i,j}), & \text{if } s^o_{i,j} = 1 \\ 1 - \sigma(s^h_{i,j}), & \text{if } s^o_{i,j} = 0 \end{cases}$$

where $\sigma(x) = \frac{1}{1+e^x}$ is the sigmoid function, transforming the hash code derived similarity into a probability. If the similarities in the original space are larger, the probabilistic model expects the similarities in the Hamming space also to be larger.

By transforming this into a log-likelihood gives:

$$logp(S|B) = \sum_{(i,j)\in P} logp(s^o_{i,j}|s^h_{i,j})$$

Substituting the conditional probabilities and simplifying, we get:

$$logp(S|B) = \sum_{(i,j)\in P} [s^o_{i,j} log\sigma(s^h_{i,j}) + (1 - s^o_{i,j}) log(1 - \sigma(s^h_{i,j}))]$$

To minimize the discrepancy between the predicted probabilities and the actual observed similarities, we take the negative of this sum:

$$L_{NLL} = -logp(S|B) = -\sum_{(i,j)\in P} log(1 + e^{s^h_{i,j}}) - s^o_{i,j} s^h_{i,j}$$

Similarly, the hashing network usually cannot directly obtain the hash codes. Hence, these codes B will replaced by the network outputs H to generate $s^h_{i,j}$ (Luo et al. (2023)).

However, traditional sigmoid function are sometimes insufficient for modeling complex data distributions. Alternatives like priority weighting (Cao et al. (2018)),

Cauchy distribution (Cao et al. (2017a)) and imbalance learning (Cao et al. (2017b)) are explored to better model the probabilities and improve the robustness of the hashing. Overall, likelihood loss minimization provides a sophisticated framework for enhancing the quality of hash codes by probabilistically modeling the similarities between items. This approach ensures that hash codes are not only computationally efficient but also semantically meaningful, making them highly effective for tasks involving similarity search and retrieval in massive datasets.

2.3.3 Triplet Loss Functions

Triplet loss is a powerful loss function used extensively in deep hashing where the goal is to maintain the natural order and relationships observed in the original data space within the constraints of a binary Hamming space (De Giacomo et al. (2020)). It leverages the relative similarity between triplets of items to train hashing models. A triplet typically consists of an anchor (x_i), a positive item (x_j), and a negative item (x_k), where the anchor is more similar to the positives than to the negatives ($s^o_{i,j} > s^o_{i,k}$) (Duan et al. (2022)). The triplet loss can be expressed as:

$$L_{Triplet}(h_i, h_j, h_k) = max(0, d^h_{i,j} - d^h_{i,k} + m)$$

m is a margin parameter that enforces a buffer, ensuring the positives are not just closer to the anchor than the negatives, but significantly closer by at least the margin m.

List-methods go beyond the triplet comparisons and consider the rankings of entire sets of items within the dataset, which aims to optimize global ranking metrics, making them more comprehensive but also computationally intensive. Typical ranking metrics can include average precision (AP) and normalized discounted cumulative gain (He et al. (2018a)). These are sophisticated metrics used in information retrieval that consider the order of all retrieved items, rewarding hits at higher ranks more than those at lower ranks. Mutual information is also utilized in Cakir et al. (2019) to optimize the hash network by measuring and maximizing the dependency between the input and the output hashes.

2.3.4 Pointwise Methods

Pointwise methods in deep supervised hashing provide a distinct approach from the pairwise and triplet methods by focusing directly on individual data points and their labels, rather than on relationships or similarities between pairs or groups of items. This methodology is particularly useful for scenarios where precise label information is available, allowing the hashing system to learn hash codes that directly reflect the class or category of each item. The primary goal of pointwise methods is to ensure that the hash codes generated for each item accurately reflect its label or category. The method is commonly applied in settings where the labels provide significant standalone information about the data points, such as in supervised learning tasks with distinct class labels.

In a typical pointwise approach, each item x_i in the dataset is associated with a label y_i. The objective is to learn a function $h(x_i)$ that maps x_i to a binary hash code b_i such that the hash code is indicative of y_i. One straightforward method to implement loss function that directly compares the predicted hash code with the label is to treat each bit of the hash code as a binary classification problem (Yuan et al. (2020)):

$$L(h_i.y_i) = \sum_{k=1}^{K} l(g_k(h_i), y_{i,k})$$

where K is the number of bits in the hash code, g_k is a function that maps the k-th component of the hash code to a predicted label bit. l is a suitable loss function, such as binary cross-entropy defined as:

$$l(p, q) = -[qlog(p) + (1 - q)log(1 - p)]$$

which takes the predicted probability and actual label bit as input in this context.

The training process involves optimizing the parameters of the function h to minimize the loss function across all training examples:

$$min_h \frac{1}{N} \sum_{i=1}^{N} L(h_i, y_i)$$

The pointwise method is conceptually simpler and more straightforward to implement compared to pairwise and triplet methods since they do not require the construction of pairs or triplets of samples. However, in the mean time, this method may not capture the relative similarities between different items as effectively, as they focus solely on individual items and their labels. The performance is also heavily dependent on the quality and correctness of the labels (Chen et al. (2007)). Any errors or ambiguities in the labels can directly affect the quality of the learned hash codes.

2.3.5 Quantization

Quantization technique is used to bridge the gap between the continuous features generated by deep neural networks and the discrete binary codes needed for efficient retrieval (Singh and Gupta (2022)). After generating deep features, we need to quantize these binary features into binary hash codes. Product quantization is a popular approach for this step, which involves partitioning the feature space into smaller sub-spaces that can be independently quantized. We can frame the quantization loss with the objective to minimize the loss between the original deep features and their quantized versions:

$$L_{quant}(h_i, b_i) = ||h_i - q(b_i)||^2$$

To ensure that the quantized features preserve the semantic relationships inherent in the data, quantization loss is often combined with other types of loss functions

such as pairwise difference loss (Cao et al. (2018)), pairwise likelihood loss (Liu et al. (2018a)) and triplet loss (Liu et al. (2018b)). In the work of Cao et al. (2018), DNNs are involved in the product quantization, which might be used to dynamically determine how the feature space could be partitioned for optimal quantization, based on that data distribution learned during training. By integrating DNNs, the quantization process become more adaptive to the data, improving how well the quantized codes represents the original features. Additionally, while traditional quantization methods can lead to significant information loss, especially when compressing large feature sets into small codes. DNNs can help minimize this loss by learning to preserve essential information through more efficient encoding strategies. Hence, the performance is improved through this end-to-end training where both feature extraction and quantization are optimized simultaneously (Yang et al. (2014)).

3 Methods

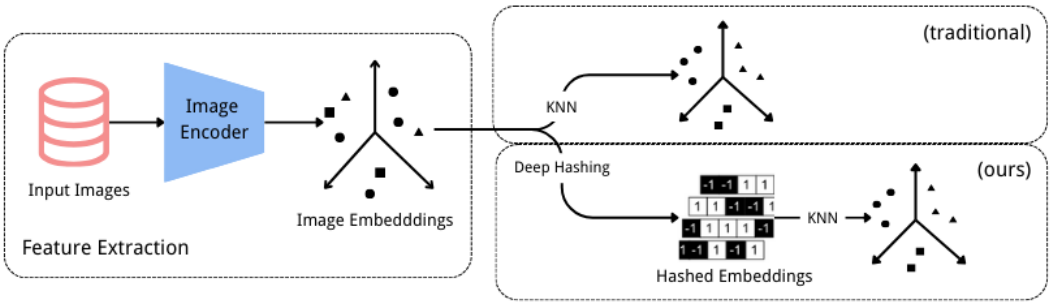

Figure 4: Illustration of our proposed approach. Firstly, the image embeddings are generated though ViT-B/16 image encoder. KNN is applied on these image embeddings to evaluate accuracy as a baseline metric. Secondly, a series of experiments were conducted to transfer the image embeddings to hashed image embeddings through various deep supervised hashing methods, mainly including pairwise and triplet methods. We again retrieve the accuracy using KNN on these hashed image embeddings to make a comparison between the baseline accuracy.

Fig.4 illustrates our proposed approach consisting of 1. Feature Extraction, 2. Deep Hashing, and 3. Inference using KNN. The following section will provide an overview of each phase.

3.1 Feature Extraction

Feature extraction is a pivotal step in our methods, enabling the transformation of raw image data into image embeddings. In this study, we employ the Vision Trans-

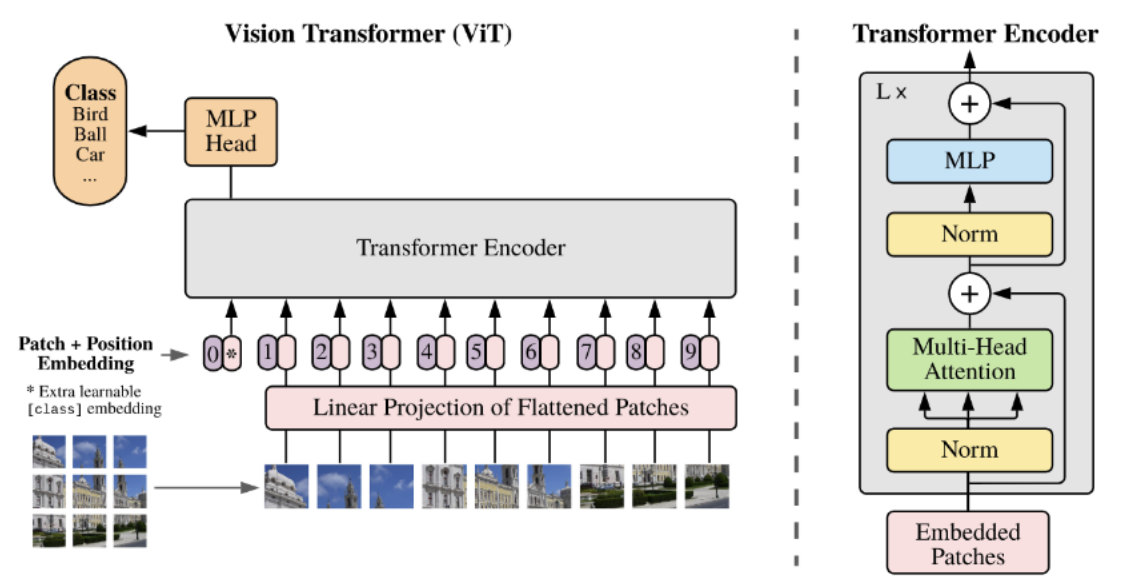

Figure 5: Vision transformer model overview (Dosovitskiy et al. (2020)). Image is split into fixed-size patches, which are then linearly embedded and combined with a position embeddings. The resulting sequence of vectors is then fed into the transformer encoder. To perform classification, an extra learnable "classification token" is added to the sequence.

former (ViT) (Dosovitskiy et al. (2020)) as our primary image encoder to extract and subsequently store feature embeddings and their associated labels $(f_i, y_i) \in F$ for each labeled input image $(x_i, y_i) \in X$, with an associated categorical class distribution C (Di Salvo et al. (2024)). The extracted feature representations f_i represent the positional information of x_i within the feature space of selected feature extractor (Di Salvo et al. (2024)). Unlike CNNs that process images through localized filters, ViT divides an image into fixed-size patches. These patches are then flattened, linearly embedded, and subsequently processed in a sequence-like manner akin to tokens in a language model. Each patch is also associated with a positional encoding to retain order information. This setup allows the transformer to leverage its self-attention mechanism, enabling it to weight the importance of different patches as it processes the image, thereby capturing both local and global features effectively.

We specifically rely on the DINOv2 (Oquab et al. (2023)) architecture for embedding extraction, which is a self-supervised learning framework built upon the ViT architecture. DINOv2 leverages a self-distillation approach where the network is trained to predict its own ouput under different views of the same image, facilitated by data augmentation techniques. This method encourages the model to discover invariant and discriminative features across various perturbations of the input data. The core of DINOv2 involves two key components: a teacher network and a student network (Doan et al. (2022)). Both networks share the same architecture but are initialized differently. During training, the student network learns to predict the output of

the teacher network. Importantly, the teacher network's parameters are updated as an exponential moving average of the student network's parameters (Zhang et al. (2019)), which stabilizes the learning process over time. Through this procedure, DINOv2 effectively learns to generate embeddings that compactly represent the essential features of the input images. These embeddings are high-dimensional vectors that capture the salient characteristics of the images, making them suitable for downstream tasks such as similarity search, where distances between embeddings directly correlate to visual similarities between images. The embeddings generated by DINOv2 are particularly useful for our purposes as they are designed to be robust to variations in input data, thereby enhancing both the effectiveness and reliability of hashing methods applied subsequently.

3.2 Hashing Techniques

In this section, we'll introduce the hashing techniques applied in our experiments. Table 3.2 summarizes the deep supervised hashing methods with different manner of similarity measurements.

Approach	Pairwise	Ranking-based	Pointwise	Binarization	Other skills
DSH	Prod + Margin	-	-	Quan	-
CNNH	Diff.	-	Part of Hash Codes	-	Two-step
DPSH	Like	-	-	Quan	-
DHN	Like	-	-	Quan + Smooth	-
DSDH	Like	-	Linear Reg + L2	Quan + Alternation	-
HashNet	Weighted Like	-	-	Tanh + Continuation	-
DSHGAN	-	Triplet + Margin	Cla.Layer	Drop	GAN
DNNH	-	Triplet + Margin	-	Piecewise Thresholding	-
TCL	Center-focus	Triplet + Margin	-	Sigmoid/Thresholding	Central Metric

Table 1: A Summary of Deep Supervised Hashing Methods w.r.t the Different Manner of Similarity Measurement (Pairwise Methods, Ranking-based Methods, Pointwise Methods, Binarization and other skills.) Diff. = Difference Loss, Prod. = Product Loss, and Like. = Likelihood Loss, Quan. = Quantization Loss, Drop = Drop the sign operator in the neural network and treat the binary code as an approximation of the network output, Two-step = Two-step optimization, Reg. = Regression, Cla. = Classification.

3.2.1 Pairwise Methods

Deep Supervised Hashing (DSH) Liu et al. (2016) Deep supervised hashing (DSH) combines the representation learning of deep neural networks with the efficiency of hashing-based nearest neighbor search. It builds upon the ideas from earlier hashing techniques like Spectral Hashing (Weiss et al. (2008)) but adapts them within a deep learning framework. This adaptation allows DSH to learn more

complex and high-level image representations, which are more effective for image retrieval compared to traditional methods.

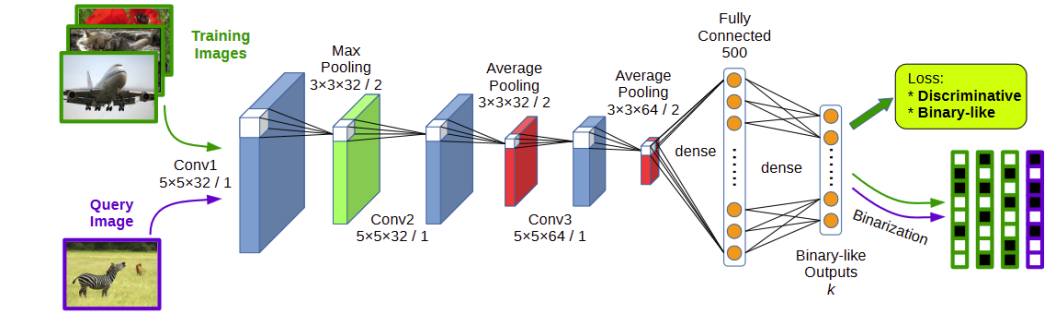

Figure 6: Approach in Liu et al. (2016): The network consists of 3 convolution-pooling layers and 2 fully connected layers. The filters in convolution layers are of size 5×5 with stride 1 (32, 32, and 64 filters in the three convolution layers respectively), and pooling over 3×3 patches with stride 2. The first fully connected layer contains 500 nodes, and the second (output layer) has k (the code length) nodes. The loss function is designed to learn similarity-preserving binary-like codes by exploiting discriminability terms and a regularizer. Binary codes are obtained by quantizing the network outputs of images.

DSH typically utilizes a deep convolutional neural network (CNN) structure that includes three convolutional pooling layers and two fully connected layers. The outputs of the DSH network are binary-like codes $h_{i i=1}^{N}$. The goal is to train the network so that these codes effectively preserve the semantic similarities among the images. The original pairwise loss function for DSH is defined as:

$$L_{DSH} = \sum_{(i,j)\in\mathcal{E}} (\frac{1}{2} s_{ij}^{o} d_{ij}^{h} + \frac{1}{2}(1 - s_{ij})[\epsilon - d_{ij}^{h}]_{+})$$

where $[\cdot]$ denotes $max(x, 0)$, implementing a hinge-like feature and $\epsilon > 0$ is a threshold parameter that sets a margin for dissimilar pairs. This loss function encourages the network to map similar images to close points in the binary code space and dissimilar images to distant points.

DSH also introduces a relaxation to the binary constraints of the output codes by approximating them with continuous outputs of the network, which are then regularizes towards binary values:

$$L_{DSH} = \frac{1}{2} s_{ij}^{o} ||h_i - h_j||_2^2 + \frac{1}{2}(1 - s_{ij})[\epsilon - ||h_i - h_j||_2^2]_{+} + \lambda_1 \sum_{k=i,j} |||h_k| - 1||_1$$

where λ_1 is a regularization parameter that controls the strength of the penalty pushing the hash values towards -1 or 1. The term $|||h_k| - 1||_1$ penalizes deviations from -1 and 1, effectively regularizing the hash codes to be close to binary values.

DSH employs end-to-end backpropagation for training, where the gradients of the loss function are used to update the weights of the CNN. This training process does not utilize saturating nonlinearities (like sigmoid or tanh at the final layer) because these can potentially slow down the learning due to vanishing gradients. For generating binary codes from the trained network, a sign activation function is applied to the output of the network (Yang et al. (2019)). This function converts each element in the output to -1 or 1 based on its sign, producing the final binary hash codes (Luo et al. (2023)).

Algorithm 2 Deep Supervised Hashing (DSH)

Input: Image set X with n images; Similarity matrix $S \in \{0,1\}^{n \times n}$; Number of hash bits L; Learning parameters: learning rate η, regularization parameter λ_1, margin threshold ϵ; Maximum epochs E
Output: Binary hash codes H for all images; Trained CNN parameters Θ
procedure DSH TRAINING: Initialize CNN with random weights Θ and binary hash codes $H \in \mathbb{R}^{n \times L}$ randomly
  for each epoch **do**
    for each minibatch in X **do**
      Compute hash codes for the minibatch; Compute pairwise losses and regularization; Update Θ by backpropagating losses
    end for
  end for
end procedure
procedure GENERATE BINARY CODES
  for each image in X **do** Compute hash code using CNN and sign function
  end for
end procedure
Return H and Θ

DSH represents an early yet influential approach in the realm of deep supervised hashing, setting foundational concepts for subsequent developments in hashing techniques that leverage deep learning.

Convolutional Neural Network Hashing (CNNH) (Xia et al. (2014)) CNNH uses the two-step strategy to address the complex challenge of learning to hash with neural networks as illustrated in Fig.7. This strategy involves two main phases:

1. Learning approximate hash codes using coordinate descent. The goal is to learn an initial set of binary codes that approximate the ideal binary hash codes. This is typically achieved by minimizing an objective function that quantifies the similarity between the learned codes and the desired semantic relationships among the images (Choi et al. (2018)):

$$L_{CNNH} = ||\frac{1}{L} H H^T - S||^2$$

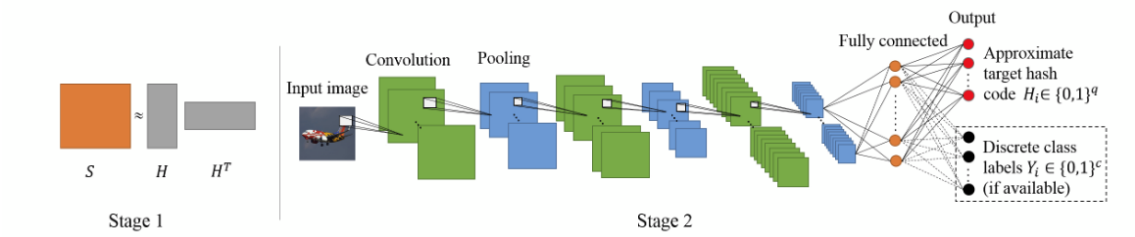

Figure 7: Overview of two-stage method: 1. the pairwise similarity matrix S is decomposed into a product HH^T, where H is a matrix of approximate target hash codes. 2. use a convolutional network to learn the feature representation for the images as well as a set of hash functions. The network consists of three convolution-pooling layers, a fully connected layer and an output layer. The output layer can be simply constructed with the learned hash codes in H (the red nodes). If the image tags are available in training, one can add them in the output layer (the black nodes) so as to help to learn a better shared representation of the images. By inputting an test image to the trained network, one can obtain the desired hash code from the values of the red nodes in the output layer (Xia et al. (2014)).

where H is the matrix of hash codes for all training images, S is the similarity matrix, where S_{ij} is 1 if images i and j are similar and 0 otherwise. L is the normalization factor, typically the number of hash bits. This objective function aims to make the inner product between the hash codes of any two images approximate the semantic similarity defined by S. The function penalizes deviations from this desired relationship, pushing $\frac{1}{L}HH^T$ to closely mirror S. The optimization of this function is typically performed using a coordinate descent strategy, where each coordinate (or a subset of coordinates) is optimized iteratively while fixing the others. This process helps to converge towards a set of hash codes that preserve semantic relationships as encoded in S.

2. Training the CNN with hash codes. Once the approximate binary codes are generated, the second step involves fine-tuning the convolutional neural network using these hash codes. This training involves adjusting the network weights to minimize the discrepancy between the outputs of the CNN (in the last layer before the hashing layer) and the hash codes (Xu et al. (2018)). This way, the CNN learns to produce feature representations that directly correlate with the binary codes, effectively making the process of generating binary codes a part of the forward pass of the network in future deployments. If class labels are available, a fully connected layer with K outputs units (corresponding to the K class labels) is added to the CNN architecture (Luo et al. (2023)). This layer is trained to perform image classification concurrently with the hashing task. The overall loss function is then a combination of the hashing loss (to ensure that the CNN's output matches the binary codes) and the

classification loss, allowing the network to optimize for both accurate image retrieval and correct classification.

Algorithm 3 Convolutional Neural Network Hashing (CNNH)

Input: Training images $X = \{x_i\}_{i=1}^{N}$; Similarity labels $S = \{s_{ij}\}$, indicating whether pairs (i, j) are similar
Output: Hash codes H for all images; Trained CNN parameters Θ
Initialization: Initialize CNN parameters Θ randomly; Initialize hash codes H randomly or based on preliminary features
Training Process:
while not converged **do**
    Select a batch of images from X
    Perform forward propagation through CNN to compute features
    Compute approximate binary codes for the batch
    Calculate the loss function using S and the computed codes
    Backpropagate the loss and update Θ
end while
Generate Hash Codes:
for each image x_i in X **do**
    Compute final binary hash code h_i using the trained CNN
end for
Return Hash codes H and trained parameters Θ

The time complexity of the CNNH algorithm involves multiple factors, including the complexity of the hashing bit learning and the CNN training. Assuming efficient management of matrix updates, the hashing part's complexity is $O(Tnq)$. CNN training complexity is largely dependent on the specific architecture but is generally significant due to the computational cost of convolutions. Overall, complexity of CNN training needs to be considered, especially for large-scale image datasets.

Deep Pairwise Supervised Hashing (DPSH) Li et al. (2015) DPSH utilizes a deep convolutional neural network (CNN) as the backbone for feature extraction and hash code generation. A common choice for the architecture is CNN-F, which is known for its robustness in handling visual data. This network architecture typically consists of several convolutional layers followed by pooling layers and fully connected layers, culminating in a hashing layer that produces the hash codes. In DPSH, the final output layer of the CNN is specifically designed to generate binary-like hash codes. These codes are the embeddings that represent the input images in a compressed binary form, facilitating efficient storage and fast retrieval.

The loss function in DPSH is composed of two main components:

1. Likelihood loss. This loss component is defined based on the pairwise similarity between images. It uses the standard form of likelihood loss that is modified to suit the hashing context.

$$L_{DPSH}^{likelihood} = -\sum_{(i,j)\in\mathcal{E}} (s_{ij}^{o} s_{ij}^{h}) - log(1 + e^{s_{ij}^{h}})$$

where s_{ij}^{o} indicates the ground truth similarity between images i and j. $s_{ij}^{h} = \frac{1}{2} h_i^T h_j$ is the cosine similarity transformed output between the hash codes of images i and j, which maps the real-valued outputs of the network to a value that indicates the degree of similarity. This part of the loss function encourages images that are similar (as per the label information) to have hash codes that yield a higher dot product value, thus being closer in the Hamming space, and vice versa.

2. Quantization loss. To ensure the outputs of the network are close to actual binary values, quantization loss is added:

$$L_{DPSH}^{quant} = \lambda_1 \frac{1}{N} \sum_{i=1}^{N} ||h_i - sgn(h_i)||_2^2$$

This term penalizes the deviation of the network outputs from -1 or 1, effectively pushing the continuous outputs towards binary values. λ_1 is a regularization parameter that controls the trade-off between the similarity-preservation and the binarization of the hash codes.

Algorithm 4 Learning Algorithm for Deep Pairwise Supervised Hashing (DPSH)

Input: Training images $X = \{x_i\}_{i=1}^{n}$; Set of pairwise labels $S = \{s_{ij}\}$
Output: Parameters W, v, θ and binary codes B
Initialization: θ with the CNN-F model; Each entry of W and v by randomly sampling from a Gaussian distribution with mean 0 and variance 0.01
repeat
    Randomly sample a minibatch of points from X
    for each sampled point x_i **do**
        Calculate $\phi(x_i; \theta)$ by forward propagation
        Compute $u_i = W^T \phi(x_i; \theta) + v$
        Compute the binary code of x_i with $b_i = \text{sgn}(u_i)$
        Compute derivatives for point x_i according to specified loss equations
        Update the parameters W, v, θ by utilizing back propagation
    end for
until a fixed number of iterations are completed

During training, DPSH learns both the deep features and the hash codes simultaneously through backpropagation, using the combined loss function $L_{DPSH} =$

$L_{DPSH}^{likelihood} + L_{DPSH}^{quant}$. This end-to-end training approach allows the network to effectively learn how to map raw image data directly to compact binary codes that preserve semantic similarity, which is a key advantage over methods that separate feature learning from hash code generation.

Deep Hashing Network (DHN) (Zhu et al. (2016)) Similar to other deep learning based hashing methods such as DPSH, DHN utilizes a CNN as its backbone for feature extraction. This CNN processes input images to extract rich feature representations, which are then transformed into binary codes. The architecture typically includes several layers of convolutions, pooling, and fully connected layers, ending in a hashing layer where the actual hash codes are generated.

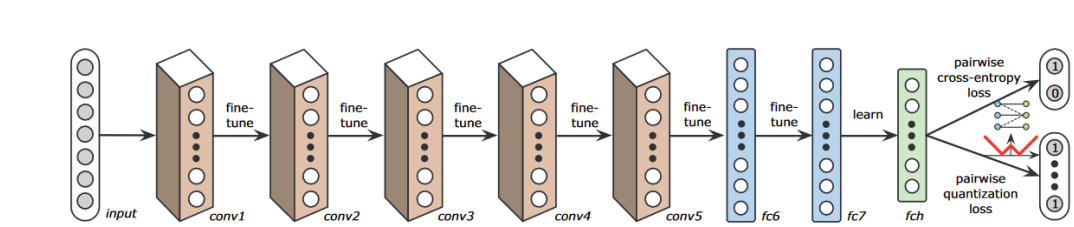

Figure 8: DHN with a hash layer fch, a pairwise cross-entropy loss, and a pairwise quantization loss. (Zhu et al. (2016))

DHN employs a combination of a likelihood loss function similar to that used in DPSH and a specially designed quantization loss, which incorporates Bayesian priors:

1. Likelihood loss. The likelihood loss is akin to the loss used in DPSH, focusing on preserving the semantic similarities between images through their hash codes. It aims to minimize the discrepancy between the hash codes of similar images and maximize it for dissimilar ones, typically using a pairwise or triplet loss formulation depending on the dataset and specific implementation details.

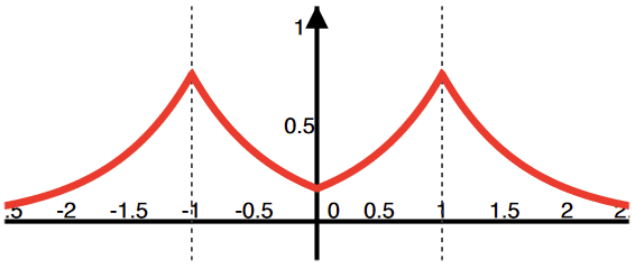

Figure 9: The bimodal Laplacian prior for quantization. Zhu et al. (2016)

2. Quantization loss with bayesian prior. DHN introduces a Bayesian prior modeled by a bimodal Laplacian distribution for each component of the hash code vectors h_i. This distribution is defined as

$$p(h_i) = \frac{1}{2\epsilon} exp(-\frac{||h_i - 1||_1}{\epsilon})$$

where ϵ is a diversity parameter that adjusts the sharpness of the distribution around ± 1, which are the desired values for binary codes as illustrated in Fig.9. The use of the Laplacian prior provides a probabilistic foundation to the hashing process, viewing each bit in the hash code as a random variable following the specified distribution. The quantization loss in DHN is then formulated as the negative log-likelihood of this Bayesian prior, which effectively measures how well the distribution of the hash codes h_i aligns with the expected binary values under the prior. The loss is given by:

$$L_{Quan} = N \sum_{i=1}^{N} ||h_i - 1||_1$$

The L_1 norm is used here instead of the L_2 norm, as it not only provides an upper bound for the L_2 norm but also promotes sparsity and pushes the values of h_i towards zero or away from it, ideally setting at ± 1. By minimizing the negative log-likelihood, the network learns to generate hash codes that maximize the posterior probability under this distribution.

To further align the quantization loss with the Bayesian prior and make the optimization process more tractable, DHN employs a smooth surrogate for the L_1 loss:

$$L_{Quan} = N \sum_{i=1}^{N} \sum_{l=1}^{L} log(cosh(|h_{il} - 1|))$$

This smoothing with the log-cosh function helps in handling the optimization challenges typically associated with the non-differentiability of the L_1 norm at zero, providing a gradient that is more stable and continuous to gradient-based optimization methods.

DHN is trained using backpropagation with the overall loss being a combination of the likelihood and quantization losses. The network learns to generate hash codes that not only preserve semantic content but are also close to binary, thus reducing the retrieval time and storage space typically required for real-valued vectors.

HashNet Cao et al. (2017b) HashNet addresses specific challenges in training deep hashing networks, particularly focusing on the issue of training imbalance and the non-linearity in transforming continuous activations to binary hash codes.

One of the significant challenges in deep hashing is the imbalance between similar and dissimilar image pairs, which can skew the learning process and affect the quality

Algorithm 5 Deep Hashing Network (DHN)

Input: Training images $X = \{x_i\}_{i=1}^N$; Pairwise similarity labels $S = \{s_{ij}\}$
Output: Binary hash codes H; Trained network parameters Θ
Initialization: Network parameters Θ with pretrained or random values; Parameter for Laplacian prior ϵ
Training Process:
while not converged **do**
  Sample a minibatch from X
  Perform forward propagation to compute h_i for each x_i
  Calculate $L_{likelihood}$ based on S and h_i
  Compute quantization loss L_{Quan} using Laplacian prior
  Total loss $L = L_{likelihood} + L_{Quan}$
  Backpropagate the total loss and update Θ
end while
Generate Binary Codes:
for each x_i in X **do**
  $h_i =$ forward pass(x_i)
  $b_i = \text{sign}(h_i)$ ▷ Convert to binary
  Store b_i in H
end for
Return H and Θ

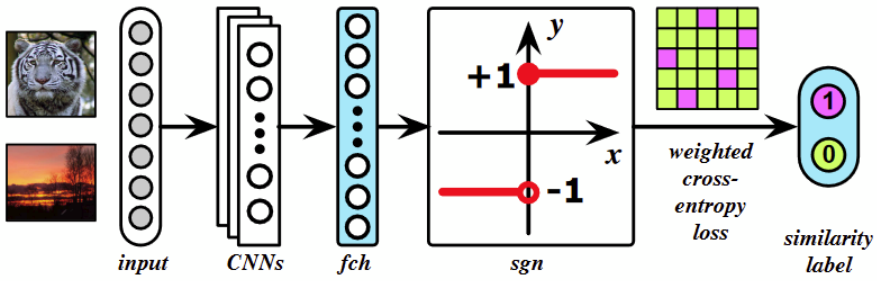

Figure 10: HashNet for deep learning to hash by continuation, which is comprised of four key components: (1) Standard CNN for learning deep image representations, (2) a fully-connected hash layer fch for transforming the deep representation into K-dimensional representation, (3) a sign activation function for binarizing the K-dimensional representation into K-bit binary hash code, and (4) a novel weighted cross-entropy loss for similarity-preserving learning from sparse data (Cao et al. (2017b)).

of the generated hash codes. HashNet introduces the Weighted Maximum Likelihood (WML) loss to handle the imbalance issue effectively as illustrated in Fig.10. For each image pair, a weight w_{ij} is assigned based on whether the pair is similar ($s_{ij} = 1$) or dissimilar ($s_{ij} = 0$):

$$w_{ij} = \begin{cases} \frac{|S|}{|S_1|}, & \text{if } s_{ij} = 1 \\ \frac{|S|}{|S_0|}, & \text{if } s_{ij} = 0 \end{cases}$$

where $|S_1|$ is the number of similar pairs and $|S_0|$ is the number of dissimilar pairs. For multiple-label datasets, the label overlap coefficient c_{ij} is calculated based on the overlap between the labels of the two images $\frac{y_i \cap y_j}{y_i \cup y_j}$, and it equals to 1 for single-label datasets.

HashNet modifies the sigmoid activation function used in calculating the conditional probabilities of the hash bits:

$$\sigma(x) = \frac{1}{1 + e^{-\alpha x}}$$

This adaption introduces a hyper-parameter α, which scales the input to the sigmoid function, effectively controlling the steepness of the activation curve and allowing the model to be more or less aggressive in classifying pairs as similar or dissimilar. The conditional probabilities are then defined as:

$$P(s_{ij}|h_i, h_j) = \begin{cases} \sigma(\langle h_i, h_j \rangle), & s_{ij} = 1 \\ 1 - \sigma(\langle h_i, h_j \rangle), & s_{ij} = 0 \end{cases}$$

The WML loss incorporates these probabilities into a logistic regression framework, adjusted for the imbalance by using weights w_{ij} for each pair:

$$L_{WML} = min_\theta \sum_{s_{ij} \in S} w_{ij}(log(1 + e^{\alpha \langle h_i, h_j \rangle}) - \alpha s_{ij} \langle h_i, h_j \rangle)$$

$\alpha \langle h_i, h_j \rangle$ represents the input to the adaptive sigmoid function scaled by α, enhancing the model's capacity to deal with the linear separability of the data in the transformed feature space.

To overcome the non-differentiability of the sign function, HashNet employs a continuation method by approximating the sign function with differentiable functions during training. The typical choice for approximation is the hyperbolic tangent function ($tanh$), as it provides outputs between -1 and 1, similar to the sign function, but it is smoothly differentiable as illustrated in Fig.11. The steepness of the $tanh$ function can be controlled by a scaling parameter β, making $tanh(\beta x)$ increasingly steep as β increases:

$$lim_{\beta \to \infty} tanh(\beta z) = sgn(z)$$

Continuation learning refers to gradually increasing the parameter β during the training process. This technique starts with a smaller value of β, making the $tanh$

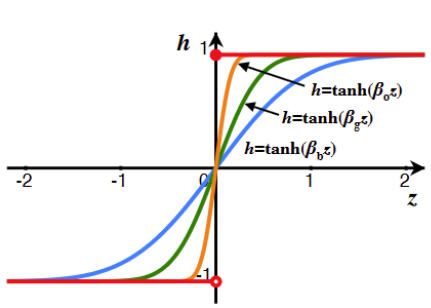

Figure 11: Plot of smoothed responses of the sign function $h = sgn(z)$: Red is the sign function, and blue, green and orange show functions $h = tanh(\beta z)$ with bandwidths $\beta_b < \beta_g < \beta_o$. The key property is $lim_{\beta \to \infty} tanh(\beta z) = sgn(z)$. (Cao et al. (2017b))

function relatively flat, and gradually increases β to make it sufficiently large so that $tanh(\beta x)$ closely approximates the $sgn(x)$ function, effectively pushing the outputs towards -1 or 1, thus producing binary hash codes.

Algorithm 6 Optimizing HashNet by Continuation

Input: A sequence $1 = \beta_0 < \beta_1 < .. < \beta_m = \infty$
for stage $t = 0$ to m **do**
  Train HashNet(WML loss) with $tanh(\beta)$ as activation
  Set converged HashNet as next stage initialization
end for
Output: HashNet with $sgn(z)$ as activation, $\beta_m \to \infty$

The combination of these techniques - WML loss for training balance, adaptive sigmoid for flexible activation, tanh for smooth approximation of the sign function, and continuous learning - significantly enhances the performance of HashNet over other deep hashing methods. These innovations lead to more robust, accurate, and efficient image retrieval systems capable of handling large-scale datasets and complex image distributions.

Deep Supervised Discrete Hashing (DSDH) (Li et al. (2017)) The overall loss function in Deep Supervised Discrete Hashing (DSDH) includes several terms designed to address different objectives:

1. Pairwise similarity loss, which is modeled using a logistic regression-based approach:

$$L_{similarity} = -\sum_{(i,j)\in\mathcal{E}} (s_{ij}^o s_{ij}^h - log(1 + e^{s_{ij}^h}))$$

where s_{ij}^o is the ground truth similarity and $s_{ij}^h = \frac{1}{2} h_i^T h_j$ represents the similarity derived from the hash codes.

2. Quantization loss, which encourages the outputs of the network h_i to approximate binary values:

$$L_{quant} = \lambda_1 \frac{1}{N} \sum_{i=1}^{N} ||h_i - sgn(h_i)||_2^2$$

3. Linear regression loss, which utilizes label information for guiding the hash codes:

$$L_{regression} = \lambda_2 ||Y - W^T B||$$

Here, Y represents the matrix of one-hot encoded labels, B is the matrix of binary codes for the images, and W is a learnable transformation matrix. This term aligns the hash codes with the label space, enhancing the label-specific discriminative power of hash codes.

4. Regularization on transformation matrix, which prevents overfitting:

$$L_{regularization} = \lambda_3 ||W||_F$$

The term adds a Frobenius norm ($||W||_F := \sqrt{\sum \sum |w_{ij}|^2}$) regularization on the matrix W.

The optimization mechanisms in DSDH involve gradient descend for continuously updating parameters such as the transformation matrix W and intermediate continuous outputs h_i, and discrete cyclic coordinate descent, which is specially employed for updating the binary codes B directly, ensuring that despite the linear regression model, the discreteness of the hash codes is maintained. This approach cyclically optimizes each bit of the hash codes while fixing others, effectively handling the discrete nature of the problem.

DSDH's strategy to simultaneously leverage label and pairwise similarity information allows it to outperform many traditional and some deep learning-based hashing methods, especially in scenarios where label information is rich and indicative of underlying semantic structures. The integration of linear regression with traditional hashing loss components helps in creating more semantically meaningful and compact binary hash codes, significantly enhancing retrieval accuracy and efficiency.

3.2.2 Triplet Methods

Deep Neural Network Hashing (DNNH) (Lai et al. (2015)) DNNH is built around the idea of triplet ranking objective (Norouzi et al. (2012)), a method that considers the relative similarity between triplets of samples. This method ensures that the learned binary codes reflect the relative semantic similarities among the samples, enhancing the efficacy of the retrieval system. The DNNH model is composed of three primary components: 1. Sub-network with convolution-pooling layers, 2. Divide-and-Encode module, and 3. Triplet ranking loss layer as illustrated in Fig.12.

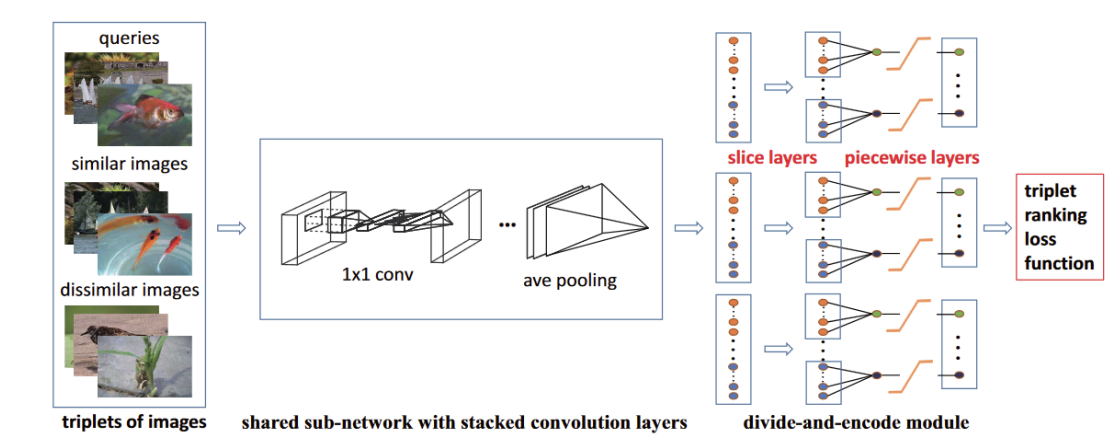

Figure 12: Overview of DNNH. The input is in the form of triplets (I, I^+, I^-) with a query image I being more similar to an image I^+ than to another image I^-. The image triplets are first encoded into a triplet of image feature vectors by a shared stack of multiple convolutional layers. Then, each image feature vector in the triplet is converted to a hash code by a divide-and-encode module. After that, these hash codes are used in a triplet ranking loss that aims to preserve relative similarities on images. (Lai et al. (2015))

In the sub-network, the architecture generally follows the "Network in Network" (NiN) model (Lin et al. (2013)), which includes additional 1×1 convolution layers (pointwise convolution) inserted after traditional convolutions. These 1×1 convolutions act as per-channel linear transformations of the input maps, introducing additional non-linearity and capacity to the model. Instead of fully connected layers, an average pooling layer concludes the sub-network. This choice reduces overfitting and decreases the number of learnable parameters, focusing on capturing the most significant features. The sub-network is shared across all three images in the triplet, significantly reducing the number of parameters and ensuring that the network learns a unified representation across different but related images.

After processing through the sub-network, the image features are divided into slices. The division process partitions the output feature vector into equal segments, with each segment corresponding to one bit of the final hash code. For example, if the target hash code length is q bits then the feature vector from the pooling layer is $50q$ dimensions, the vector is divided into q slices, each of 50 dimensions. Each sliced segment undergoes a transformation, which is crucial as it aggregates the information from each segment, ensuring that each bit of the hash code encapsulates a distinct aspect of the feature vector, thereby reducing redundancy across the bits. After dimension reduction, each output is passed through a sigmoid activation function. The sigmoid function normalizes the output to a range between 0 and 1, making

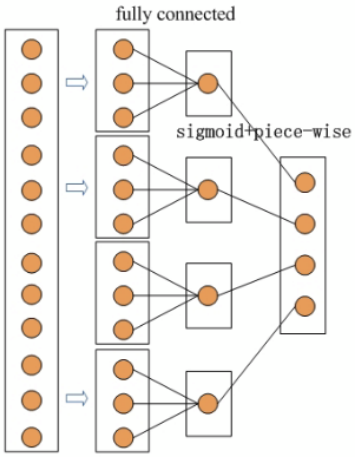

Figure 13: A divide-and-encode module. (Lin et al. (2013))

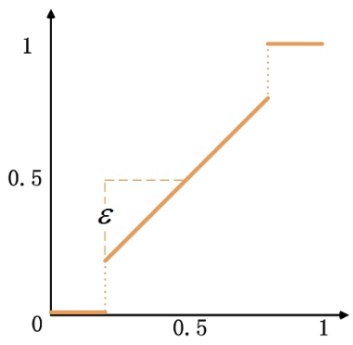

Figure 14: The piece-wise threshold function. (Lin et al. (2013))

it suitable for thresholding. A piece-wise threshold function then processes these sigmoid outputs to enforce binary values:

$$g(s) = \begin{cases} 0, & s < 0.5 - \epsilon \\ s, & 0.5 - \epsilon < s < 0.5 + \epsilon \\ 1, & s > 0.5 + \epsilon \end{cases}$$

where ϵ is a small positive hyper-parameter that defines the sensitivity of the thresholding around 0.5, effectively creating a "soft" binary transition region around this midpoint.

The loss function used in DNNH is designed to handle triplets of data points. Given a triplet (x_i, x_j, x_k), where x_i is considered the anchor, x_j is a positive sample, and x_k is a negative sample. The goal is to ensure that the hash codes for x_i is closer to the hash code for x_j than to the hash code for x_k:

$$L_{DNNH}(h_i, h_j, h_k) = max(0, 1 + d^h_{ij} - d^h_{ik})$$

By substituting the Euclidean distance for the Hamming distance, the loss function is made convex:

$$L_{DNNH}(h_i, h_j, h_k) = max(0, 1 + ||h_i - h_j||^2_2 - ||h_i - h_k||^2_2)$$

This convex formulation allows for straightforward optimization through gradient-based methods, facilitating effective learning in the neural network.

Approximate time complexity analysis: The forward pass through the CNN is the most computationally intensive part. The complexity depends on the number of layers, the size of the filters, and the dimensions of the input images. For simplicity, if we consider all layers to have roughly equal computational load, and each layer operates in $O(n)$ time for one image, then processing one triplet would be $3 \times O(n)$.

Algorithm 7 Deep Neural Network Hashing (DNNH)

Input: Training triplets of images (x_i, x_j, x_k)
Output: Binary hash codes for images
Initialize parameters of CNN
Training Process:
while not converged **do**
  for each triplet (x_i, x_j, x_k) in the training set **do**
    $h_i, h_j, h_k \leftarrow$ forward pass of x_i, x_j, x_k through CNN
    $b_i, b_j, b_k \leftarrow$ divide-and-encode module on h_i, h_j, h_k
    Compute triplet loss $L = \max(0, 1 + \|b_i - b_j\|^2 - \|b_i - b_k\|^2)$
    Backpropagate loss and update CNN parameters
  end for
end while
Return binary hash codes for all images

If the output vector has m elements and we divide it into q slices, the complexity of processing each slice can be considered $O(m/q)$. Processing all slices for one image thus contributes $O(m)$, and for a triplet, it's $3 \times O(m)$. The triplet loss involves calculating the squared Euclidean distances between the hash codes of three images and then applying the loss formula. Computing the distance between two vectors of length q is $O(q)$, so for the two distances needed in the triplet loss, the time complexity is $2 \times O(q) = O(q)$ for each triplet. Combining all parts, the time complexity per training iteration for a single triplet is dominated by the forward pass through the CNN and the divide-and-encode module, roughly approximating to $3 \times O(n) + 3 \times O(m) + O(q)$. If n and m are of similar magnitude and significantly larger than q, the overall per-triplet complexity in each iteration can be approximated as $O(n)$. In training, the total computational load will depend on the number of triplets processed, which typically scales with the size of the training dataset.

Deep Semantic Hashing with GAN (DSH-GAN) Qiu et al. (2017) Deep Semantic Hashing with GAN (DSH-GAN) is a complex and innovative method that integrates Generative Adversarial Networks (GANs) into the hashing process for image retrieval tasks. The system consists of the key components: (1). Shared CNN for learning image representations, which captures the deep features of images, forming the backbone for the other components. (2). Adversary Stream for distinguishing between synthetic and real images. (3). Hash Stream for encoding images into binary hash codes. (4). Classification Stream, which utilizes class labels to enforce semantic integrity in the hashing process.

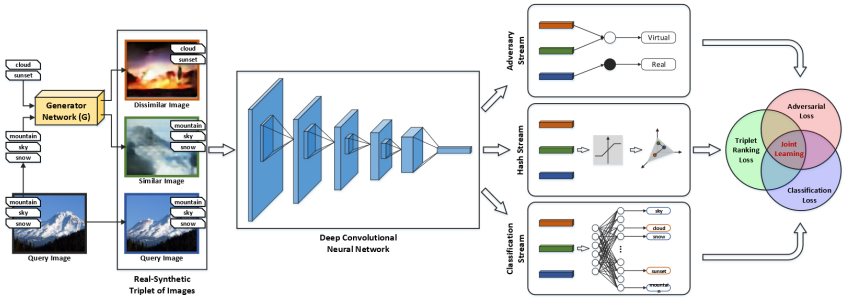

Figure 15: Deep Semantic Hashing with GANs framework. The input to DSH-GANs architecture is in the form of real-synthetic image triplets and each tuple consists of one real image as query image, one synthetic and similar image produced with same labels of query image through generator network G, and another synthetic but dissimilar image synthesized by G conditioning on different labels. A shared deep CNN is exploited for learning image representations, followed by three streams, i.e., hash stream, adversary stream and classification stream. Hash stream is to encode each image into hash codes with relative similarity preservation measured by a triplet ranking loss. Adversary stream is to distinguish synthetic images from real ones trained with an adversarial loss. Classification stream is to characterize the semantic structures on image and softmax loss or cross entropy loss is computed for single label and multi-label classification, respectively. The whole architecture is jointly optimized in an end-to-end fashion. (Qiu et al. (2017))

Algorithm 8 Training DSH-GANs

Input: Set of real images with labels, number of epochs, batch size
Output: Trained generator G and discriminator D
Initialize parameters of generator G and discriminator D
Pre-train G on labeled and unlabeled data (if applicable)
for each epoch **do**
  for each batch **do**
    $X_{real}, labels \leftarrow$ sample_real_images(batch_size)
    $X_{syn} \leftarrow G$.generate(X_{real})
    Train Discriminator: $D\_loss \leftarrow$ compute_D_loss($X_{real}, X_{syn}, labels$)
    Update D to minimize D_loss
    Minimizing adversarial and classification loss
    Train Generator: $G\_loss \leftarrow$ compute_G_loss($X_{real}, X_{syn}, labels$)
    Update G to minimize G_loss
    Minimizing triplet ranking loss and fooling D
  end for
end for

The semi-supervised GANs setup in DSH-GAN includes a generator G and a discriminator D. The generator in DSH-GAN plays a crucial role by creating synthetic images. It combines a noise vector (randomly generated) with a label embedding

to produce images that both look realistic and maintain semantic properties consistent with the label. The discriminator's objective is twofold: accurately distinguish between real and synthetic images and correctly classify the semantic labels of all images. The training of DSH-GAN involves feeding the network with triplets of images: a real image x, a synthetic image generated with the same label as the real image x^+_{syn}, and a synthetic image generated with a different label x^-_{syn}. Given the triplet $(x, x^+_{syn}, x^-_{syn})$, the adversarial loss for the discriminator, aimed at correctly identifying the source of an image is defined as:

$$\hat{l}_a(x, x^+_{syn}, x^-_{syn}) = \frac{1}{3}(l_a(x), l_a(x^+_{syn}), l_a(x^-_{syn}))$$

where the log-likelihood adversarial loss l_a for each image is defined as:

$$l_a = \begin{cases} -logP(S = real|x), & \text{if} x \in \text{real images} \\ -logP(S = synthetic|x), & \text{if} x \in \text{synthetic images} \end{cases}$$

In converse, the generator is trained to fool the discriminator by maximizing the adversarial loss, producing synthetic images that the discriminator misclassifies, enhancing the realism and semantic accuracy of generated outputs.

Hash Stream learns a hash function H to preserve the similarity relations in the real-synthetic triplets through the triplet ranking loss:

$$\hat{l}_{triplet} = max(0, 1 + ||H(x) - H(x^-_{syn})||^2_2 - ||H(x) - H(x^+_{syn})||^2_2)$$

ensuring that the hash code of x is more similar to x^+_{syn} than to x^-_{syn}.

Classification Stream This component leverages label information to ensure hash codes mirror semantic similarities, minimizing the classification error across real and synthetic images as:

$$\hat{l}_c(x, x^+_{syn}, x^-_{syn}) = \frac{1}{3}(l_c(x), l_c(x^+_{syn}), l_c(x^-_{syn}))$$

where l_c is computed using softmax loss in single label classification and cross entropy loss in multi-label classification.

Optimization The overall training of DSH-GAN involves minimizing a combined loss function that includes the triplet ranking loss, adversarial loss, and classification loss, balancing the need to create discriminative and generative capabilities within the model. The optimization of DSH-GAN uses a classic minimax mechanism common in GANs. The discriminator aims to maximize its classification accuracy by minimizing the term:

$$\hat{l}_{CNN} = \sum_{Triplets} (\hat{l}_{triplet} + \hat{l}_c(x) + \hat{l}_a(x))$$

where the shared CNN is trained to reserve the relative similarity ordering in the real-synthetic triplets and simultaneously recognize both correct sources and class

labels of images in the triplets. While the generator's loss, designed to fool the discriminator while maintaining semantic accuracy, is defined as:

$$\hat{l}_G = \sum_{Triplets} (\hat{l}_{triplet} + \hat{l}_c(x) - \hat{l}_a(x))$$

as the $l_c(x)$ helps to produce semantically correct images and $-l_a(x)$ effectively minimize the likelihood that the discriminator makes correct predictions. During the training process, both the discriminator and the generator update their weights concurrently. The discriminator improves its ability to distinguish real from fake, while the generator improves its ability to create convincing fakes. Training iterates until the generator achieves deception efficacy such that the discriminator classifies synthetic images as real at a rate equivalent to random guessing, signifying a state of equilibrium where neither model can improve unilaterally.

3.2.3 Triplet Center Loss in Deep Hashing

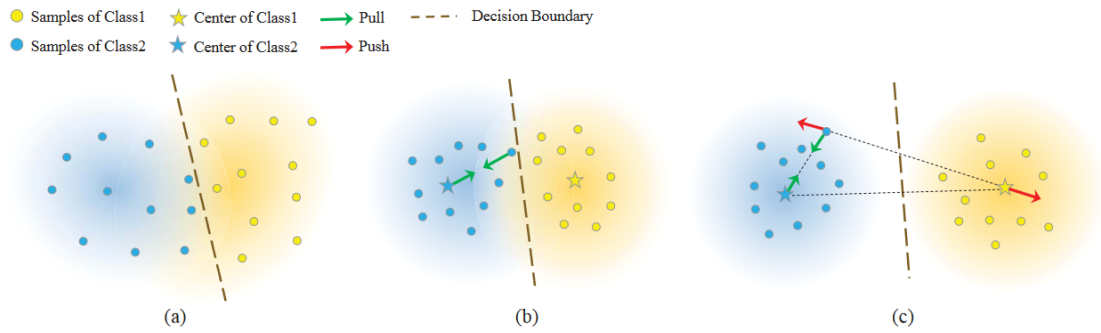

Figure 16: A toy illustration of the distributions of deep features learned by (a) softmax loss, (b) center loss + softmax loss, and (c) triplet-center loss + softmax loss. Intuitively, the decision boundary of the softmax classifier separates the two classes elaborately. The center loss pulls features toward their corresponding centers. The TCL pulls the features to their corresponding centers and pushes the features away from the other centers (He et al. (2018b)).

Although triplet loss is widely used for training deep neural networks on tasks that require learning fine-grained similarity distinctions, it still has several limitations. The effectiveness of triplet loss heavily depends on the selection of triplets. Poorly chosen triplets can lead to slow convergence or sub-optimal performance. While triplet loss drives apart different classes, it doesn't inherently ensure that all examples of a single class are close together in the embedding space, as it only considers relative distances within selected triplets. As the number of classes grows, the potential combinations of triplets explode exponentially, making it computationally expensive to find the most informative triplets.

Thus, to further enhance the feature discrimination capability of hashing methods, particularly in terms of managing intra-class compactness and inter-class separability

Algorithm 9 Triplet Center Loss (TCL) in Deep Hashing

Input: Training dataset $\{(x_i, y_i)\}_{i=1}^{N}$, where x_i are samples and y_i are labels, Number of classes $|Y|$, feature dimension d
Output: Trained model parameters, class centers $\{c_1, c_2, \ldots, c_{|Y|}\}$
Initialize neural network parameters f_θ, class centers $\{c_y\}_{y=1}^{|Y|}$ randomly in $\mathbb{R}^d$
Define loss functions:
Define squared Euclidean distance $D(f_i, c_y) = \frac{1}{2}\|f_i - c_y\|^2$ and margin m
Training Process:
while not converged **do**
  for each mini-batch $\{(x_i, y_i)\}$ **do**
    Compute features $f_i = f_\theta(x_i)$ for all x_i in the batch
    Compute distances to corresponding centers and nearest negative centers
    for each f_i in the batch **do**
      $c_{y_i} \leftarrow$ center for the class y_i
      $c_{q_i} \leftarrow$ nearest negative center not equal to y_i
      Compute $L_{tc_i} = \max(0, D(f_i, c_{y_i}) + m - \min_{j \neq y_i} D(f_i, c_j))$
    end for
    Compute total loss $L_{tc} = \sum L_{tc_i}$
    Backpropagate L_{tc} and update f_θ and $\{c_y\}$
  end for
end while
Return trained model and class centers

more effectively, we utilize the triplet center loss (TCL) (He et al. (2018b)) in deep hashing. TCL addresses the aforementioned limitations by incorporating concepts from both triplet loss and center loss. This hybrid approach retains the benefits of using triplets to define relative distances while also ensuring that each class's features are tightly clustered around a central point.

- **Intra-class:** By defining a center for each class and minimizing the distance from each data point to its respective class center, TCL systematically pulls all members of a class toward a common point.
- **Inter-class:** TCL not only minimizes the distance to the class center but also maximizes the distance to the nearest non-matching class center by a predefined margin. This dual focus goes beyond typical triplet loss by embedding an explicit mechanism to push classes apart, thus reinforcing inter-class boundaries more robustly.

Hence, assuming each class y has a center c_y in a d-dimensional space, each sample x_i with label y_i is mapped by the neural network $f_\theta(\cdot)$ into a feature f_i. The set

of all class centers is $C = \{c_1, c_2, ..., c_{|Y|}\}$, where $|Y|$ is the total number of classes. Given a batch of training data with M samples, TCL is defined as:

$$L_{TCL} = \sum_{i=1}^{M} max(D(f_i, c_{y_i}) + m - min_{j \neq y_i} D(f_i, c_j), 0)$$

where $D(f_i, c_{y_i})$ is the squared Euclidean distance between the feature f_i and its class center c_{y_i} defined as:

$$D(f_i, c_{y_i}) = \frac{1}{2}||f_i - c_{y_i}||_2^2$$

m is the margin that enforces f_i should be closer to its own class center c_{y_i} than any other class center by at least m. To train the model with TCL, we compute the gradients for backpropagation. The gradient of the loss L_{TCL} with respect to the feature embedding f_i is influenced by the distance to the correct center and the closest incorrect center:

$$\frac{\partial L_{TCL}}{\partial f_i} = (c_{q_i} - c_{y_i}) \cdot \mathbb{1}[\tilde{L}_i > 0]$$

where $q_i = argmin_{j \neq y^i} D(f_i, c_j)$ is an integer index which indicates the nearest negative center of i-th sample and $\tilde{L}_i$ represents the individual component of the TCL for the i-th sample, which activates the gradient update only if $\tilde{L}_i > 0$. This expression highlights that the gradient directs the update to increase the distance from the incorrect center and decrease the distance from the correct center, but only when the current configuration does not satisfy the margin requirement. The gradient with respect to class center c_j adjusts the position of the center c_j based on the aggregate of vectors pointing towards or away from c_j , weighted by whether the sample contributes to the TCL being positive. The update aims to optimize the position of c_j to better represent its class while maximizing separation from closely competing classes.

By incorporating centers into the loss function, TCL helps stabilize and guide the learning process, reducing the dependence on the specific selection of triplets. Additionally, TCL reduces reliance on the direct computation of pairwise distances among all possible triplets, which can be more scalable and computationally efficient. The center-based approach simplifies the need to calculate and compare vast numbers of triplet combinations, making it more suitable for large-scale applications.

In the work of He et al. (2018b), TCL can be combined with softmax loss for enhanced discrimination:

$$L_{total} = \lambda L_{TCL} + L_{softmax} \tag{1}$$

where softmax loss serves as a stabilizer, especially when centers c_y are initialized randomly and updated based on mini-batches. It helps guide the learning towards effective class centers, while TCL focuses more on maintaining correct relative distances.

3.3 Evaluation Metrics

The storage requirements for deep hashing algorithms depend solely on the length of the hash codes generated (Luo et al. (2023)). Since the codes are binary, each bit represents a piece of information about the data. To ensure a fair comparison of different hashing algorithms, the length of these hash codes is typically kept constant (Zhao et al. (2017)). The compactness of these codes allows for efficient use of memory. The search efficiency of these algorithms is largely influenced by the architecture of the neural networks used to generate the hash codes. The efficiency is often measured in terms of the average search time it takes for a query to return results. Complex neural architectures might provide more accurate hash codes but could also increase search time.

Accuracy In our experiments, we use accuracy as a metric to evaluate the performance of KNN on image embeddings and hashed image embeddings. It measures the proportion of test image embeddings for which the model's predicted label matches the correct label (Sammut and Webb (2011)). It is calculated as:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

In more formal terms:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP (True Positive) denotes the number of correctly predicted positive instances, TN (True Negative) is the number of correctly predicted negative instances, FP (False Positives) is the number of incorrectly predicted positive instances, and FN (False Negatives) is the number of incorrectly predicted negative instances.

Confusion matrix The confusion matrix provides a comprehensive summary of the prediction results by comparing the actual labels with the predicted labels for each class (Sammut and Webb, 2011). A confusion matrix is a square matrix that breaks down the performance of a classification model by showing the counts of true positive, true negative, false positive, and false negative predictions for each class:

Actual	Predicted Positive	Predicted Negative
Positive	TP	FN
Negative	FP	TN

Table 2: Confusion Matrix.

The confusion matrix is particularly valuable for understanding the performance of a classifier in situations where the classes are imbalanced or we need to distinguish between different types of errors (e.g., false positives versus false negatives). By analyzing the confusion matrix, we can gain deeper insights into how the KNN model performs across different classes.

4 Experiments and Results

4.1 Datasets

CIFAR-10 The CIFAR-10 dataset is a labeled subset of the 80 million tiny images dataset, curated by Krizhevsky et al. (2009). CIFAR-10 consists of 60,000 32 × 32 color images in classes, with 6,000 images per class. The dataset is divided into 50,000 training images and 10,000 test images. The classes are mutually exclusive and include categories such as airplane, automobile, bird, cat, horse, ship and truck. The training and test sets are predefined, ensuring that the evaluation of models is standardized and that the results are comparable across different studies. The images in CIFAR-10 are taken from a larger dataset called the "80 million tiny images" dataset, created by web scraping in 2008 (Torralba et al. (2008)). These images were collected from various online sources intended to represent a broad range of real-world conditions. The images were manually reviewed for quality and relevance and were labeled with the assistance of a human workforce. The selection process aimed to balance the dataset across the classes. The CIFAR-10 dataset has been instrumental in many state-of-the-art advancements in image classification. Research papers often report accuracy measures on CIFAR-10 to demonstrate the effectiveness of new models or techniques.

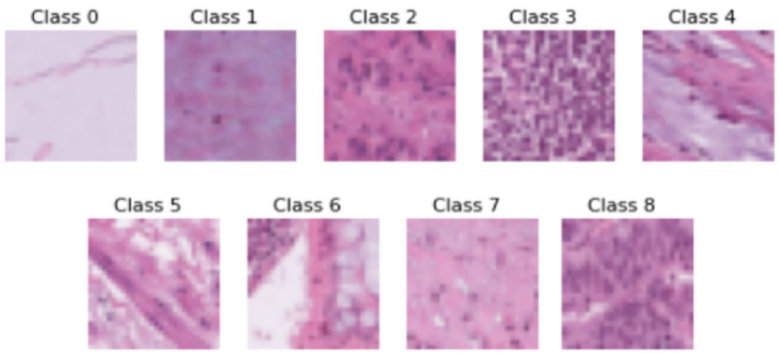

Figure 17: Diffenrent classes of PathMNIST dataset (Halder et al. (2024)).

PathMNIST PathMNIST, part of the larger MedMNIST v2 dataset, is designed to serve as a benchmark for machine learning models in the field of biomedical image classification (Yang et al. (2023)). PathMNIST focuses on pathology images, specifically on colorectal cancer histology slides. The dataset contains 107,180 images derived from hematoxylin and eosin (H & E) stained slides, which are typical in medical histology for distinguishing cellular components (Kather et al. (2019)). The images are sourced from the NCT-CRC-HE-100K and CRC-VAL-HE-7K datasets, which provide non-overlapping image patches from colorectal cancer histology slides and a validation set from a different clinical center, respectively (Kather et al. (2018)). PathMNIST includes nine distinct tissue types, categorized for multi-class

classification challenges. Images originally of size 3×224×224 pixels are resized to 3×28×28 to create a uniform, small-scale dataset suitable for quick processing and analysis. The dataset is split into training (89,996), validation (10,004), and test (7,180) sets with a standard ratio, promoting fair and consistent model evaluations across different machine learning approaches. The dataset, along with other MedMNIST datasets, is publicly available for research and educational use, encouraging widespread adoption and innovation in the biomedical imaging domain.

4.2 System Setup Parameters for Experiments

Parameters for TCL As indicated by loss function of TCL in Eq.1, the margin parameter m and λ may affect the final combination of losses. We adopt $m = 5$ and $\lambda = 0.01$ as these hyper-parameters achieve the best results shown in He et al. (2018b).

k-value for KNN approach

- If not stated otherwise, the experiments on CIFAR-10 dataset were conducted with hyperparameter $k = 7$. Euclidean Distance was used on the image embeddings, while Hamming Distance was used on hashed image embeddings.
- For PathMNIST dataset, we conduct an experiment on image embeddings to fine-tune the hyperparameter k, which is then used on infer hashed image embeddings. Based on the results in Fig.3, we adopt $k = 7$ in the following experiments on PathMNIST dataset.

Table 3: Results of KNN with different k-values on PathMNIST image embeddings.

k-value	3	5	7	10	15
Accuracy	0.75	0.76	0.77	0.77	0.77

Hash bits Output dimension of the hashed image embeddings are 12/24/36/48.

4.3 Results and Analysis

4.3.1 General Results of KNN on Image Embedding

The achieved accuracies of KNN on CIFAR-10 and PathMNIST image embeddings are summarized in table.4.3.1.

Image Encoder	Dataset	Accuracy
ViT-B/16	CIFAR-10	98.7
	PathMNIST	75.6

Table 4: General Results of KNN on Image Embedding

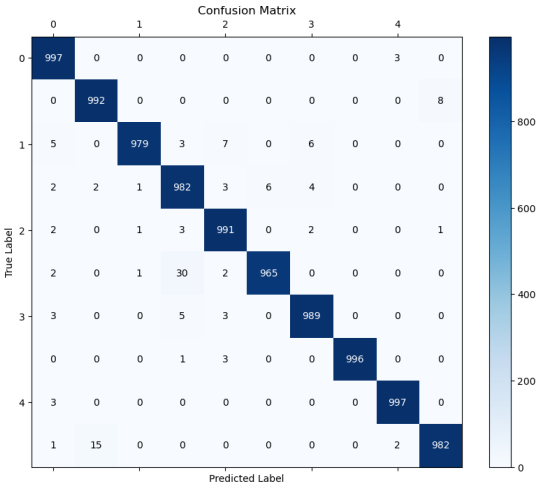

Figure 18: Confusion Matrix of KNN on CIFAR-10 dataset.

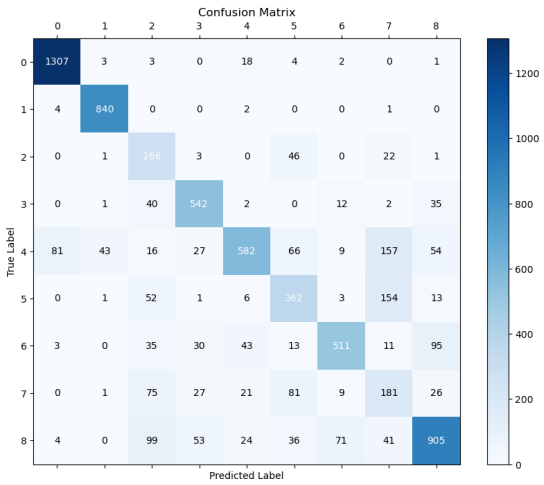

Figure 19: Confusion Matrix of KNN on PathMNIST dataset.

4.3.2 Results Comparison for different Supervised Deep Hashing Methods

Results for Pariwise Methods This section provides a detailed comparison and analysis of pairwise hashing methods evaluated on the CIFAR-10 and PathMNIST datasets. The accuracy results are tabulated in 4.3.2 for different hash code lengths: 12, 24, 36, and 48 bits.

Observations:

- The performance across all methods is generally lower on PathMNIST compared to CIFAR-10, reflecting the dataset's complexity and the challenges in medical image hashing.
- There is a noticeable upward trend in performance as the bit length increases, indicating that higher bit lengths capture more relevant information for accurate hashing.
- Label information helps to increase the performance of deep hashing. This point can be shown from the fact that DSDH outperforms DPSH (Luo et al. (2023)).

	CIFAR-10				PathMNIST			
Method	12bits	24bits	36bits	48bits	12bits	24bits	36bits	48bits
DSH	88.78	88.93	88.99	91.03	51.54	52.89	51.62	57.20
CNNH	97.88	98.05	98.08	98.14	23.40	26.64	26.72	27.89
DPSH	88.74	91.43	92.28	95.56	66.74	67.91	69.02	69.56
DHN	87.13	93.07	93.96	95.19	69.09	69.64	69.98	69.80
HashNet	97.74	98.81	97.90	97.98	31.31	33.24	47.44	49.79
DSDH	97.73	98.05	98.04	98.03	68.19	69.33	69.81	70.15

Table 5: Accuracy for Pairwise Methods.

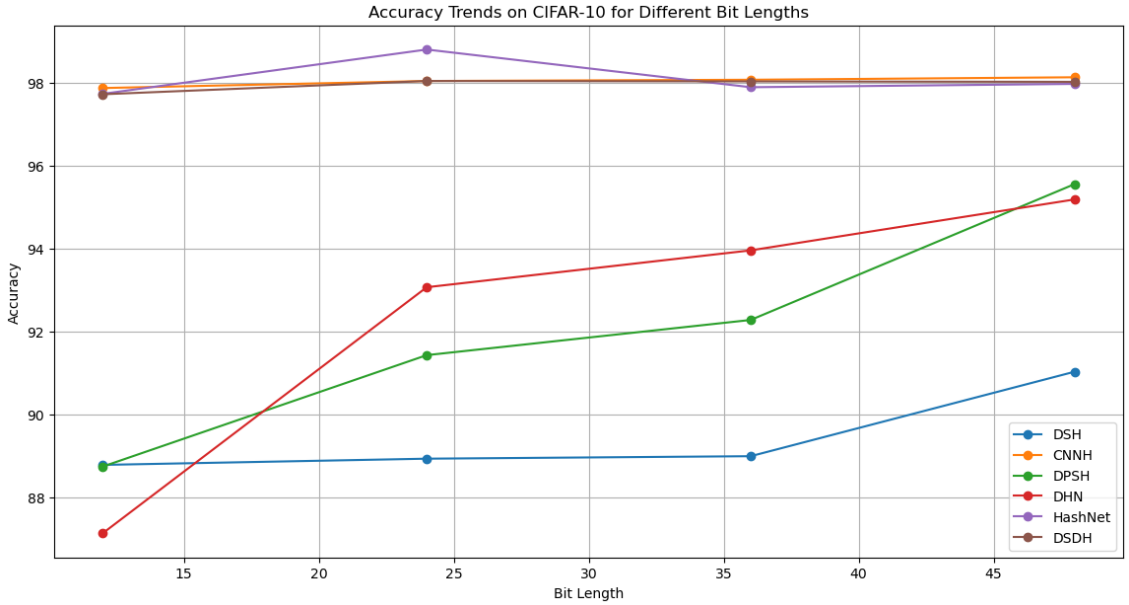

Figure 20: Accuracy on CIFAR-10 for different bit lengths compared by different pairwise methods.

- Methods like CNNH perform exceptionally well on CIFAR-10 but falter on PathMNIST, highlighting the need for dataset-specific adaptations or considerations in method design.
- DSDH demonstrate a degree of robustness across both datasets, suggesting that these methods have potential applications in diverse scenarios beyond standard image datasets.

Following are confusion matrices of classification results of KNN on hashed image embeddings.

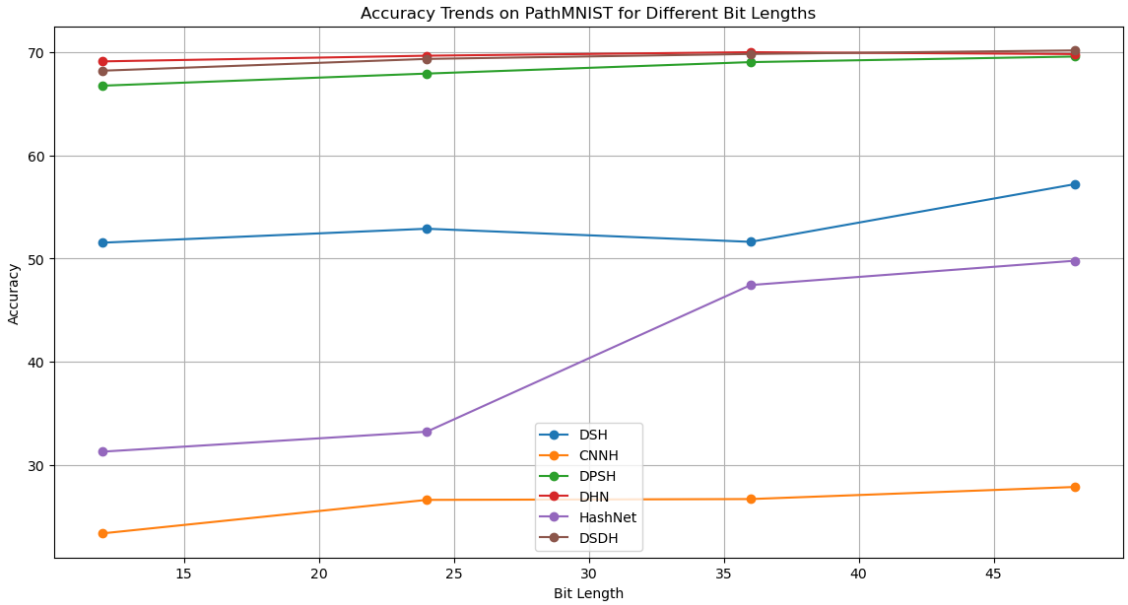

Figure 21: Accuracy on PathMNIST for different bit lengths compared by different pairwise methods.

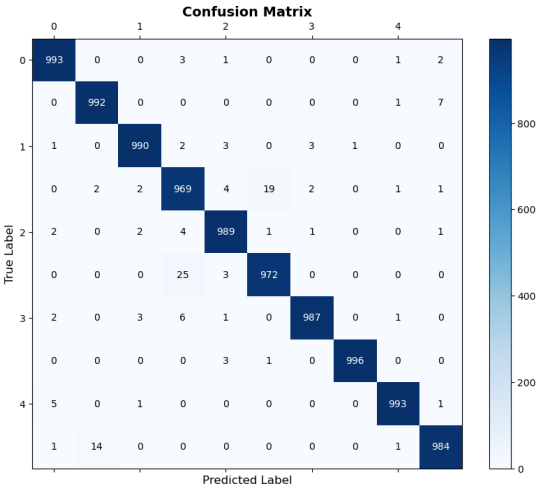

Figure 22: Confusion matrix of KNN results on hashed CIFAR-10 image embeddings (48 bits) using HashNet.

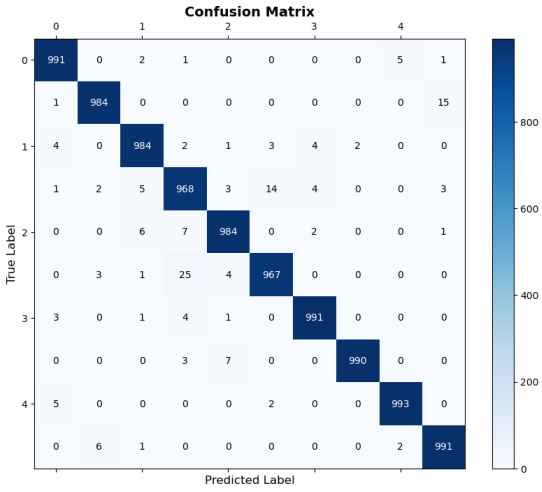

Figure 23: Confusion matrix of KNN results on hashed CIFAR-10 image embeddings (48 bits) using CNNH.

4.3.3 Results for Triplet Methods

Observations:

- Triplet methods generally perform better on both datasets and all bit length.

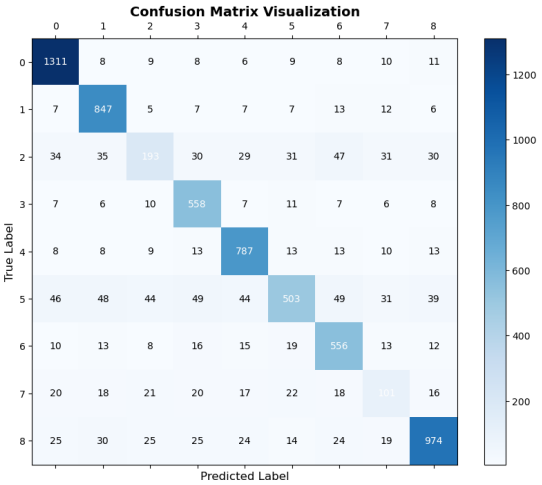

Figure 24: Confusion matrix of KNN results on hashed PathMNIST image embeddings (48 bits) using DSDH.

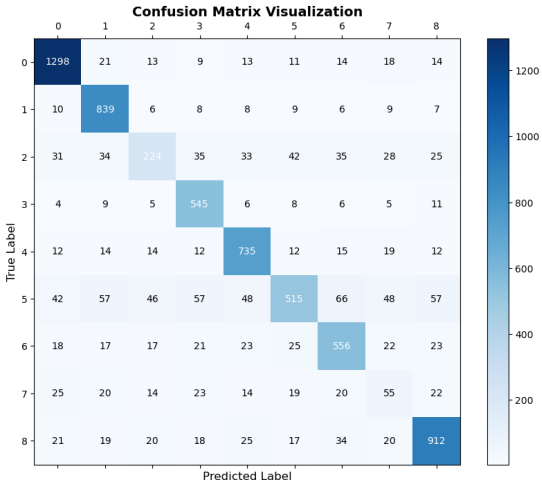

Figure 25: Confusion matrix of KNN results on hashed PathMNIST image embeddings (48 bits) using DHN.

	CIFAR-10				PathMNIST			
Method	12bits	24bits	36bits	48bits	12bits	24bits	36bits	48bits
DNNH	98.21	98.32	98.31	98.29	72.32	72.85	72.86	72.92
DSHGAN	98.65	98.68	98.71	98.73	75.75	75.85	76.77	77.89
TCL+softmax loss	98.51	98.67	98.74	98.78	75.57	75.67	75.50	76.55

Table 6: Accuracy for Triplet Methods.

- KNN can perform better on hashed image embeddings than original image embeddings. Two reasons may play important roles: (1). Hashed embeddings typically reduce the dimensionality of the original data, which may lead to noise reduction and overfitting reduction as hashing move redundant and less informative features and lower-dimensional data helps in mitigating overfitting. (2). Triplet methods may enhance discriminative power as learning-based hashing techniques aim to embed higher semantic information into lower-dimensional spaces.

- The consistency across bit sizes for each method indicates robustness in their hashing techniques.

- DSHGAN and TCL with softmax loss generally offer superior performance over DNNH, particularly on PathMNIST. This suggests that methods incorporating generative components or composite methods may better capture the complex patterns present in specialized datasets.

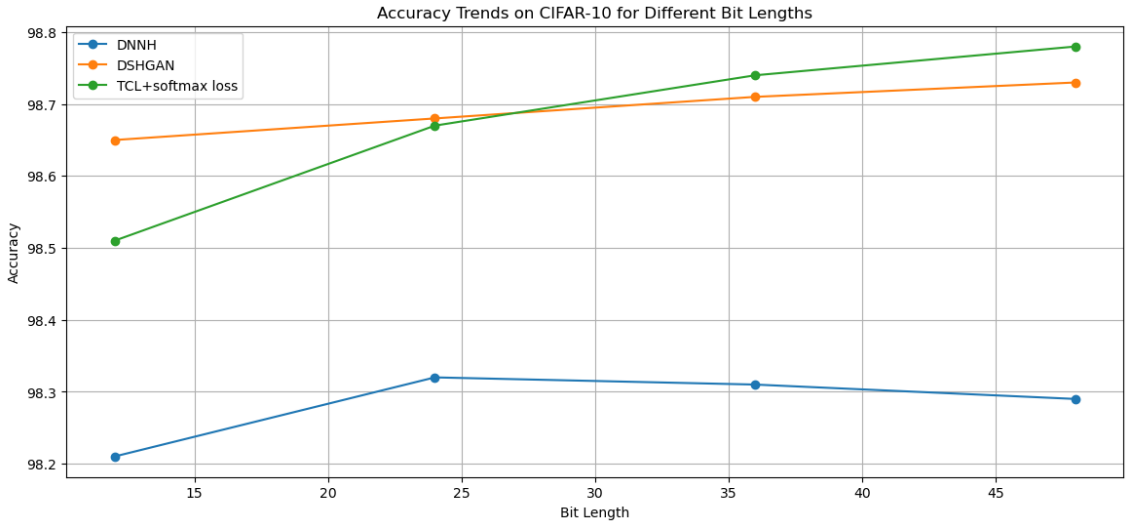

Figure 26: Accuracy on PathMNIST for different bit lengths compared by different triplet methods.

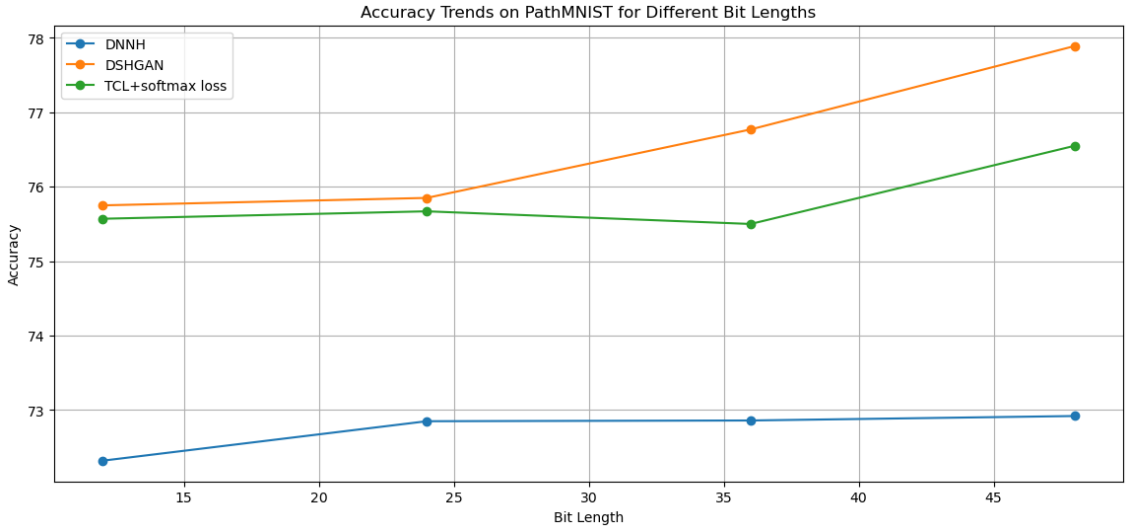

Figure 27: Accuracy on PathMNIST for different bit lengths compared by different triplet methods.

Following are confusion matrices of computing DSH-GAN and TCL+softmax loss on PathMNIST datasets.

4.3.4 Computation Efficiency of TCL over Triplet Methods

Traditional triplet loss requires comparisons across triplets of samples (anchor, positive, negative). For each training sample, finding and evaluating suitable triplets

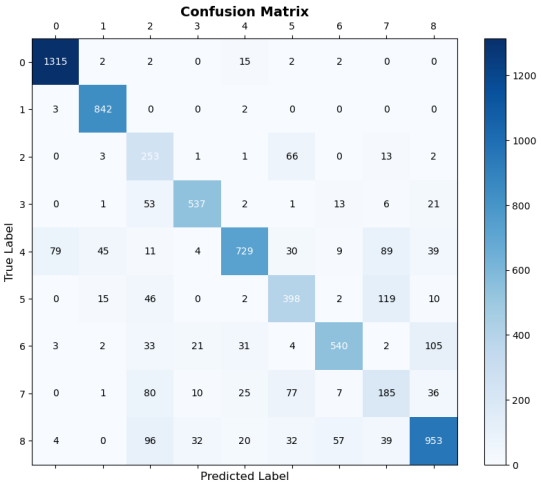

Figure 28: Confusion matrix of KNN results on hashed PathMNIST image embeddings (48 bits) using DSHGAN.

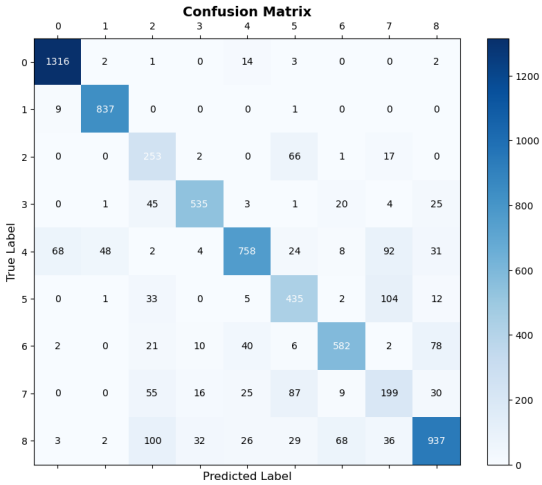

Figure 29: Confusion matrix of KNN results on hashed PathMNIST image embeddings (48 bits) using TCL.

involves significant computational overhead, particularly in terms of distance calculations. TCL, by focusing on distances to fixed class centers rather than pairwise or triplet comparisons, significantly reduces the number of distance computations required per training instance. In TCL, each class has a single center that is updated during training. The distance from each sample to its class center (and potentially to the nearest other class center) is typically all that is needed to compute the loss. This is usually less computationally intensive than evaluating distances across multiple potential triplets. Hence, want to infer that TCL is computationally more efficient than other triplet methods.

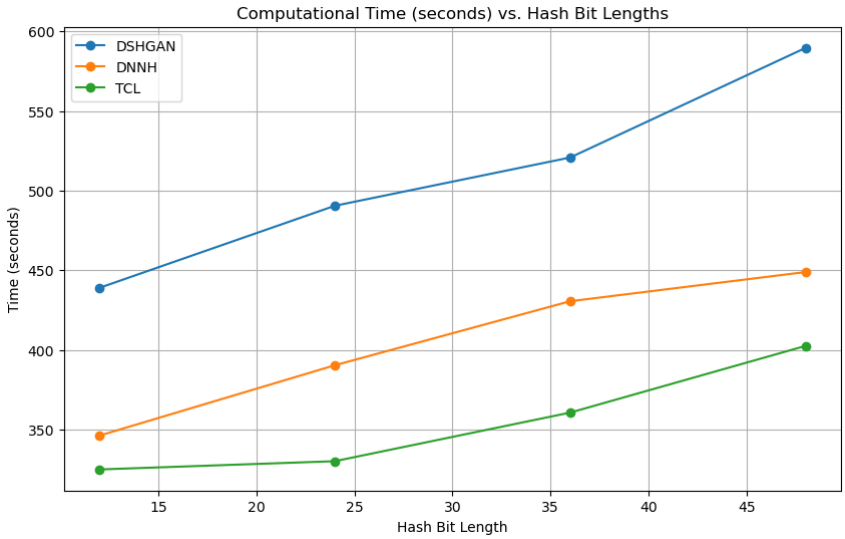

Figure 30: Comparison of computation cost between triplet methods (in seconds).

We can observe that:

- The trend in computational cost generally exhibits an increase when increasing hash bits.
- TCL is generally more computation efficient than other two triplet methods.

5 Discussion

In this work, empirical experiments are conducted on nine state-of-art deep supervised hashing methods on two benchmark datasets. Experiment setup and loss function construction are inspired by the framework in each proposed paper of each method and the overall framework in Luo et al. (2023). Sources for all datasets are identical. Image embeddings produced by ViT-B/16 serve as input to the deep hashing methods. This section delves into the critical analysis of the performance of various deep hashing methods applied to CIFAR-10 and PathMNIST datasets, focusing on the influence of hash bit lengths on accuracy.

The hashing methods generally achieved high accuracy on CIFAR-10 dataset, particularly notable in CNNH and HashNet. CNNH's approach starts with generating approximate binary codes using a coordinate descent strategy which optimizes the similarity matrix representation. This helps in forming a preliminary structure of the data in a binary format which is crucial for capturing essential features in images. In the second step, CNNH refines these binary codes using a deep CNN, further training the network to align these hash codes with the actual labels of the images. This method leverages the power of deep learning to enhance the discriminative ability of the hash codes. By incorporating class labels into the final training of the CNN, CNNH directly ties the hash codes to the class-discriminative features of the data, making it highly effective for datasets like CIFAR-10 which have distinct and well-separated classes. HashNet addresses the imbalance in training data (more similar than dissimilar pairs) by using Weighted Maximum Likelihood (WML) loss. This weighting ensures that each pair, whether similar or dissimilar, contributes appropriately to the learning process, preventing the model from bias towards more frequent pair types.

However, both CNNH and HashNet may struggle with medical datasets because their method of generating and refining hash codes might not capture the intricate, less visually obvious patterns present in medical images. Medical images can also suffer from issues like label imbalance and fewer examples per class, complicating the training process for methods that heavily rely on balanced data and clear class separation. Both methods might require modifications to better handle the nuances of medical imaging. For instance, tuning the parameters of HashNet's adaptive sigmoid or adjusting the CNN architecture in CNNH to focus on features relevant to medical images could potentially improve performance.

Label information helps to increase the performance of deep hashing as DSDH outperforms DPSH on both datasets. DSDH incorporates label information through

linear regression. Linear regression loss directly ties the hash codes to the actual labels of the data. And the inclusion of an L2 regularization term on the weights W (used in the linear regression) encourages the model to find a simpler, more generalizable mapping from hash codes to labels, preventing overfitting to the training data. DPSH, while also utilizing a deep learning framework for hashing, primarily relies on similarity information and a quantization loss to shape its hash codes. Thus, the integration of label information through linear regression helps in aligning the hash codes more closely with the class-specific features, thereby enhancing the discriminative power of the generated hashes.

The triplet methods generally perform better than pairwise methods on both two datasets. Method like DSHGAN that incorporate GANs might incur higher computational costs but could offer scalability benefits through generative approach. TCL, while achieving an ideal performance, is also computationally balanced. By using class centers as reference points, TCL provides direct feedback on how well the embeddings are aligning with the conceptual centers of the classes. It requires fewer pairwise distance calculations per sample, as distances are calculated relative to class centers and not every possible negative sample. This reduction in the number of required operations can lead to faster training times and lower computational overhead. Incorporating the center loss component also simplifies the training regime by reducing the dependency on the hard triplet mining process, which can be challenging to tune and implement effectively. Additionally, TCL is versatile and can be applied across a variety of domains where deep learning is used for feature extraction and similarity learning, as the case we use it in this work in domain of deep hashing.

6 Conclusion

This bachelor thesis make a novel evaluation between various deep supervised hashing methods.

Firstly, the image embeddings are generated though ViT-B/16 image encoder. KNN is applied on these image embeddings to evaluate accuracy as a baseline metric. Secondly, a series of experiments were conducted to transfer the image embeddings to hashed image embeddings through various deep supervised hashing methods, mainly including pairwise and triplet methods. We again retrieve the accuracy using KNN on these hashed image embeddings to make a comparison between the baseline accuracy. KNN's reliance on distance or similarity metrics makes it a suitable method to test how well the hashing preserves the necessary information for tasks like classification. Hence, the goals of privacy preservation and reduction of computational costs are achieved as converted binary codes are designed to not only preserve semantic similarity but also reduce the possibility of recovering any original image data. The binary nature and the compactness of these hashes make it computationally infeasible to reverse-engineer the original embeddings or the raw images, thereby enhancing privacy.

The key findings of this work as following:

- Label information helps increase the performance of deep hashing.
- Methods incorporating adaptive weighting mechanisms are more adept at handling datasets with class imbalance.
- Adopting TCL in deep hashing is an innovative way to balance trade-off between high performance and computational efficiency.
- Computation costs generally increase as hash bits increase.

Future Research Future research could focus on developing new hashing algorithms that offer even better trade-offs between data compression, privacy preservation, and accuracy. Algorithms that incorporate recent advancements in neural architectures or unsupervised learning techniques could provide new insights and capabilities. Combining different types of hashing techniques, such as integrating deep unsupervised or semi-supervised hashing methods, might yield models that leverage the strengths of both cluster approach and deep learning-based approaches.

As datasets continue to grow, optimizing hashing algorithms for scalability and efficiency becomes crucial. Research could also focus on algorithms that reduce computational complexity and enhance processing speed without compromising hashing quality.

Developing methods that support real-time hashing of image data would be beneficial for applications in security and real-time surveillance, where quick processing is critical. Tailoring hashing methods to specific application domains such as medical imaging or autonomous driving could be explored. Each domain comes with unique challenges that might be better addressed by specialized hashing techniques. However, challenges may also remain in explain-ability in AI, as people may not accept a black-box when it comes to safety-related issue.

A Appendix

If needed for supplementary material, such as detailed description of data collection, tables, or figures.

Symbol	Description
$x_i(X)$	input images (in matrix form)
$b_i(B)$	output hash codes (in matrix form)
$h_i(H)$	network outputs (in matrix form)
$y_i(Y)$	one-hot image labels (in matrix form)
$\Psi(\cdot)$	hashing network
N	the number of input images
L	hash code length
$\mathcal{E}$	a set of pair items
s_{ij}^o	the similarity of item pair in the input space
s_{ij}^h	the similarity of item pair in the Hamming space
d_{ij}^o	the distance of item pair in the input space
d_{ij}^h	the distance of item pair in the Hamming space
ϵ	margin threshold parameter
W	weight parameter matrix
Θ	set of neutral parameters

Table 7: Summary of Symbols and Notation in Section.3.

Bibliography

Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

Vassilis Athitsos, Marios Hadjieleftheriou, George Kollios, and Stan Sclaroff. Query-sensitive embeddings. *ACM Transactions on Database Systems (TODS)*, 32(2): 8–es, 2007.

Fatih Cakir, Kun He, Sarah Adel Bargal, and Stan Sclaroff. Hashing with mutual information. *IEEE transactions on pattern analysis and machine intelligence*, 41 (10):2424–2437, 2019.

Yue Cao, Mingsheng Long, Jianmin Wang, and Shichen Liu. Deep visual-semantic quantization for efficient image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1328–1337, 2017a.

Yue Cao, Mingsheng Long, Bin Liu, and Jianmin Wang. Deep cauchy hashing for hamming space retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1229–1237, 2018.

Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. Hashnet: Deep learning to hash by continuation. In *Proceedings of the IEEE international conference on computer vision*, pages 5608–5617, 2017b.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

Chi-hau Chen et al. *Signal and image processing for remote sensing*. CRC/Taylor & Francis, 2007.

Lianhua Chi and Xingquan Zhu. Hashing techniques: A survey and taxonomy. *ACM Computing Surveys (Csur)*, 50(1):1–36, 2017.

Yoo Jin Choi, Mostafa El-Khamy, and Jungwon Lee. Method and apparatus for neural network quantization, April 19 2018. US Patent App. 15/697,035.

Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, pages 1–9, 2009.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

Giuseppe De Giacomo, Alejandro Catala, and Bistra Dilkina. *ECAI 2020: 24th European Conference on Artificial Intelligence, 29 August–8 September 2020, Santiago de Compostela, Spain–Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325. IOS Press, 2020.

Francesco Di Salvo, David Tafler, Sebastian Doerrich, and Christian Ledig. Privacy-preserving datasets by capturing feature distributions with conditional vaes. *arXiv preprint arXiv:2408.00639*, 2024.

Thanh-Toan Do, Anh-Dzung Doan, and Ngai-Man Cheung. Learning to hash with binary deep neural network. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part V 14*, pages 219–234. Springer, 2016.

Khoa D Doan, Jianwen Xie, Yaxuan Zhu, Yang Zhao, and Ping Li. Coophash: Cooperative learning of multipurpose descriptor and contrastive pair generator via variational mcmc teaching for supervised image hashing. *arXiv preprint arXiv:2210.04288*, 2022.

Sebastian Doerrich, Tobias Archut, Francesco Di Salvo, and Christian Ledig. Integrating knn with foundation models for adaptable and privacy-aware image classification. *arXiv preprint arXiv:2402.12500*, 2024.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Jiali Duan, C-C Jay Kuo, et al. Bridging gap between image pixels and semantics via supervision: a survey. *APSIPA Transactions on Signal and Information Processing*, 11(1), 2022.

Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2475–2483, 2015.

Lorenzo Federici, Alessandro Zavoli, and Guido Colasurdo. Evolutionary optimization of multirendezvous impulsive trajectories. *International Journal of Aerospace Engineering*, 2021(1):9921555, 2021.

Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

Yunchao Gong. Large-scale image retrieval using similarity preserving binary codes. 2014.

Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 35(12):2916–2929, 2012.

Albert Gordo, Florent Perronnin, Yunchao Gong, and Svetlana Lazebnik. Asymmetric distances for binary embeddings. *IEEE transactions on pattern analysis and machine intelligence*, 36(1):33–47, 2013.

Kristen Grauman and Trevor Darrell. Pyramid match hashing: Sub-linear time indexing over partial correspondences. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

Arindam Halder, Sanghita Gharami, Priyangshu Sadhu, Pawan Kumar Singh, Marcin Woźniak, and Muhammad Fazal Ijaz. Implementing vision transformer for classifying 2d biomedical images. *Scientific Reports*, 14(1):12567, 2024.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Kun He, Fatih Cakir, Sarah Adel Bargal, and Stan Sclaroff. Hashing as tie-aware learning to rank. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4023–4032, 2018a.

Xinwei He, Yang Zhou, Zhichao Zhou, Song Bai, and Xiang Bai. Triplet-center loss for multi-view 3d object retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1945–1954, 2018b.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

Licheng Jiao, Zhongjian Huang, Xiaoqiang Lu, Xu Liu, Yuting Yang, Jiaxuan Zhao, Jinyue Zhang, Biao Hou, Shuyuan Yang, Fang Liu, et al. Brain-inspired remote sensing foundation models and open problems: A comprehensive survey. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2023.

Jakob Nikolas Kather, Niels Halama, and Alexander Marx. 100,000 histological images of human colorectal cancer and healthy tissue (v0.1). *Zenodo*, 2018.

Jakob Nikolas Kather, Johannes Krisam, Pornpimol Charoentong, Tom Luedde, Esther Herpel, Cleo-Aron Weis, Timo Gaiser, Alexander Marx, Nektarios A Valous, Dyke Ferber, et al. Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study. *PLoS medicine*, 16(1): e1002730, 2019.

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6): 84–90, 2017.

Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *2009 IEEE 12th international conference on computer vision*, pages 2130–2137. IEEE, 2009.

Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3270–3278, 2015.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

Qi Li, Zhenan Sun, Ran He, and Tieniu Tan. Deep supervised discrete hashing. *Advances in neural information processing systems*, 30, 2017.

Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. Feature learning based deep supervised hashing with pairwise labels. *arXiv preprint arXiv:1511.03855*, 2015.

Kevin Lin, Huei-Fang Yang, Jen-Hao Hsiao, and Chu-Song Chen. Deep learning of binary hash codes for fast image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 27–35, 2015.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Bin Liu, Yue Cao, Mingsheng Long, Jianmin Wang, and Jingdong Wang. Deep triplet quantization. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 755–763, 2018a.

Bin Liu, Yue Cao, Mingsheng Long, Jianmin Wang, and Jingdong Wang. Deep triplet quantization. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 755–763, 2018b.

Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. Deep supervised hashing for fast image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2064–2072, 2016.

Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *2012 IEEE conference on computer vision and pattern recognition*, pages 2074–2081. IEEE, 2012.

Xingbo Liu, Xiushan Nie, and Yilong Yin. Mutual linear regression-based discrete hashing. *arXiv preprint arXiv:1904.00744*, 2019.

Xiao Luo, Haixin Wang, Daqing Wu, Chong Chen, Minghua Deng, Jianqiang Huang, and Xian-Sheng Hua. A survey on deep hashing methods. *ACM Transactions on Knowledge Discovery from Data*, 17(1):1–50, 2023.

Dinesh P Mehta and Sartaj Sahni. *Handbook of data structures and applications*. Chapman and Hall/CRC, 2004.

Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

Avinash Navlani. Knn classification using scikit-learn. *Data Camp, August*, 2018.

Mohammad Norouzi, David J Fleet, and Russ R Salakhutdinov. Hamming distance metric learning. *Advances in neural information processing systems*, 25, 2012.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Gerhard Paaß and Sven Giesselbach. *Foundation Models for Natural Language Processing: Pre-trained Language Models Integrating Media*. Springer Nature, 2023.

Zhaofan Qiu, Yingwei Pan, Ting Yao, and Tao Mei. Deep semantic hashing with generative adversarial networks. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 225–234, 2017.

Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. *Advances in neural information processing systems*, 22, 2009.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

Adil Redaoui, Amina Belalia, and Kamel Belloulata. Deep supervised hashing by fusing multiscale deep features for image retrieval. *Information*, 15(3):143, 2024.

Derek JS Robinson. *An introduction to abstract algebra*. Walter de Gruyter, 2003.

Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE transactions on neural networks and learning systems*, 28(11): 2660–2673, 2016.

Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

Iqbal H Sarker. Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2(6):420, 2021.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Avantika Singh and Shaifu Gupta. Learning to hash: A comprehensive survey of deep learning-based hashing methods. *Knowledge and Information Systems*, 64 (10):2565–2597, 2022.

Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Bill Waggener and William N Waggener. *Pulse code modulation techniques*. Springer Science & Business Media, 1995.

Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, 40(4): 769–790, 2017.

Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data—a survey. *Proceedings of the IEEE*, 104(1):34–57, 2015.

Xunguang Wang, Zheng Zhang, Guangming Lu, and Yong Xu. Targeted attack and defense for deep hashing. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2298–2302, 2021.

Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. *Advances in neural information processing systems*, 21, 2008.

Dayan Wu, Qi Dai, Jing Liu, Bo Li, and Weiping Wang. Deep incremental hashing network for efficient image retrieval. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9069–9077, 2019.

Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28, 2014.

Xiang Xu, Xiaofang Wang, and Kris M Kitani. Error correction maximization for deep image hashing. *arXiv preprint arXiv:1808.01942*, 2018.

BS Yang, ZH Zhou, Z Gong, ML Zhang, and SJ Huang. Advances in knowledge discovery and data mining. In *Proceedings*, volume 405. Springer, 2014.

Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):41, 2023.

Zhan Yang, Osolo Ian Raymond, Wuqing Sun, and Jun Long. Deep attention-guided hashing. *IEEE Access*, 7:11209–11221, 2019.

Li Yuan, Tao Wang, Xiaopeng Zhang, Francis EH Tay, Zequn Jie, Wei Liu, and Jiashi Feng. Central similarity quantization for efficient image and video retrieval. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3083–3092, 2020.

Shifeng Zhang, Jianmin Li, and Bo Zhang. Semantic cluster unary loss for efficient deep hashing. *IEEE Transactions on Image Processing*, 28(6):2908–2920, 2019.

Juan-Juan Zhao, Ling Pan, Peng-Fei Zhao, and Xiao-Xian Tang. Medical sign recognition of lung nodules based on image retrieval with semantic features and supervised hashing. *Journal of Computer Science and Technology*, 32:457–469, 2017.

Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 30, 2016.

Declaration of Authorship

Ich erkläre hiermit gemäß §9 Abs. 12 APO, dass ich die vorstehende Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Des Weiteren erkläre ich, dass die digitale Fassung der gedruckten Ausfertigung der Abschlussarbeit ausnahmslos in Inhalt und Wortlaut entspricht und zur Kenntnis genommen wurde, dass diese digitale Fassung einer durch Software unterstützten, anonymisierten Prüfung auf Plagiate unterzogen werden kann.

Place, Date	Signature