A Deep Reinforcement Learning-based Image Hashing Technique

by

MITRA REZAEI

A thesis submitted to the
Department of Computer Science
in conformity with the requirements for
the degree of Master of Science

Bishop's University
Canada
August 2023

Abstract

In recent years, deep hashing algorithms have gained a lot of attention in image retrieval because of their high efficiency, low memory consumption, and high retrieval accuracy. Although a binary hash code with a longer length is technically more accurate, it may contain redundant binary values, which is a major problem with most hashing algorithms. This thesis proposes a Deep Reinforcement Learning-Based Image Hashing Technique, which generates a binary hash code database using CNN in the first step, and then adaptively selects the most informative bits from each binary hash code in the second step, resulting in a more accurate binary hash code database. In our approach, Markov Decision Process (MDP) is used to model an efficient binary value selection process, and Reinforcement Learning is used to solve it. The most informative bits are identified by optimizing mean Average Precision (mAP) during training. Our method is shown to produce highly efficient compact binary hash codes with different lengths and perform better retrieval than state-of-the-art methods on three public datasets, CIFAR-10, NUS-WIDE, and MS-COCO.

Keywords: Deep Learning, Markov Decision Process, Reinforcement Learning, PPO, CNN, Image Hashing

Acknowledgment

I would like to acknowledge all the people without whom this thesis would not have been fulfilled. First place, I would like to express my sincere gratitude to my supervisors, Dr. Madjid Allili and Dr. Mohammed Ayoub Alaoui Mhamdi for accepting to be my supervisor and for always encouraging me to work hard. Thanks to your time, effort, and understanding, my studies were a success.
I would also like to thank my family, especially my husband, Mehdi, for their encouragement and support, without which I could not have completed my master's degree.
Furthermore, I would like to thank all of the faculty members in the Computer Science department at Bishop's University for their kindness and support. In every education-related question I had, they always responded promptly.
In closing, I would like to express my gratitude to the Mitacs Foundation for funding my internship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Large data sets of high-dimensional raw data analysis are typically computationally and memory costly. Due to this, it is frequently essential to translate data into a lower-dimensional space as part of the preprocessing step while roughly maintaining crucial geometric aspects, such as pairwise L2 distance [1]. Binary embedding also known as binary hashing or binary representation, a technique to nonlinearly embed high dimensional sets $X \subseteq \mathbb{R}^n$ to the binary cube $\{1, -1\}^m$ with $m << n$, has developed to further decrease memory requirements and simplify the contents of the high dimensional data, such as images [2]. Image hashing is the method of employing binary hashing for image datasets, thus rather than directly comparing the images, it utilizes this hash instead [2]. As hashes are significantly smaller than images, the comparison is quicker and uses less storage. Image hashing has applications in image authentication and image retrieval [3], tamper detection [4], digital watermarking [5], copy detection [6], digital forensics [7], and reduced-reference image quality evaluation [8]. For example, people often use image editing tools, such as Photoshop and ACDsee software, to process photographs and save them in JPEG format with different filenames. Consequently, there may be several copies of an image on the computer. These copies have the same visual contents as the original image, but their digital representations are different from that of the original one. In this case, people can exploit the image hash function to efficiently search all similar versions (including the original one and its copies) of the image from a large-scale image database.

On the other hand, there is a vast variety of smart digital devices, internet connections, and people all over the world who can use these devices and services to create and distribute digital content that may be harmful to others. The advancement of digital forgery has posed a significant challenge to multimedia authentication and security [2]. Additional layers of prevention are required to protect against such crimes and to support Law Enforcement Agencies (LEAs). In addition to social awareness and cybersecurity, computer vision techniques such as image hashing can be employed to detect, stop, and respond to sophisticated crimes such as distributing fake images. In the fight against cybercrimes, such as the distribution and consumption of Child Sexual Abuse Material (CSAM), hotlines, law enforcement agencies, industry stakeholders, and other child protection organizations utilize image hashing technology to detect and remove CSAM [9].

Image hash function takes an image as input and maps it to a compact hash value such that perceptually similar images always map to similar values, and different values if the images are perceptually different.

In this thesis, we explore the utilization of deep reinforcement learning for image hashing, presenting a novel approach to decision-making. We frame the image hashing problem within the framework of a Markov Decision Process (MDP), which provides a formal representation of sequential decisions made by an agent. Our objective is to construct binary hash codes for images that effectively minimize the Hamming distance between images of the same class and maximize the distance between images of different classes. The reason for this is that it allows the detection and removal of known CSAM items without requiring an analyst to assess them again. Using hashing technology is critical because once CSAM exists online, it is often shared thousands of times. As a result, analysts and law enforcement are relieved of the repetitive review of the same content and the number of people watching the abuse is minimized. Furthermore, due to the rapid growth of images on the web, large scale image retrieval has been gaining a lot of attention. To retrieve images, many hashing methods have been proposed [10], [11]. Image hashes provide an automated way of deciding whether two media files are still perceptually identical, for example, whether one image is a copy of another, which was processed without changing its semantics. Unlike cryptographic hash functions such as Message Digest Algorithm (MD5), and Secure Hash Algorithm (SHA-1) the image hash function is not sensitive to digital representation of an image. It produces the same or very similar hash values for visually identical images no matter whether their representations are the same or not. In general, an image hash function must have two properties:

- **Perceptual Robustness**: A hash function should be robust against content-preserving operations, such as JPEG compression and denoising [12]. Image hash function should learn the same or similar hashes from those visually similar images, which may undergo common digital operations, such as image compression and geometric transformations. Thus, Images and their attacked versions should have the same hash.
- **Discriminative Capability**: The discriminative capability in image hashing refers to the ability of an image hash function to produce distinct hash values for different images. In other words, it measures how well the hash function can distinguish between two images and generate different hash codes for them. The higher the discriminative capability of an image hash function, the lower the probability of two different images having the same hash value. This property is essential for tasks such as image retrieval, duplicate detection, and content-based image recognition, where similar images need to be identified accurately [13].

It should be noted that there exists a constraining relationship between Perceptual Robustness and Discriminative Capability. The relationship between these two properties is that they are often trade-offs. An image hash function that is highly perceptually robust may have lower discriminative capability since it needs to generate the same hash value for similar images. Similarly, an image hash function with high discriminative capability may not be as perceptually robust since it needs to generate different hash values for even slightly different images. Therefore, designing an image hash function that strikes a balance between these two properties is crucial, depending on the intended application. For instance, if the primary application is image retrieval, the high discriminative capability is desirable, while if the application is image authentication, high perceptual robustness is more critical. By using binary codes instead of real-valued features, searching can be greatly sped up and memory costs reduced. For instance, during the detection of illegal material, if the hash of the original copy of that illegal image is stored in a database, the found media files are also hashed and these hashes are searched for in the database. If a very similar hash is found, illegal content has been detected.

1.2 Review of Hashing Theory

Hashing is a technique for finding nearest neighbors in large-scale datasets that involves embedding high-dimensional feature descriptors into a similarity-preserving Hamming space with a low dimension and a fixed length known as a Hash [11]. Cryptographic Hash Functions (CHFs) are mathematical operations that convert input data into a fixed-length string of bits, called a Hash value, and are used to verify the validity of data with varying levels of complexity and difficulty [9]. Cryptographic hashes add security features to traditional hashes, making it more difficult to decipher message contents or access recipient information [14]. The applications of CHFs namely, message authentication, user authentication, and secure matching provide advantages that are highly desirable in information security. CHFs have two characteristics:

- **Collision-free**: The output hash of a well-designed CHF is unique for every input, and collisions are minimized by using mathematical techniques.
- **Irreversibility**: Since hash functions are one-way procedures, it might be challenging to infer the input value of a hash function from its result.

Although CHFs are ideal for hashing passwords and PINs, they are not suitable for image processing or biometrics since perceptually identical photos or biometrics of the same user can differ at various times or locations. The pixel values of images or biometrics can be changed so that they remain perceptually the same but have a significantly different appearance. Nevertheless, we know that when utilizing CHFs, even the smallest change to the image results in an entirely different hash.

The reason is that hash functions used in CHFs are extremely sensitive to variations and thus incapable of providing effective hashes for images or biometrics. To hash images without being sensitive to small input changes, Image Hashing has been proposed as an alternative to CHFs. In other words, binary embedding or image hashing is a hash function that is robust to minor changes. Therefore, an image hash function should be resistant to visual manipulation and prevent bit-by-bit comparison, it should produce a different hash if the visual content differs [15].

The objective of image hashing is to convert high-dimensional feature vectors into low-dimensional hash codes, so that the hash codes of similar or near-similar items are as similar or as close as possible, while the hash codes of dissimilar things are as distinct as possible. Hash functions scramble data and convert it into a numerical value and no matter how long the input is, the output value is always of the same length. Message digest functions or hash algorithms are other names for hash functions. As was previously said, a hash function is essential in establishing the retrieval accuracy in applications of the hashing approach. Basically, the generic hashing procedure can be stated as $hash(x) \rightarrow z$, where the value of x refers to the pixel values of the image's feature vectors. The hash function $hash(.)$ can be summarized into two main types:

- Linear hash functions,
- Nonlinear hash functions.

The generalized linear hash function is illustrated as $sign(Wx + y) \rightarrow z$ where $W \in \mathbb{R}^{\mathrm{p \times d}}$ and $y \in \mathbb{R}^{P}$ represent the linear projection matrix and bias vector, respectively. Hard thresholding function $sign(x)$ equals 1 if $x \geq 0$, otherwise -1. However, in real applications, the linear hash function usually suffers from low discrimination power [16]. To overcome the low discrimination power of linear hash functions, researchers have developed nonlinear hash functions by applying kernel, spherical function, or boosting models to the original feature before binarization [16]. As opposed to linear hashes, nonlinear operations enhance feature expressiveness and are more suitable for data collected from complex real-world scenarios. Figure 1.1 gives a simple example of such a case. A kernel-based hash function is presented below without loss of generality:

$$sign\left(\sum_{i} W_i \varphi(c_i, x) + y\right) \rightarrow z \qquad (1.1)$$

where c_i denotes the randomly sampled data point or cluster center from the dataset. $\varphi(.)$ is the kernel function and W_i represents the weight matrix. In recent years, deep neural networks have been widely integrated into hashing frameworks, which can be considered as an advanced form of nonlinear hash functions with varied activation functions [16].

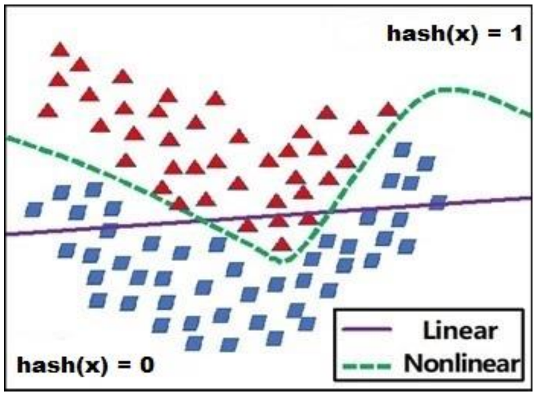

Figure 1.1: Linear and Non-Linear hash function

Image Hashing, which considers changes in the visual domain, is an efficient multimedia technology that facilitates content identification and comparison, ensuring reliable and secure multimedia management and retrieval processes. In an image hashing process, the input is an image, and the output is a compact binary representation based on the visual content of the input image [15]. Generally, the compact representation is called an image hash. An image hash is significantly lower in size than the actual image data since it just comprises the fundamental perceptual components of a picture. These perceptual components according to Robin Landa's book [17] include:

- **Shape**: The form or outline of objects in the picture.
- **Color**: The hue, saturation, and brightness of the picture.
- **Texture**: The surface quality or feel of the objects in the picture.
- **Contrast**: The difference between light and dark areas in the picture.
- **Depth**: The perceived distance between objects in the picture.
- **Composition**: The arrangement of objects and visual elements within the picture.
- **Movement**: The appearance of motion or activity in the picture.
- **Perspective**: The way that objects appear to change in size and position based on their distance from the viewer.

The image hash's small size speeds up search processes and reduces memory use. Note that the robust hash values are not only generated from images, but also from other multimedia formats, such as audio hashing [18] and video hashing [19]. In this thesis, we focus on image hashing since it is the most widely researched with the longest developing history. There are many other academic terms used for image hashing in literature: perceptual image hashing, robust image hashing, robust perceptual hash, soft hash, etc. [20]. We use the term Image Hashing in a broad sense to include all the foregoing technologies.

Given a data sample represented by a feature vector $x \in \mathbb{R}^{d}$, the goal of hashing techniques is to design an optimal hash function $hash(.)$ that projects x from the original high-dimensional space into compact binary space $z: z \in \{-1,1\}^{g} (\mathbb{R}^{d} \to \mathbb{R}^{g}\ and\ d \gg g)$ while keeping its true nearest neighbors as close as possible in the Hamming space [21]. In other words, similar data samples in the original feature space should be represented with similar binary codes in the Hamming distance, significantly enhancing retrieval efficiency while maintaining reasonable accuracy.

1.3 Similarity Search

The Internet era has created a huge amount of data and the explosion of data in all its forms, like image, text, audio, and video, has led to a multitude of problems concerning their authenticity and validity [16]. According to public statistics website, the average number of photos being shared every day on Flickr is about 1 million. To manage such massive data sources, conducting reliable and effective content-based similarity retrieval has received a lot of interest from both industry and academics.

Nearest Neighbor (NN) search, also known as proximity search, is often used in image retrieval applications, where the task is to find the closest matching image in a database to a query image [22]. According to Doan et al. [23] the NN algorithm can be used in this context by computing the similarity between the query image and the images in the database and selecting the image with the highest similarity as the closest match. NN has been successful in many classification and regression problems, including handwritten digits and satellite image scenes [23]. The cost of finding the exact nearest neighbor is prohibitively high in the case that the reference database is very large or that computing the distance between the query item and the database item is costly [22]. Furthermore, the proliferation of multimedia information such as images has made the security of media content an important research concern [24]. Thus, multimedia authentication techniques have emerged to verify content integrity and prevent forgery.

Being a non-parametric method, NN is often successful in classification situations where the decision boundary is very irregular [16]. Particularly, given a query feature vector $x_q \in \mathbb{R}^{\boldsymbol{d}}$, a gallery set consists of n feature vectors $X = [x_i]_{i=1}^{n} \in \mathbb{R}^{\boldsymbol{d}\times\boldsymbol{n}}$, d is the dimensionality, the NN search problem can be formulated as $NN(x_q) = arg\,min\,dist(x_q, x)\,,\ x \in X$ where $dist(.)$ represents a specific distance metric (e.g., Euclidean distance) that determines the closest candidates to x_q in the feature space [16].

To further clarify the similarity retrieval process, a simple flowchart of the general Content-Based Image Retrieval (CBIR) [24] is presented in Figure 1.2 which is taken from [16].

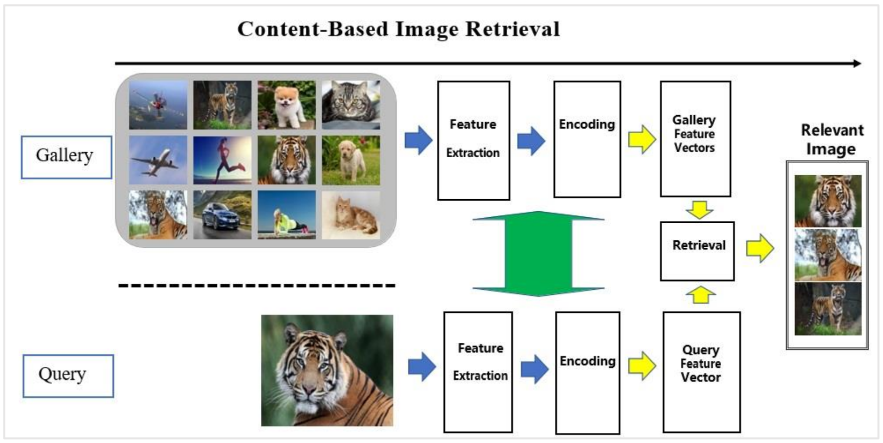

Figure 1.2: The general procedure of CBIR.

This framework of CBIR could be easily extended to the other related tasks involving different data types. In CBIR, images are first represented with various feature vectors and then encoded into alternative representations following certain patterns like encoding function [16]. Here, the encoding function, which is also the learning objective in most search frameworks, should be carefully designed to obtain better performance in the upcoming retrieval tasks. Then, the gallery and query data are pre-computed by the learned encoding function and their encoded feature vectors are measured under a distance metric. By sorting those distances in an ascending order, the candidates from the gallery with the smallest distances are returned as the relevant (i.e., similar) neighbors to one specific query. Generally speaking, the search efficiency depends on the computational complexity of the retrieval phase. Depending on how complex the retrieval phase is, the search efficiency will vary. At the same time, the search accuracy is usually determined by the proper design of the encoding mechanism when using fixed feature extractor.

Early works in the research of similarity retrieval perform the exhaustive/exact NN search in the retrieval process. In practical applications, such a search strategy (i.e., linear scan) is weak in tackling large datasets with many samples [21]. Later on, some tree-based search schemes are proposed to subdivide the feature space for data samples via employing various tree structures for fast search. As a representative, the K-Dimensional tree (KD-tree), is a method for indexing the data for quick query response [21]. However, a disadvantage of this method is that it cannot handle cases with high dimensions, known as the curse of dimensionality, whose computational costs grow exponentially with increasing dimensions, making it less suitable for large-scale retrieval applications [16].

Consequently, Approximate Nearest Neighbor (ANN) search has been developed rapidly, where the Hash-based approaches draw considerable attention in this research field to overcome the limitations via conducting efficient retrieval in low-dimensional (i.e., compact) Hamming space [22]. The ANN approach is more efficient and has been shown to be sufficient and useful in many practical situations. Additionally, ANN frequently meets search requirements and greatly reduces search complexity, drawing a lot of research [22]. The core idea of hashing is to represent the high-dimensional real-valued original data with a series of compact binary codes while preserving the semantics as much as possible during the code learning, thus accelerating the retrieval process without compromising the accuracy. Every duplicate copy of the image also has the exact same hash value. Thus, it is sometimes referred to as a Digital Fingerprint.

According to Gionis et al. [25] two advantages of hashing algorithms are presented as follows:

- Memory requirements are drastically reduced when using compact binary codes to store huge features and retrieve massive volumes of data.
- It is advantageous to use compact binary code for similarity computation because Hamming distance computation just requires bitwise operations.

These two benefits make hashing extremely competitive in conducting large-scale visual-related similarity search tasks. Robust hash functions are closely related to CHFs in mapping a large input data into a small fixed-length binary string. The key difference is their tolerance to minor incidental changes in the input sequence whilst remaining sensitive to large content changes. Given images I and I' and their perceptually similar copies with a minor distortion I_d and I'_d , and an image hashing function $hash_k(.)$ depending on a secret key k, the required characteristics of $hash_k(.)$ can be categorized into the following five groups [26]:

1. **Uniqueness**: Perceptually distinct images should have unique hashes,

$$Pr\left(hash_k(I) \neq hash_k(I')\right) \geq 1 - \tau, 0 \leq \tau < 1, \tag{1.2}$$

2. **Compactness**: The hash size should be much smaller than that of the original image I,

$$Size\left(hash_k(I)\right) \ll size(I), \tag{1.3}$$

3. **Perceptual Robustness**: Perceptually identical images should have similar hashes,

$$Pr\left(hash_k(I) \approx hash_k(I_d)\right) \geq 1 - \varepsilon, 0 \leq \varepsilon < 1, \tag{1.4}$$

When images are distributed via the Internet, distortions are inevitable due to lossy compression and noisy transmission channels, etc. Therefore, image hashing should be resistant to such distortions and attacks against image identification and retrieval processes and ensure that perceptually comparable images have similar image hashes. Figure 1.3 is taken from [26] and displays an original image along with deformed reproductions of it. These images experienced some content-preserving distortions and attacks, yet perceptually they are the same in the Human Visual System (HVS).

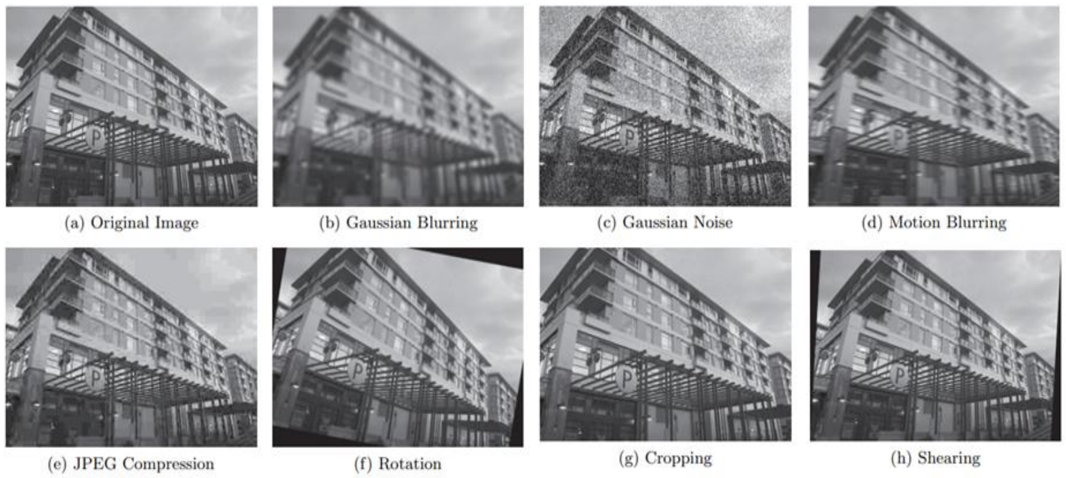

Figure 1.3: Examples of distorted image copies under different content-preserving attacks.

The perceptual robustness of image hashing guarantees that these images have very close hashes if the algorithms are robust enough against these attacks.

4. **One-way**: Ideally, a hash function should be irreversible, $I \mapsto hash_k(I)$
5. **Unpredictability**: The hash is intractable without the secret key,

$$Pr\big(\, hash_k(I) \neq hash_{k'}(I)\big) \geq 1 - \delta, 0 \leq \delta < 1, \tag{1.5}$$

Ideally, all the above parameters ε, τ, and δ should be close to zero for a proper designed hashing function to generate unique and compact image hashes, which are robust enough against perceptually insignificant distortions and secure enough to prevent unauthorized access. In this thesis, we will focus on learning effective binary representation of an image with deep reinforcement hashing technique.

In this thesis, we explore the utilization of deep reinforcement learning for image hashing, presenting a novel approach to decision-making. We frame the image hashing problem within the framework of a MDP, which provides a formal representation of sequential decisions made by an agent. Our objective is to construct binary hash codes for images that effectively minimize the Hamming distance between images of the same class and maximize the distance between images of different classes. Our proposed Deep Reinforcement Learning-based Image Hashing architecture comprises two key components: a CNN-based binary hash code extraction module and a binary hash code regeneration module that retains valuable bits. Firstly, we leverage CNN, specifically AlexNet [38], to determine which hash code preserves the similarity between image pairs and their corresponding semantic labels. To achieve this, we set the output dimension of the last layer of AlexNet to match the desired length of the extracted hash codes, while keeping all other layers unchanged. Given the potential limitations of hash functions, which can result in long binary codes that are the same for different inputs, it becomes crucial to remove redundant bits while preserving those that maintain uniqueness and accurately represent the original data. Therefore, in the second step, we employ the Proximal Policy Optimization (PPO) algorithm [39], which belongs to the actor-critic family of reinforcement learning algorithms. Actor-critic algorithms combine value-based and policy-based methods, with the actor representing the policy that maps states to actions (i.e., hash codes), and the critic estimating the expected reward for a given state or state-action pair. Throughout this thesis, we investigate and develop the aforementioned deep reinforcement learning-based approach to enhance the generation of binary hash codes for images. By incorporating CNN-based extraction and PPO-based regeneration, we aim to optimize the preservation of similarity and uniqueness, thereby providing an improved representation of the original image data. The rest of the thesis is organized as follows. Chapter 2 briefly reviews image hashing and content-based fingerprinting and some related works. In Chapter 3, we describe in detail our new framework of Deep Reinforcement Learning-based Image Hashing Technique. In Chapter 4, we present the experimental results and evaluation analysis of our proposed approach, and the thesis concludes in Chapter 5 along with possible future work.

Chapter 2

Background and Literature Review

2.1 Hashing Framework for Images

An image hashing framework refers to a system of methods and algorithms used to convert an image into a compact digital signature [27]. The purpose of image hashing is to efficiently represent an image in a compact and unique form, allowing for tasks such as CBIR, image authentication, and tamper detection [21]. The major components that are critical to designing a robust and secure digital image hashing algorithm are shown in Figure 2.1.

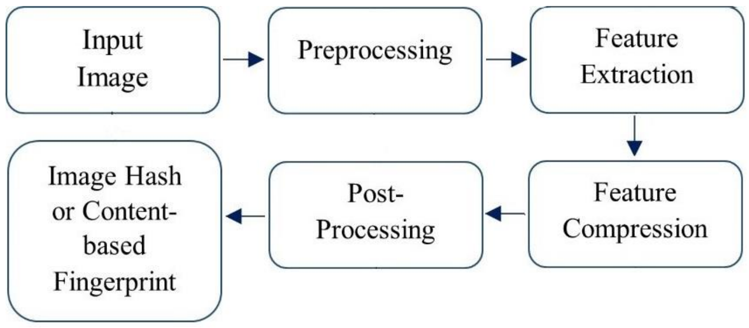

Figure 2.1: The framework of digital image hashing and content-based fingerprinting review.

2.1.1 Preprocessing

Prior to feature extraction, the preprocessing step filters the image content. As a result, some distortions, such as additive noise, are prevented from affecting the robustness of features. As part of the preprocessing, some works normalize images into a standardized format, which can facilitate the feature extraction process. The common pre-processing operations applied on digital images are illustrated as follows:

- **Color Space Dimension Reduction**: Color images are first converted to grayscale images to reduce the computational cost for feature extraction (e.g., 3D to 2D) [28]. Another way is based on the color space transform that converts RGB space to Hue-Saturation-Value (HSV) space. The conversion from RGB color space to HSV color space is nonlinear. Let H be the Hue of a pixel in the HSV model, S be the Saturation and V be the Brightness, respectively. Thus, their values can be determined in the Equations below:

$$H = \begin{cases} \frac{(-B+G)\times\frac{\pi}{3}}{Max(R,G,B)-Min(R,G,B)}, & if\ R = Max(R,G,B) \\ \frac{(-R+B)\times\frac{\pi}{3}}{Max(R,G,B)-Min(R,G,B)}, & if\ G = Max(R,G,B) \\ \frac{(-G+R)\times\frac{\pi}{3}}{Max(R,G,B)-Min(R,G,B)}, & if\ B = Max(R,G,B) \\ Undefined, & if\quad R = G = B \end{cases} \tag{2.1}$$

$$S = \begin{cases} \frac{Max(R,G,B)-Min(R,G,B)}{Max(R,G,B)}, & if\ Max(R,G,B) \neq 0 \\ 0 \qquad\qquad , & if\ Max(R,G,B) = 0 \end{cases} \tag{2.2}$$

$$V = Max(R,G,B) \tag{2.3}$$

- **Resizing**: Images are resized to a predefined size (usually very small, e.g., 256 × 384) as a default format. Resizing has two benefits: first, it is significantly less computationally expensive, improves the effectiveness of hash production, and enables quick indexing and retrieval. Second, characteristics retrieved from images with a defined size are more resistant to geometric attacks like changing aspect ratios.
- **Filtering**: It is a productive method for making the derived characteristics more noise resistant. Digital images can be processed to reduce noise using some well-known filters, like the median filter and the Gaussian filter. The image hashing system must be resistant to blurring distortions since these low-pass filters would also remove certain picture content information and produce blurred images.
- **Illumination Normalization**: In computer vision and image processing, illumination normalization is a technique used to account for differences in lighting conditions across distinct images. It seeks to eliminate or minimize the impacts of illumination variations, such as shadows, highlights, and overall brightness, in order to increase the precision and dependability of ensuing image analysis activities.

2.1.2 Feature Extraction

In image hashing and fingerprinting algorithms, feature extraction is one of the fundamental modules. The image features are extracted from the transformed image to generate the feature vector of L features where $L \ll M \times N$. A digital image hash is unique since it is derived from distinctive features of digital images [26].

Nevertheless, if two digital images are perceptually identical, then the extracted features, and therefore the image hashes, should be as similar as possible even when the images are subject to additive noise, blurring, geometric attacks, and other content-preserving techniques.

According to the related literature review, most previous works have focused on finding robust features that can withstand distortions and attacks, which are summarized as follows:

- **Image Pixels**: Image pixel values are the raw features that could be directly used for hash generation. However, an $N \times N$ image will have a feature vector with length N^2, which can be quite high dimensional. Therefore, a dimension reduction technique that could preserve the local similarity is required.
- **Invariant Feature Transform**: Coefficients in a transformed domain can be important features and sufficiently resistant to a broad class of attacks and distortions. There are several state-of-the-art transforms that can be used to extract robust features, including Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT).
- **Convolutional Neural Network**: A neural network which is designed to process multi-dimensional data like image and time series data is called a Convolutional Neural Network (CNN). The main benefit of CNNs is that automatic feature extraction is offered. The input data is initially forwarded to a feature extraction network, and then the extracted features can be fed into a hash function to produce related binary hash code.

2.1.3 Feature Compression and Post-Processing

Compactness is an important characteristic of image hashing. Since substantial characteristics can be condensed into brief real-valued or even binary sequences, this can essentially be thought of as a dimension-reduction procedure. Some typical methods of compression are summarized as follows:

- **Quantization**: It is widely employed for converting continuous feature space to finite discrete feature space and helpful for further signature encoding [29]. Popular approaches include interval quantization, binary quantization using threshold, and so on for image hash generation.
- **Random Projection**: It is one of the state-of-the-art dimension reduction techniques to project data in a high dimensional space into a lower dimensional space, while preserving the local similarity of the data [30]. The random projection approach can result in performances comparable to that of the conventional dimension reduction methods such as Principal Component Analysis (PCA) but be computationally more efficient.

2.2 Comparison and Decision Making

Following the framework of image hashing or content-based fingerprinting, a compact and secure hash is generated and associated with the corresponding original image in database as an index. When a query hash is received, it will be compared with the existing hashes based on the selected distance metrics and the corresponding image will be retrieved according to the classifiers. Hence, the distance metrics to measure the similarity between hashes and the classifiers to make decisions are also two important issues in hashing and fingerprinting schemes.

2.2.1 Distance Metrics

Given two hashes $H_1 = \{h_1(1), h_1(2), \dots, h_1(k)\}$ and $H_2 = \{h_2(1), h_2(2), \dots, h_2(k)\}$ of two images I_1 and I_2 with length k, the following distance metrics are usually employed:

$$Euclidean\ Distance: dist(H_1, H_2) = \sqrt{\sum_{i=1}^{k}\left(h_1(i) - h_2(i)\right)^2} \tag{2.4}$$

$$L1\ Norm: \ dist(H_1, H_2) = \sum_{i=1}^{k}|h_1(i) - h_2(i)| \tag{2.5}$$

$$Hamming\ Distance: \ dist(H_1, H_2) = \sum_{i=1}^{k}|h_1(i) \oplus^{*} h_2(i)| \tag{2.6}$$

The choice of distance metrics depends on the type of hashes. When the generated hashes are real-valued vectors, Euclidean distance or L1 Norm is usually employed. Otherwise, Hamming distance should be used for binary hashes. Hamming distances are preferable for the lower computational cost, while Euclidean distance or L1 Norm provide higher identification accuracy with the cost of the more computational burden.

2.2.2 Classifiers

After the similarity between hashes is measured by the selected distance metrics, classifiers are employed to make the decision for content identification. In most image hashing and fingerprinting algorithms, the simple nearest neighbor classifier or threshold-based classifiers are usually used for making decisions. Hash codes are classified using $Dist(H_1, H_2) \leq \xi$, where ξ is the selected threshold [33]. Although there are a lot of advanced classification methods proposed in machine learning, they are rarely employed in the image hashing area. The underlying reason is as follows: image hashing is an infinite clustering problem, which takes each original image as a new cluster and all its perceptually identical copies are assumed

*: The Hamming distance operator is $\oplus$

to lie in the neighborhood of the centroid (e.g., the original image). Hence, if advanced supervised classifiers, such as Support Vector Machine (SVM), are employed, they could only deal with the finite classification problems and must be re-trained, whenever a new original image is registered in a dataset or collection of images [10]. The re-training process may incur heavy computational cost when thousands of images are registered and training advanced classifiers to deal with classification for infinite classes is not feasible in practice.

2.3 Previous Work

The existing hashing methods can be divided into two categories: Traditional Hashing methods and Deep Neural Network (DNN) Image Hashing [33]. Image retrieval performance is limited by traditional hashing methods because they use handcrafted features as image representations, which cannot adequately represent the image content. The current deep hashing methods, also, have two major disadvantages. Firstly, to train most DNN methods, data is often broken into smaller sets termed "mini batches" and then processed. This is because mini-batch optimization, while widely used for training neural networks, has several drawbacks. One significant disadvantage is the potential loss of generalization due to the stochastic nature of mini-batch sampling, introducing noise into gradient estimates and leading to less stable optimization and suboptimal convergence. Selecting an appropriate batch size is crucial, as a small batch size can result in noisy gradients and slow convergence, while a large batch size may lead to memory limitations and longer training times. Additionally, mini-batch optimization is susceptible to getting trapped in local minima, particularly with small batch sizes, hindering the model’s ability to find the global minimum of the loss function. The choice of learning rate and its schedule presents challenges, requiring careful tuning to balance rapid convergence and preventing overshooting. Moreover, the frequent weight updates inherent to mini-batch optimization introduce increased computational overhead compared to batch optimization, where weights are updated after processing the entire dataset. Secondly, most existing methods generate redundant hash codes [33]. It may even be possible to throw away some bits of the generated hash codes without harming the retrieval accuracy. According to Shaik et al. [3] these redundant or even harmful bits can come from two sources. One source is the dataset that contains noisy data and, the second one, mini-batch training approaches only preserve local similarity relationships in the generated hash codes. Mini-batch methods, employed in training deep learning models, involve optimizing the model’s parameters using subsets of the entire dataset in each iteration. While this approach accelerates the optimization process, it can inadvertently emphasize local relationships between data points present in each mini batch. As a consequence, the model may become biased towards capturing local patterns or features, neglecting broader global features that contribute to the overall structure and semantics of the data. In the context of image hashing, this means that the generated hash codes might prioritize encoding similarities and dissimilarities within the sampled mini batches, potentially overlooking important global characteristics that could aid in accurate and efficient retrieval. Consequently, this local focus could lead to suboptimal hash codes that lack the holistic understanding needed for effective image representation and retrieval across the entire dataset. Existing hashing methods can also be categorized into two classes [33]: data-dependent and data-independent. Data-dependent methods can be further divided into Unsupervised and Supervised methods based on their use of side information. To learn hashing functions, unsupervised methods do not require label information. In the following, we will briefly review traditional hashing approaches and DNN image hashing.

2.3.1 Traditional Hashing Methods

As a traditional hashing method, Fei-Fei et al. [31] in Bag-of-Visual-Words, model images as patches. A patch x is the basic unit of an image, all patches are indexed by $\{1, \ldots, T\}$. Bag-of-Visual-Words aims to develop a model that best represents the distribution of these patches across the categories of scenes. A scene or image is a sequence of N patches denoted by $X = (x_1, x_2, \ldots, x_n)$, where x_n is the n^{th} patch of the image. In recognition, all the patches first are identified in the unknown image. Then a category model that fits best the distribution of the patches of the image will be found. A category is a collection of I images denoted by $D = \{X_1, X_2, \ldots, X_I\}$. Choosing a category label, such as a mountain scene, is the next step. Based on the mountain class, they determine which intermediate theme(s) to pick for each patch while generating the scene. The patch is created by selecting a particular theme from a variety of possible themes. For example, if a “rock” theme is selected, this will in turn privilege some patches that occur more frequently in rocks (e.g., slanted lines). Now the theme favoring more horizontal edges is chosen, one can draw a patch, which is likely to be a horizontal line segment. The process of drawing both the theme and patch repeat many times, eventually forming an entire bag of patches that would construct an image of mountains. According to [31] the process that generates an image i formally from the model is as follows:

Choose a category label $c \sim p(c|\eta)$ for each image, where c is a variable that takes on values from the set $\{1, \ldots, C\}$ where C represents the total number of categories. Each image is assigned a category label, denoted by c which corresponds to a specific category or class. η is a C-dimensional vector of a multinomial distribution. Now for a particular image in category c, we want to draw a parameter that determines the distribution of the intermediate themes (e.g., how “water”, “sky” etc. are distributed for this scene). This is done by choosing

$\pi \sim p(\pi|c,\theta)$ for each image. π is the parameter of a multinomial distribution for choosing the themes. θ is a matrix of size $C \times K$, where θ_c is the K-dimensional Dirichlet parameter conditioned on the category c. K is the total number of themes. For each N patches x_n in the image:

- Choose a theme $z_n \sim$ Mult(π). z_n is a K-dim unit vector. $z_n^k = 1$ indicates that the k^{th} theme is selected (e.g., "rock" theme).
- Choose a patch $x_n \sim p(x_n|z_n,\beta)$ where β is a matrix of size $K \times T$. K is again the number of themes and T is the total number of patches. Therefore, we have $\beta_{kt} = p(x_n^t = 1|z_n^k = 1)$

Given the parameters θ, η and β, we can now write the full generative equation of the model. It is the joint probability of a theme mixture π, a set of N themes z, a set of N patches x and the category c, as shown in Equation 2.7:

$$p(x,z,\pi,c\,|\theta,\eta,\beta) = p(c|\eta)p(\pi|c,\theta) \tag{2.7}$$

The Dirichlet parameter θ for each category is a category-level parameter, sampled once in the process of generating a category of scenes. The multinomial variables π are scene-level variables, sampled once per image. Finally, the discrete theme variable z and patch x are patch-level variables, sampled every time a patch is generated. Fei-Fei et al. [31] train and test their model on a dataset containing 13 categories of natural scenes, making it the largest dataset of its kind at that date. In their model when detecting or creating an image, the combined likelihood of patches, themes, and categories is computed. However, in extensive datasets like ImageNet, which contain over 20,000 categories and more than 14 million images, the manually designed features employed as image representations are inadequate for accurately capturing the content of an image. This is due to the fact that the identified image can potentially belong to hundreds or even thousands of categories. This approach also limits the performance of single-image retrieval because it imposes long-term computation and is applicable to datasets that are labeled. Indyk et al. [25] propose Locality Sensitive Hashing (LSH) as a data-independent unsupervised method to image hashing which is also the most representative one. LSH uses randomly generated hash functions that hash data points or image features into buckets in such a way that data points within a bucket have a high probability of being the same, while data points farther apart are likely to be in different buckets. LSH maps images into binary codes while preserving cosine similarity using random projections derived from Gaussian distributions. The probability of collision of two points p and q is closely related to the distance between them. Specifically, the larger the distance, the smaller the collision probability. This intuition is formulized as follows [47]. Let $dist(.,.)$ be a distance function of elements from a set S, and for any $p \in S$ let $\beta(p,d)$ denote the set of elements from S within the distance d from p. A family H of functions from S to U is called (r_1, r_2, p_1, p_2) −sensitive for $dist(.,.)$ if for any $q, p \in S$:

$$\begin{cases} if\ p \in \beta(q, r_1)\ then\ Pr_H[h(q) = h(p)] \geq p_1, \\ if\ p \notin \beta(q, r_2)\ then\ Pr_H[h(q) = h(p)] \leq p_2. \end{cases} \tag{2.8}$$

In the above definition, probabilities are considered with respect to the random selection of a function h from the family H. In order for a locality-sensitive family to be useful, it has to satisfy the inequalities $p_1 > p_2$ and $r_1 < r_2$. It is worth noting that if $dist(.,.)$ corresponds to the Hamming distance $dist_H(.,.)$, the family of projections on a single coordinate is an example of a locality-sensitive family. However, achieving satisfactory performance in LSH typically requires generating longer codes and employing multiple hashing tables. Observe that if $dist(.,.)$ is the Hamming distance $dist_H(.,.)$, then the family of projections on one coordinate is locality sensitive. Therefore, based on the supported statements, we can say that the family of projections on one coordinate is a specific instance of a locality-sensitive family that satisfies the desired properties. Nevertheless, effective LSH implementations often involve generating longer codes and utilizing multiple hashing tables to ensure desired performance levels.

2.3.2 Deep Neural Network Image Hashing

Deep neural networks (DNNs) have been successful on many computer-vision tasks, like image classification and object detection, and now image retrieval using deep hashing methods has shown promising results by leveraging powerful feature representation capabilities. Cao et al. [27] propose HashNet, a supervised method for image hashing. It utilizes data-dependent hash encoding schemes to enhance image retrieval, outperforming data-independent methods like LSH. HashNet leverages DNNs to encode nonlinear hash functions, enabling more efficient learning of end-to-end feature representation and hash coding. In similarity search problems, we are given a training set of N points $\{x_i\}_{i=1}^{N}$, each represented by a d-dimensional feature vector $x_i \in \mathbb{R}^d$. Some pairs of points x_i and x_j are provided with similarity labels s_{ij} :

$$\begin{cases} s_{ij} = 1 & if\ x_i\ and\ x_j\ are\ similar \\ s_{ij} = 0 & if\ x_i\ and\ x_j\ are\ dissimilar \end{cases} \tag{2.9}$$

The goal of deep learning to hash is to learn nonlinear hash function $f : x \longrightarrow h \in \{-1, 1\}^k$ from input space R^d to Hamming space $\{-1, 1\}^k$ using DNNs, which encodes each point x into compact k-bit binary hash code $h = f(x)$ such that the similarity information between the given pairs S can be preserved in the compact hash codes. In supervised hashing, the similarity set $S = \{s_{ij}\}$ can be constructed from semantic labels of data points or relevance feedback from click-through data in real retrieval systems. Figure 2.2 taken from [27] shows the HashNet architecture for solving the data imbalance and ill-posed gradient problems. The architecture accepts pairwise input images $\{x_i, x_j, s_{ij}\}$ and processes them through an end-to-end pipeline of deep representation learning and binary hash coding:

1. A CNN for learning deep representation of each image $\mathrm{x_I}$
2. A fully connected hash layer, fch, for transforming the deep representation into k-dimensional representation $\mathrm{z_I} \in \mathbb{R}^{\mathrm{k}}$
3. A sign activation function $\mathrm{h} = \mathrm{sgn}(\mathrm{z})$ for binarizing the k-dimensional representation $\mathrm{z_I}$ into k-bit binary hash code $\mathrm{h_I} \in \{-1, 1\}^{\mathrm{k}}$
4. A weighed cross-entropy loss for similarity-preserving learning from imbalanced data

According to Cao et al. in [27] the ill-posed gradient problem of the non-smooth activation function $h = sgn(z)$ is addressed using a continuation approach. This approach begins with a smoothed activation function $y = tanh(\beta x)$ and gradually increases the non-smoothness by raising the value of β as the training progresses. Eventually, the function transitions back to the original sign activation function. This technique is employed to mitigate optimization challenges associated with the original sign activation function.

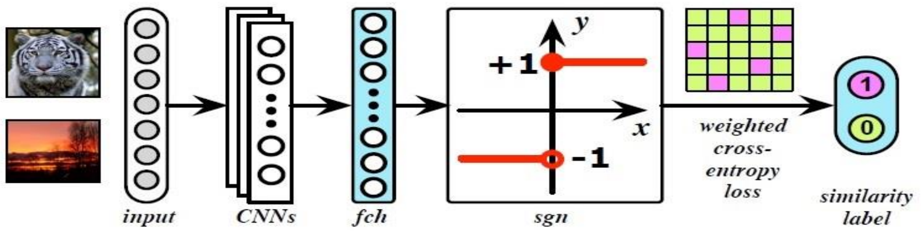

Figure 2.2: The proposed HashNet for deep learning to hash by continuation, which is comprised of four key components: (1) Standard convolutional neural network (CNN), e.g. AlexNet and ResNet, for learning deep image representations, (2) a fully-connected hash layer (fch) for transforming the deep representation into K-dimensional representation, (3) a sign activation function (sgn) for binarizing the K-dimensional representation into K-bit binary hash code, and (4) a novel weighted cross-entropy loss for similarity- reserving learning from sparse data.

Chapter 3

Proposed Approach

3.1 Deep Reinforcement Learning to Hash

Binary embedding a.k.a hashing enables efficient linear scan for two reasons. First, computing Hamming distances is much faster than computing distances between high-dimensional floating-point vectors. Second, the entire dataset consumes much smaller memory so, it may reside in fast memory rather than hard disk. Traditional image hashing methods take a feature vector as input and produce a compact binary vector. According to Wang et al. [34], in order to overcome the challenge of directly learning the optimal binary code, many traditional methods employ a two-step approach. In the training phase, they initially relax the discrete constraint, allowing for continuous solutions. Later, during the testing phase, these continuous solutions are rounded to obtain the binary code. However, such an inconsistency between training and testing could result in an undesired performance. Moreover, since the binary embedding procedure is independent of the feature extraction stage, the performance of traditional image hashing methods is constrained by the quality of original features.

To address this problem, deep hashing is recently proposed to jointly optimize the feature extraction and binary embedding steps [34]. Typically, most deep hashing methods first adopt a CNN or multi-layer perceptron to extract the real-value representation of the input image, then quantize the real-value representation to binary code. Thanks to the simultaneous optimization of feature extraction and binary embedding steps, deep hashing, especially deep supervised hashing, has demonstrated superior retrieval performance than those traditional hashing methods. Nevertheless, the train/test inconsistency in deep hashing is even more severe than in traditional hashing, due to the inherent conflict between hashing and back propagation [30][34].

One inherent conflict between the two techniques is that backpropagation requires the ability to perform gradient-based optimization on the inputs, which is not possible when the inputs have been hashed. Hashing is a non-differentiable function, meaning it does not have a gradient and therefore cannot be used in backpropagation. Another conflict is that hashing is a deterministic process, meaning the same input will always produce the same output. This makes it difficult to update the weights of the network during training, as the same input will always produce the same output, making it hard to learn from the data. For example, a rigid sign function, whose gradient is either zero or does not exist, is often necessary for binarizing the real-value representation to binary code in deep hashing approaches. Meanwhile, DNNs require all components to be differentiable, so that the parameters of the network could be gradually updated by back-propagation. Deep hashing methods generally approximate rigid sign functions by smooth functions such as sigmoid and Hyperbolic Tangent (tanh) in the training phase to facilitate the backward pass of the network [34]. After the training is completed, the rigid sign function is harnessed to output the binary code. Such inconsistency could lead to sub-optimal performance. Also, sigmoid or tanh unit is known as inappropriate for deep learning, as their gradients are nearly zero for most inputs. A tiny gradient could cause vanishing gradient problem, which is exactly why ReLU activation function is required to overcome this problem, allowing models to learn faster and perform better. Deep Learning (DL) and Reinforcement Learning (RL) are both methods that learn autonomously. The difference between them is that DL involves learning from a training set, then applying that learning to a new data set, while RL involves dynamically learning through receiving continuous feedback by taking actions in each environment to maximize rewards. Due to its ability to make accurate decisions even on unknown data, RL has recently been used to generate image hashes. On the other hand, according to the possibility of using DNNs as the function approximator in an RL system, the term deep reinforcement learning arises which is a significant technical part to fulfill this thesis.

3.2 Markov Decision Process

MDP is a decision-making model in continuous, discrete, stochastic, and sequential environments [35]. As shown in Figure 3.1 the MDP represents a straightforward approach to achieving a goal through learning from interaction. The learner or decision maker is called the Agent. An agent interacts with the Environment through its choices, which are called Actions. As the agent selects an action, the environment responds to the action by presenting a new situation, called State, to it. The environment also provides rewards, special numerical values that agent seeks to maximize through its actions. The agent’s objective is to select actions to maximize a long-term measure of total reward. The Equation 3.1 represents the Transition Function (T) and Reward Function (R) in an MDP, describing the process of transitioning from state s to state s' by taking a specific action a.

$$\begin{gathered} \mathrm{T}(s, a, s') = \mathrm{P}[\mathrm{S}_{t+1} = s' \mid \mathrm{S}_t = s,\ \mathrm{A}_t = a] \\ \mathrm{R}_t = \mathrm{E}[\mathrm{R}_{t+1} \mid \mathrm{S}_t = s,\ \mathrm{A}_t = a] \end{gathered} \tag{3.1}$$

A method of learning known as RL, which is based on MDP, arises when an agent has to learn how to behave through trial-and-error interactions with a dynamic environment. Deep reinforcement learning has achieved breakthroughs in human-level performance in games like AlphaZero [36].

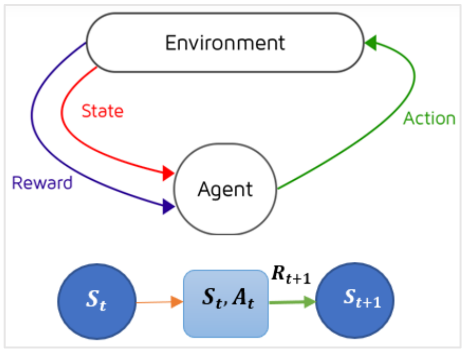

Figure 3.1: Markov Decision Process

In this thesis, we formulate image hashing as an MDP and solve it with deep reinforcement learning. Image hashing approaches are often addressed through unsupervised or supervised learning, but rarely through RL, while in recent studies, RL techniques have been used to obtain binary codes. Yuan et al. [37] propose a deep hashing CNN that generates binary codes and optimizes their deep hashing network using policy gradients from RL theory. Zhang et al. [33] as a further step, treat a batch of images as environment, triplet-loss as reward, and adjust the combination of hashing function to maximize the reward in RL.

3.3 Proposed Approach Components

Using deep reinforcement learning to hash, we introduce a new decision-making approach. Our image hashing problem can be represented within the framework of an MDP, which is a formal formulation of sequential decisions made by an agent. We consider binary space as the environment where the agent moves an image towards the binary codes using a set of actions. The purpose of our agent is to construct a binary code for each image that minimizes the Hamming distance between images of the same class and maximizes the Hamming distance between images of different classes. Our Deep Reinforcement Learning-based Image Hashing architecture consists of two parts:

- CNN-based binary hash code extraction,
- Regeneration of binary hash codes by retaining valuable bits.

Our first step determines which hash code preserves the similarity between each pair of images and their semantic labels using CNN. As our CNN, we use AlexNet [38]. The output dimension of the last layer in AlexNet is set to the length of the extracted hash codes, while all other layers remain unchanged. Due to the weakness of the hash functions, which can result in long binary hash codes that may also be the same for two different inputs, we must remove redundant bits while keeping valuable ones that preserve the uniqueness and most accurately represent the original data. Redundant bits, if present, tend to contribute unnecessary similarities between distinct inputs, potentially leading to collisions in the hash codes. By removing these redundant bits, we emphasize the unique characteristics of each input, making it less likely for different inputs to share the same hash code. This process enhances the discriminative power of the hash codes, reducing the chances of false positive similarities and improving the overall accuracy and effectiveness of the image hashing method in preserving similarity relationships between images and their semantic labels. Therefore, in the second step, we take advantage of PPO [39]. PPO is an actor-critic style algorithm, where the critic tries to fit the value function and the actor aims to update the policy distribution in the direction suggested by the critic.

3.3.1 CNN-based binary hash code extraction

Learning to hash involves representing a set of training images in the form of feature vectors $\boldsymbol{X} = \{x_i\}_{i=1}^{n} \in \mathbb{R}^{n\times D}$, where x_i can be image feature or raw pixels of an image. As part of supervised hashing, images are annotated with semantic labels $\boldsymbol{Y} = \{y_i\}_{i=1}^{n} \in \{0,1\}^{n\times m}$ where m is the total number of semantic categories. For example, $y_{ij} = 1$, means the i^{th} image belongs to the j^{th} category, and there can be multiple categories for an image.

A similarity matrix $\boldsymbol{S} \in \{-1,+1\}^{n\times n}$ is constructed to illustrate similarities between two images, where $\boldsymbol{S_{ij}} = +1$ means the i^{th} image and the j^{th} image are similar, otherwise $\boldsymbol{S_{ij}} = -1$ means the i^{th} image and the j^{th} image are dissimilar. When at least one semantic category is shared between two images, they are considered similar. There is a relationship between semantic labels and similarity matrix in the sense that semantic labels can be used to classify data into different categories, and similarity matrix can be used to measure the similarity between different pieces of data and classify it into different semantic labels. The relationship between semantic label $\boldsymbol{Y}$ and the similarity matrix $\boldsymbol{S}$ in shown in Equation 3.2:

$$\boldsymbol{S} = \min(\boldsymbol{Y}\boldsymbol{Y}^T, \boldsymbol{1}) \times 2 - \boldsymbol{1} \tag{3.2}$$

Where $\boldsymbol{Y}^T$ represents the transpose of $\boldsymbol{Y}$, $\min(.)$ is an element-wise minimum function, and $\boldsymbol{1}$ is a matrix whose elements are all ones. Deep learning to hashing is learning a non-linear hash function $Hash(\cdot)$ that maps the images from the original feature space to a compact binary space, while preserving the similarity relationships as shown in Equation 3.3:

$$Hash(\boldsymbol{X}) = \boldsymbol{Z} \tag{3.3}$$
$$\boldsymbol{X} = \{x_i\}_{i=1}^{n} \in \mathbb{R}^{n\times d}, \qquad \boldsymbol{Z} = \{\boldsymbol{z}_i\}_{i=1}^{n} \in \{-1,+1\}^{n\times k}$$

where k is the length of hash code (k bits). After that, we can use the Hamming distance to measure the number of bits that differ between every two hash codes. The complete scheme of the stage CNN-based binary hash code extraction is shown in Figure 3.2. Our first step in this stage is to extract a set of hash codes that preserves the original similarity relationships globally. As shown in Equation 3.4, Hamming distance, and inner product of two hash codes are naturally related.

$$H_{dist}(\boldsymbol{z}_i, \boldsymbol{z}_j) = \frac{1}{2}\left(k - \boldsymbol{z}_i \boldsymbol{z}_j^T\right) \tag{3.4}$$

where k is the length of a hash code. As can be seen, Hamming distances decrease with an increasing inner product. The inner product of two hash codes can therefore be viewed as a factor of affinity, which we can use to reconstruct the original similarity relationship as follows:

$$\min_{\boldsymbol{Z}} \mathcal{L}_1 = ||\boldsymbol{Z}\boldsymbol{Z}^T - k\boldsymbol{S}||_2^2 \tag{3.5}$$

as the variable $\boldsymbol{Z}$ exists in a binary space, optimizing Equation 3.5 would require exponential time. Therefore, it is necessary to eliminate the binary constraint from $\boldsymbol{Z}$. This can be achieved by employing the sigmoid function, a continuous function, to approximate $\boldsymbol{Z}$. The resulting approximation, denoted as $\widetilde{\boldsymbol{Z}}$, serves as a demonstration of $\boldsymbol{Z}$ in real space, as depicted in Equation 3.6:

$$\min_{\widetilde{\boldsymbol{Z}}} \mathcal{L}_1 = \left|\left|\widetilde{\boldsymbol{Z}}\widetilde{\boldsymbol{Z}}^T - k\boldsymbol{S}\right|\right|_2^2 \tag{3.6}$$

after the optimization, we can get $\boldsymbol{Z}$ by simply applying the sign function $sign(.)$ to $\widetilde{\boldsymbol{Z}}$:

$$\boldsymbol{Z} = sign(\widetilde{\boldsymbol{Z}}) \quad where\ sign(x) = \begin{cases} -1, & x < 0 \\ +1, & x \geq 0 \end{cases} \tag{3.7}$$

however, there may exist a large gap between $\boldsymbol{Z}$ and $\widetilde{\boldsymbol{Z}}$. This gap is called quantization error and is damaging to the retrieval accuracy.
To reduce the quantization error a regularization term is added to the Equation 3.6:

$$\min_{\widetilde{\boldsymbol{Z}}} \mathcal{L}_1 = \left|\left|\widetilde{\boldsymbol{Z}}\widetilde{\boldsymbol{Z}}^T - k\boldsymbol{S}\right|\right|_2^2 + ||\widetilde{\boldsymbol{Z}}| - \boldsymbol{1}| \tag{3.8}$$

where $\boldsymbol{1}$ is a matrix, whose elements are all ones. The similarity matrix $\boldsymbol{S}$ is a $n \times n$ matrix, where n is the number of training images. While $\boldsymbol{S}$ can make a full description of the original similarity relationships, it is too big to be stored in memory.

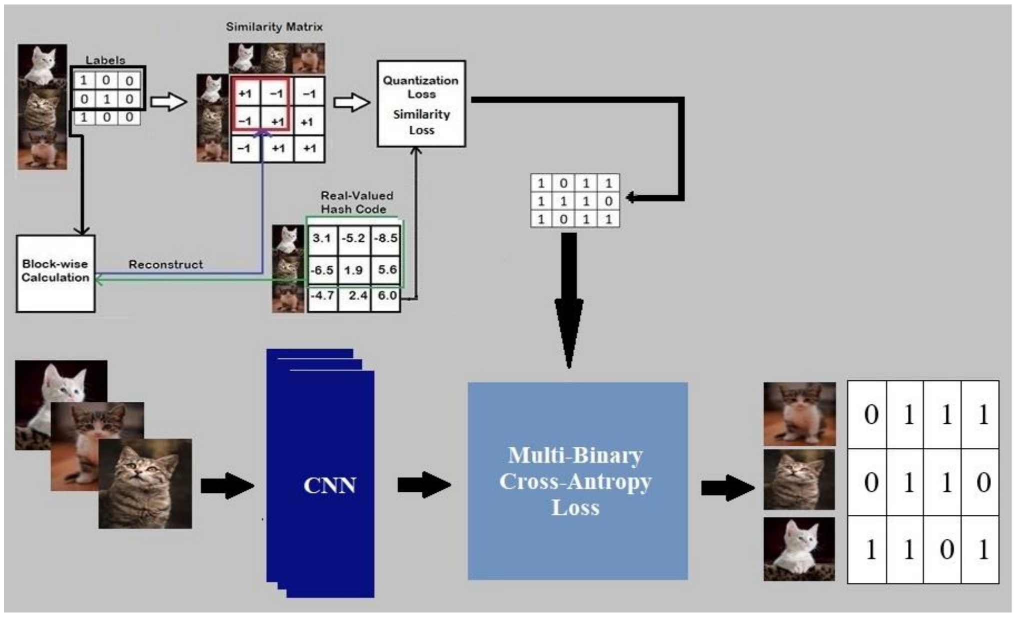

Figure 3.2: CNN-based binary hash code extraction

Since we already know how to construct $\boldsymbol{S}$ through the semantic label $\boldsymbol{Y}$ according to Equation 3.2, we then substitute $\boldsymbol{S}$ with $\boldsymbol{Y}$ as follows:

$$\min_{\widetilde{Z}} \mathcal{L}_1 = \left\|\widetilde{\boldsymbol{Z}}\widetilde{\boldsymbol{Z}}^T - k(\min(\boldsymbol{Y}\boldsymbol{Y}^T, \boldsymbol{1}) \times 2 - \boldsymbol{1})\right\|_2^2 + ||\widetilde{\boldsymbol{Z}}| - \boldsymbol{1}| \tag{3.9}$$

now we can calculate the similarity relationship in a block-wise style. Suppose the block size is $h \times w$, and n is divisible by h and w, so Equation 3.9 can be reformulated as:

$$\min_{\widetilde{Z}} \mathcal{L}_1 = \sum_{r=0}^{\frac{n}{h}-1} \sum_{c=0}^{\frac{n}{w}-1} \Big\|\widetilde{\boldsymbol{Z}}_{r,r+h-1}\widetilde{\boldsymbol{Z}}^T{}_{c,c+w-1} - k\left(\min(\boldsymbol{Y}_{r,r+h-1}\boldsymbol{Y}^T{}_{c,c+w-1}, \boldsymbol{1}) \times 2 - \boldsymbol{1}\right)\Big\|_2^2 + \left||\widetilde{\boldsymbol{Z}}| - \boldsymbol{1}\right| \tag{3.10}$$

where $\boldsymbol{Y}_{r,r+h-1}$ is the r^{th} row to the $(r+h-1)^{th}$ row of $\boldsymbol{Y}$, the same goes for $\widetilde{\boldsymbol{Z}}$. Now we can perform the optimization in a block-wise way, which is shown as the red block on the similarity matrix in Figure 3.2. This red block will slide over the whole similarity matrix $\boldsymbol{S}$. Here, the gradient descent method is utilized to optimize in Equation 3.10, in which the gradient will be calculated for each block and then added together. This accumulated gradient will back-propagate only once, and thus the similarity relationships in each block are preserved. To facilitate the subsequent hash code mapping, we transform the generated hash codes $\boldsymbol{Z}$ from $\{-1,+1\}^{n \times k}$ to $\{0,1\}^{n \times k}$, which is denoted as $\boldsymbol{B} = \{\boldsymbol{b}_i\}_{i=1}^n = \frac{1}{2}(\boldsymbol{Z} + \boldsymbol{1})$. According to what we discussed before, there exist two major disadvantages of current deep hashing methods. First, most of them are trained in a mini-batch-based style which makes them only able to preserve local similarity relationships. Additionally, this mini-batch-based training strategy must sample from the entire collection of image pairs, which has a magnitude of $O(n^2)$. Consequently, training time is greatly increased by this inefficiency in data sampling. Therefore, there are two differences between the block-wise similarity calculation and the mini-batch-based method. Firstly, the block-wise calculation accumulates gradient information and, therefore, preserves global similarity information, whereas the mini-batch method usually updates parameters every time a batch is run. Secondly, the block-wise calculation only involves semantic labels, and no image features are involved, and thus is incredibly faster than the mini-batch-based hashing method which needs to involve image features. In this part to establish the similarity between each pair of images, we only need to know the labels (i.e., semantic annotations).

The process of block-wise hash code extraction is shown in Algorithm 1. Using the hash codes extracted from semantic label and similarity matrix we are able to map training images to them. This mapping is accomplished using DNNs and is formulated as a multi-label classification problem. In this process, we use AlexNet as the CNN architecture to generate output by taking training images as input. The output of the AlexNet is denoted as $F(x_i;\ \Theta) \in \mathbb{R}^k$ where Θ is the parameter of the network, and k is the length of the hash code. After that, the multi-binary cross-entropy loss is calculated for the output of CNN and the hash code of semantic labels extracted at the previous stage. The multi-binary cross-entropy loss is a measure of how well the binary hash codes that are generated by CNN model match the binary hash codes of semantic labels. The calculation of Multi-binary cross-entropy loss is shown in Equation 3.11:

$$\min_{\Theta} \mathcal{L}_2 = -\sum_{i=1}^{n} \boldsymbol{b}_i \log\Big(\sigma\left(F(\boldsymbol{x}_i;\ \Theta)\right)\Big) + (\boldsymbol{1} - \boldsymbol{b}_i) \log\Big(\boldsymbol{1} - \sigma\left(F(\boldsymbol{x}_i;\ \Theta)\right)\Big) \quad (3.11)$$

where $\sigma(.)$ is the Sigmoid function. We use the sigmoid function here because it is a mathematical logistic function with a characteristic that can take any real value and map it between 0 and 1.

Algorithm 1: Hash Code Extraction using Block-wise Similarity Calculation

```
Input: Training image labels: $\boldsymbol{Y} = \{\boldsymbol{y_i}\}_{\boldsymbol{i=1}}^{\boldsymbol{n}}$, Hash code length: $\boldsymbol{k}$, Number of epochs: $\boldsymbol{t_1}$, Window height: $\boldsymbol{h}$, Window width: $\boldsymbol{w}$.
for $\boldsymbol{t} = \boldsymbol{1}: \boldsymbol{t_1}$ do
    for $\boldsymbol{r} = \boldsymbol{0}: \frac{\boldsymbol{n}}{\boldsymbol{h}} - \boldsymbol{1}$ do
        for $\boldsymbol{c} = \boldsymbol{0}: \frac{\boldsymbol{n}}{\boldsymbol{w}} - \boldsymbol{1}$ do
            Calculate the block loss and block gradient according to Equation 3.10.
```

```
                Add the gradient to the overall gradient.
            end
        end
        Update $\tilde{\boldsymbol{Z}}$ according to the overall gradient.
    end
    Output: Get binary hash codes. $\boldsymbol{Z} = \boldsymbol{sign}(\tilde{\boldsymbol{Z}})$. Transform $\boldsymbol{Z}$ from $\{-\mathbf{1}, +\mathbf{1}\}^{\boldsymbol{n}\times\boldsymbol{k}}$ to
    $\{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}\times\boldsymbol{k}}$ : $\boldsymbol{B} = \{\boldsymbol{b_i}\}_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} = \frac{\mathbf{1}}{\mathbf{2}}(\boldsymbol{Z} + \mathbf{1})$.
```

Since we map features extracted from images to 0 or 1, this mapping is a binary classification. In order to optimize AlexNet model we use Stochastic Gradient Descent (SGD) as an optimization algorithm. In order to update the parameters of the model using SGD we take the derivative of the multi-binary cross-entropy loss function. The gradient, as represented by the derivative in Equation 3.12, provides essential guidance for minimizing the loss function. It reveals the direction in which the loss function decreases the most, allowing us to adjust the model's parameters accordingly. By updating the parameters in the opposite direction of the gradient, we actively work towards minimizing the loss function and improving the model's performance. In gradient-based optimization, the primary objective is to minimize the loss function. Although the gradient points in the direction of steepest ascent, which corresponds to the maximum increase in the loss function, we adjust the model's parameters in the opposite direction of the gradient, enabling us to move in the direction of steepest descent. This iterative process gradually steers the model towards optimal parameter values that minimize the loss function. In summary, the correct approach involves adjusting the parameters in the opposite direction of the gradient, ultimately minimizing the loss function.

$$\frac{\partial \mathcal{L}_2}{\partial \mathrm{F}} = -\sum_{i=1}^{n} (1 - \boldsymbol{b}_i)\,\sigma(F) - \boldsymbol{b}_i\big(\mathbf{1} - \sigma(F)\big) \tag{3.12}$$

The hash function that is created is denoted as $h(\boldsymbol{x}_i;\ \Theta)$, and the hash codes are denoted as $\boldsymbol{C} = \{\boldsymbol{c}_i\}_{i=1}^{n} \in \{0,1\}^{n\times k}$. Using a threshold, we can obtain them as shown in Equation 3.13:

$$\begin{gathered} h(\boldsymbol{x}_i;\ \Theta) = \boldsymbol{c}_i = I\big(\sigma\left(F(\boldsymbol{x}_i;\Theta)\right) \geq 0.5\big) \\ I(bool) = \begin{cases} 1, & if\ bool\ is\ true \\ 0, & if\ bool\ is\ false \end{cases} \end{gathered} \tag{3.13}$$

where $I(bool)$ is an element-wise indicator function. We have shown that by having semantic labels, we can generate a hash code for each label to preserve the original similarity relationship. But what if we have a new unseen image without knowing its semantic label? We show that we can also map it to a hash code so that this hash code matches well with those hash codes of labeled images. Algorithm 2 shows the process of mapping training images to hash codes.

Algorithm 2: Hash Code Mapping via Multi-Binary Classification

```
Input: Extracted hash codes: $\mathbf{B} = \{\mathbf{b_i}\}_{\mathbf{i=1}}^{\mathbf{n}}$, Number of epochs: $\mathbf{t_2}$.
for $\mathbf{t} = \mathbf{1}: \mathbf{t_2}$ do
    for $i = \mathbf{1}: \mathbf{n}$ do
        Feed the image $\mathbf{x_i}$ and its extracted hash code $\mathbf{b_i}$, into CNN, and compute multi-binary cross-entropy loss according to Equation 3.11, and back-propagate.
    end
end
Output: hash function $\boldsymbol{F}(\mathbf{x_i}; \Theta)$ and hash code $\boldsymbol{C}$
```

3.3.2 Regeneration of binary hash codes by retaining valuable bits

As a hash function goes from a big continuous or discrete space $\boldsymbol{X}$ to a smaller binary space $\boldsymbol{B}$, it is a many-to-one function, so different files may yield the same hash value which is said that a collision has happened. In most hash codes generated, especially the long ones, bit redundancy usually exists. To address the problem of bit redundancy, we model the sequential bit selection problem as a MDP and solve it using an RL algorithm with retrieval performance as a reward to improve its bit selection policy uniformly. Hence, given the binary code $\boldsymbol{C}$ generated in the previous stage, we select one optimal bit at each time step t. This bit is chosen to offer the most informative binary codes when combined with the bits that have already been chosen. In other words, the previously selected bits are the last state and are necessary to select the current bit, thereby meeting the Markov property. With such analysis, this formulation gives rise to a finite MDP, defined by the tuple $\boldsymbol{M} = (\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{r}, \boldsymbol{\gamma})$, where the state space S and the action space A are discrete and finite, P is the Markov transition probability function $\boldsymbol{P}: \boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{S} \to [0, 1]$, $\boldsymbol{r}$ represents the reward function $\boldsymbol{r}: \boldsymbol{S} \times \boldsymbol{A} \to [\boldsymbol{R}_{min}, \boldsymbol{R}_{max}]$ and $\boldsymbol{\gamma}$ is the discount factor. The main components of the MDP in our algorithm are detailed in the following:

- **State Space**: The state in our RL method is an array of binary hash codes. For each state $s \in \mathbf{S}, \mathrm{s} = \{0, 1\}^k$ where $\boldsymbol{k}$ is the original binary hash code length. Each state s can be represented as a binary string of length k, where k represents the original binary hash code length. The value 1 indicates a bit at the specified position has been selected, and the value 0 indicates the bit is not selected. In the initial state s_0, a vector of zero is always used.
- **Action Space**: We define an action as choosing a particular bit from a binary code; for example, the expression $\boldsymbol{a_t} = i$ means that the i_{th} bit was chosen at

time step t. $\boldsymbol{A} = \{1, 2, \dots, k\}$, where k is the original hash code length, defines the action space. As long as the matching bit has not already been selected, the action is valid. The agent's goal is to discover a strategy for choosing the desired number of bits. The agent interacts with the environment during the training phase to learn the bit selection policy.

- **State Transition Function**: Taking an action $a_t \in \mathbf{A}$, the state s_t transitions to a new state s_{t+1} with the bit selected by a_t turned to 1 deterministically. Therefore, the state transition distribution is a deterministic, $\boldsymbol{P}(s = s_{t+1} \mid s_t, a_t) = 1$.
- **Reward Function**: The reward function is a crucial component in MDPs since it quantifies the immediate benefit of a state transition. In other words, it maps each state-action pair to a numerical reward that the agent receives upon taking the action in that state. The goal of the agent is to maximize the expected sum of these rewards over time. We select mean Average Precision (mAP) as the reward function. Precision is defined as the fraction of true positive predictions among all positive predictions made by the classifier. Recall, on the other hand, is defined as the fraction of true positive predictions among all actual positive instances in the dataset. In other words, recall measures the completeness of the algorithm in choosing relevant bits. We select a bit at each step and directly optimize mAP as a reward in order to select more valuable bits in binary hash codes which have redundancy. Because of the correlation between bits in the hash bit selection problem, we cannot determine the quality of bit selections until the bit selection is completed. As a result, there is no reward in the bit selection process. Once completed and the goal code length is reached, the entire sequential bit selection process is equivalent to applying selection function $\boldsymbol{F}$ on original database codes $\boldsymbol{C}$. Thus, we obtain the target binary hash codes $\boldsymbol{C'} = \boldsymbol{F}(\boldsymbol{C})$. Then, a subset of $\boldsymbol{C'}$ is randomly sampled as query binary codes to compute mAP as a reward, which represents the quality of the selected bits. Specifically, the reward function r_t is defined as shown in Equation 3.14:

$$r_t = \begin{cases} 0, & if \;\; 1 \le t \le p-1 \\ mAP \text{ with the selected bits}, & if \;\; t = p \end{cases} \tag{3.14}$$

  according to this formulation of the rewards, hash bit selection contributes directly to optimal retrieval performance. It is worth noting that, while we use mAP as a reward during training, we simply use the labels of training images as supervision.

Figure 3.3 displays the full design of our suggested RL-based method to regenerate binary hash codes without unnecessary bits. As we mentioned before, by learning a hash bit selection function $\boldsymbol{F}(h(\mathbf{x_i}))$ $where$ $h(\mathbf{x_i}) = \boldsymbol{C}$, we can generate a new database of binary hash codes, named $\boldsymbol{C'}$ with code length $\boldsymbol{p}$, from the original database of generated binary hash codes with code length $\boldsymbol{k}$, where $\boldsymbol{p} < \boldsymbol{k}$. It is worth mentioning that $\boldsymbol{C'} = \{c_i'\}_{i=1}^{n} \in \{0,1\}^{n\times p}$. Our chosen RL algorithm is PPO [39], which performs impressively in a variety of applications. According to [48][49], PPO is a widely used RL algorithm that has shown impressive results in many applications with discrete action spaces. It is considered a type of actor-critic algorithm [35], which is a class of RL algorithms that combine both value-based and policy-based methods.

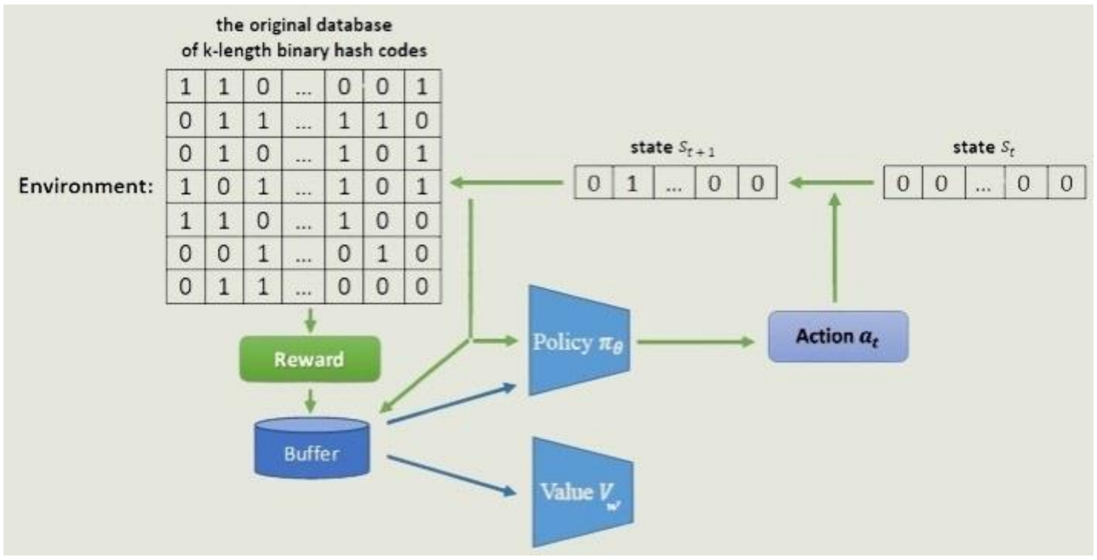

Figure 3.3*:* RL-based method to regenerate binary hash codes without unnecessary bits.

An actor-critic algorithm consists of two main components: an Actor network and a critic network. The actor network is responsible for learning the policy, which is mapping from states to actions. It is trained to select actions that maximize the expected cumulative reward. The critic network, also known as the value function, estimates the expected future cumulative reward for each state or state-action pair. It is trained to predict the value of the current policy. PPO is an on-policy RL algorithm that uses a neural network to approximate the policy and value functions. The main idea behind PPO is to keep the updates to the policy within a predefined range, which helps to ensure stability and improve the learning process. PPO combines ideas from both value-based and policy-based methods and has been shown to perform well on a wide range of RL tasks. We train with all the data in an episode, so the batch size is equal to $\boldsymbol{p}$, where $\boldsymbol{p}$ is the target binary hash code length and at the conclusion of the episode, PPO optimizes mAP as the reward and hopes to select more useful bits among redundant bits in hash binary codes by doing this. The agent chooses bits to create a target database of binary hash codes $\boldsymbol{C'}$ during the test step. In order to evaluate the states in our RL, we use a value network that maps states to state values and a policy network that maps states to the probability of actions.

The value network receives the state $\boldsymbol{s_t}$ as input at time step t and produces a real number $\boldsymbol{v} = \boldsymbol{V}_w(s_t) \in \mathbb{R}$ as output, i.e., state value. The policy network takes s_t as input, and outputs the probability distribution of actions $\boldsymbol{\pi} = \boldsymbol{\pi}_\Theta(s_t) \in \mathbb{R}^k$, where w is the parameters of the value network, and Θ is the parameters of the policy network. The value network and the policy network both have two completely connected 256-neuron hidden layers. The SoftMax activation layer is the final layer of the policy network, and it outputs a probability distribution of actions. However, the policy's calculated action can be incorrect, meaning that the bit corresponding to this action may have already been chosen previously. The repeated bits are not logical for producing informative and compact binary representation, and the policy with incorrect actions may not converge during training. Therefore, we use $-\infty$ to mask the SoftMax's input for the invalid action, or the bits that have already been chosen. The following formulation represents the policy's overall probability distribution:

$$\pi_\Theta(a_t = i \mid s_t) = \begin{cases} \pi_i, & if\ \ s_t^i\ = 0, \\ 0, & if\ \ s_t^i\ = 1, \end{cases} \tag{3.15}$$

where $i \in \{1, \cdot\ \cdot\ \cdot, k\}$ indicates the i_{th} bit of the state, and $\boldsymbol{\pi_i}$ denotes the output of the policy network for the i_{th} bit, with $\sum_{i=1}^{k} \pi_\Theta(a_t = i \mid s_t) = 1$. Algorithm 3 provides a summary of the learning method used to choose valued bits via RL.

Algorithm 3: Hash Code Re-generation using Actor-Critic

```
Input: Original database hash codes: $\boldsymbol{C} = \{\boldsymbol{c}_\mathbf{i}\}_{\mathbf{i=1}}^{\mathbf{n}\times k}$, Generated hash code length: $\boldsymbol{p}$
Prerequisite: Initialize policy parameters Θ and value function parameters $\boldsymbol{w}$;
repeat
    for $\boldsymbol{action} = \mathbf{1}, \mathbf{2}, \ldots, \boldsymbol{k}$ do
        Run policy $\boldsymbol{\pi}_{\Theta_{\boldsymbol{old}}}$ to select a bit.
    end
    Compute the mAP as a reward using the selected bits.
    Optimize network parameters Θ and $\boldsymbol{w}$.
    $\Theta_{\boldsymbol{old}} \leftarrow \Theta$
Until convergence
Use policy $\boldsymbol{\pi}_\Theta$ to select bits to obtain $\boldsymbol{C'}$.
Output: Database of Binary Hash codes $\boldsymbol{C'}$
```

Chapter 4

Experiments

We evaluate our approach against several state-of-the-art hashing methods. The evaluation is conducted on three widely used datasets: CIFAR-10, NUS-WIDE and MS-COCO. In many tasks related to computer vision and image processing, these datasets are used as standard benchmarks.

4.1 Setup and Datasets

The **CIFAR-10** [32] dataset (Canadian Institute for Advanced Research, 10 categories) is a subset of the Tiny Images dataset and consists of 60000 32x32 color images. We randomly select 1000 images, 100 images per category, as the query set. We select 5000 images, 500 images per category, as the training set, and keep the remaining images in the database.

The **NUS-WIDE** [40] is a public image dataset containing 269,648 images with a total of 5,018 tags collected from Flickr.com. These images are manually annotated with 81 semantic concepts, including objects and scenes. By keeping the top 21 concepts, we randomly select 2,100 images as the query set, 100 images per concept. The rest of the images are all as the database. We further randomly select 10,500 images from the database as the training set.

The **MS-COCO (Microsoft Common Objects in Context)** [41] dataset is a large-scale object detection, image segmentation, and captioning dataset published by Microsoft. We use the first release, which contains 82,783 training images and 40,504 validation images, where each image is labeled by some of 80 semantic concepts. There are 122,218 images in this dataset with semantic labels in total. We randomly sample 5,000 images as query images and treat the remaining as the database. Additionally, we randomly select 10,000 images from the database as training images.

Following standard evaluation protocol as previous work [27], two images i and j are considered similar if they share at least one semantic label ($s_{ij} = 1$) otherwise, they are dissimilar and ($s_{ij} = 0$). We compare retrieval performance of our approach with six classical or state-of-the-art hashing methods including:

- **Deep Reinforcement Learning for Image Hashing (DRLIH)** [33]: It is a deep reinforcement learning hashing network which uses RNN as agents to take previous hash function's error into account.
- **HashGAN** [42]: It is a technique for learning compact binary hash codes from a variety of images produced by generative models as well as actual photos.

- **CNNH (Convolutional Neural Network Hashing)** [43]: It uses CNN to learn a compact binary code for a data point. It typically consists of three main components: a feature extractor, a hash function, and a quantization function.
- **DNNH (Deep Neural Network Hashing)** [44]: A DNN is trained to produce a high-dimensional continuous feature representation of the data point, and then a quantization function is applied to the feature representation to produce a binary code. DNNH was introduced as a generalization of CNNH.
- **SDH** [45] is a supervised method, which leverages label information to obtain hash codes by integrating hash code generation and classifier training.
- **HashNet** [46] directly learns the binary hash codes and addresses the ill-posed gradient and data imbalance problems in an end-to-end framework of deep feature learning and binary hash encoding.

In each of the aforesaid hashing methods, the results are obtained from their authors' implementations. To be able to directly compare results to published reports, all methods used the same training and testing data. We evaluate the retrieval quality based on four standard evaluation metrics:

- **Mean Average Precision** (**mAP**) is the mean value of Average Precision over many queries which is a measure of the effectiveness of a retrieval system or algorithm. Average Precision (AP) is calculated for a single query or a single class in the case of object detection. It measures the average precision value at different recall levels. Precision is computed at each point where a relevant item is retrieved, and then the average is taken over these precision values. For a given query, the AP is defined as follows:

$$AP = \frac{1}{R}\sum_{k=1}^{n}\frac{k}{R_k} \times rel_k \tag{4.1}$$

  where n is the size of database, R is the number of relevant images in database, R_k is the number of relevant images in the top k returns, and rel_k is an indicator function that is 1 if the k-th image is relevant and 0 otherwise.
  Precision measures the fraction of positive detections that are actually correct, while recall measures the fraction of all positive instances that are detected. The formula of mAP is defined as follows:

$$\frac{1}{n}\sum_{i=0}^{n} AP_i \tag{4.2}$$

  where AP_i is the Average Precision for the i-th query or class. Since the calculation of AP involves sorting a large index array, it is believed to be slow. However, with the help of GPU, all mAP calculations can be accelerated through parallelism.

- **Precision-Recall curves** (**PR**) are commonly used to evaluate the performance of binary classification algorithms in computer vision and machine learning. A PR curve plots the precision values on the y-axis and the corresponding recall values on the x-axis. The curve is generated by varying the classification or retrieval threshold of the system and calculating the precision and recall values at each threshold. Typically, the curve starts at a high recall value (e.g., 1) with a lower precision value and gradually moves towards a lower recall value (e.g., 0) with a higher precision value.
- **Precision curves within Hamming distance 2** (**P@H ≤ 2**) is a variation of the standard Precision-Recall (PR) curves, used to evaluate the performance of a binary hash function used in image retrieval, but only for those retrieved items that have a Hamming distance of at most 2 from the query item. It allows us to evaluate the performance of the hash function in terms of how well it can retrieve similar items in the presence of slight variations (e.g., slight rotation, scaling, noise).
- **Precision curves with respect to the numbers of top returned samples (P@N)** are a variation of the standard Precision-Recall (PR) curves, used to evaluate the performance of a retrieval system or algorithm. A P@N curve plots the precision of the retrieval system at different recall levels, but only for the top N returned samples. The x-axis represents the number of returned samples, and the y-axis represents the precision. Following the HashNet algorithm, we adopt MAP@54000 for CIFAR-10, MAP@5000 for NUS-WIDE, and MAP@5000 for MS-COCO respectively.

When performing the block-wise hash code extraction, we use the standard gradient descent method, adopt Adam as the optimizer, and set the learning rate to 1.0. For mapping images to their semantic binary hash codes, we try AlexNet as the hash encoder, fine-tune all layers but train the last layer from scratch. Since we train the last layer from scratch, we set its learning rate 10 times large as the one in previous layers. The learning rate for previous layers is set to 10^{-5}. The learning rate for the last layer is set to 10^{-4}. The mini-batch size is set to 32. To implement the RL in a way that is both consistent with the public implementation of PPO and allows for the regeneration of binary hash codes and the removal of unnecessary hash bits:

1. A discount factor γ equal to 0.99 is chosen.
2. To accelerate training, we train all of the data in one episode, thus the batch size should be P, which is the target binary hash code length.

About the method called DRLIH [33] that leverages the power of RL as well we make a comparison with it in Table 4.2, following the same setting in it and using VGG-19 as the CNN. We adopt the VGG-19 network as the feature extractor network for mapping input features to the learned binary codes from semantic labels. As mentioned in the DRLIH method the first 18 layers of VGG-19 network follow the exact same settings as this network.

4.2 Experimental Results

Table 4.1 presents the mAP results of all methods using AlexNet, where all classical and state-of-the-art results which are shown in HashGAN [42] are used in our evaluation. SDH is a shallow hashing method that is on top of Table 4.1, whereas deep hashing methods are on the bottom. As we mentioned before, mAP provides a single scalar value that summarizes the overall performance of a retrieval or detection system. In fact, a higher mAP value indicates better performance and accuracy in retrieving relevant results. Our approach demonstrates superior performance compared to HashGAN, a deep hashing method, in terms of average mAP on three datasets: CIFAR-10, NUS-WIDE, and MS-COCO. The margins of improvement are substantial, with 7.1%, 8.71%, and 6% respectively. This success can be attributed to our algorithm's focus on preserving global similarity relationships and utilizing block-wise calculations, as opposed to constructing a small similarity matrix from a limited sample set. As a result, our approach consistently preserves the entire similarity matrix, providing a more comprehensive representation of image similarities compared to approaches relying on smaller sample-based matrices. We are also able to increase the average mAP on three datasets by 47.4%, 28%, and 36% compared to SDH, the best shallow hashing method with deep features as input.

Table 4.1: mAP of Hamming ranking for different number of bits on three datasets using AlexNet.

Method	***CIFAR-10***				***NUS-WIDE***				***MS-COCO***			
	16 bits	**32 bits**	**48 bits**	**64 bits**	**16 bits**	**32 bits**	**48 bits**	**64 bits**	**16 bits**	**32 bits**	**48 bits**	**64 bits**
SDH[45]	0.461	0.520	0.553	0.568	0.588	0.611	0.638	0.667	0.555	0.564	0.572	0.580
CNNH[43]	0.476	0.472	0.489	0.501	0.570	0.583	0.593	0.600	0.564	0.574	0.571	0.567
DNNH[44]	0.559	0.558	0.581	0.583	0.598	0.616	0.635	0.639	0.593	0.603	0.605	0.610
HashNet[46]	0.643	0.667	0.675	0.687	0.662	0.699	0.711	0.716	0.687	0.718	0.730	0.736
HashGAN[42]	0.668	0.731	0.735	0.749	0.715	0.737	0.744	0.748	0.697	0.725	0.741	0.744
Ours	**0.727**	**0.790**	**0.788**	**0.780**	**0.790**	**0.798**	**0.805**	**0.807**	**0.748**	**0.767**	**0.790**	**0.776**

Two insights are evident from the mAP results. Firstly, SDH as a shallow hashing method is unable to learn compact hash codes and discriminative deep representations across the end-to-end architecture, demonstrating why deep hashing methods are superior. Secondly, by jointly conserving similarity information and managing the quantization error, HashGAN, a deep hashing approach, learns fewer lossy hash codes and greatly outperforms the original CNNH and DNNH methods.

Our method outperforms the state-of-the-art HashGAN significantly for two key reasons. First, by using block-wise calculation in our technique, we want to preserve global similarity. As a result, it can always maintain the entire similarity matrix rather than a limited similarity matrix created from a mini batch. Because of this, we can fully preserve the global similarity relationships through block-wise hash code inference, whereas mini-batch-based approaches struggle to sample enough pairs from training data. Second, we can reduce 64-bit hash codes to 16-bit, 32-bit, and 48-bit versions by removing any redundant bits. Redundant bits are those that do not contribute significantly to the representation of the image. Redundant or potentially detrimental bits in hash codes can arise from two primary sources. The first source is the presence of noisy data within the dataset itself. The second source stems from the prevalent use of mini-batch training strategies, which limits the ability of the generated hash codes to preserve global similarity relationships and instead focuses on local similarities. There are two advantages to removing redundant bits. At first, it only needs to be trained once, and different de-redundancy settings can be used to generate hash codes of various lengths, greatly accelerating the training process for hashing algorithms. Second, we can create more valuable and compact hash codes by removing irrelevant bits from longer hash codes. Table 4.1 shows that the benefits for shorter codes (16 to 48 bits) are bigger than those for longer codes (64 bits).

Using Precision-Recall curves, we measure the retrieval performance of three datasets, and as can be seen in Figures 4.1 (a), (b), and (c), our proposed approach outperforms all comparison methods by a significant margin.

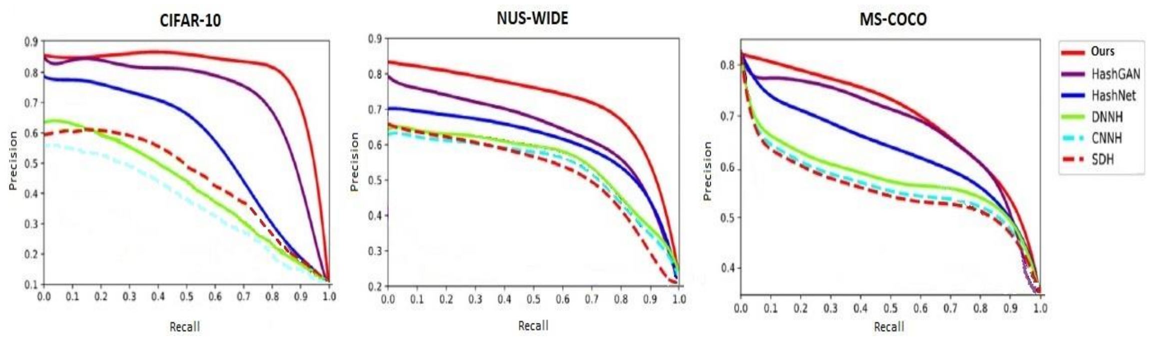

Figure 4.1: Precision-recall curve @ 64 bits; The experimental result of our approach and comparison methods on CIFAR-10, NUS-WIDE, MS-COCO

The performance in Precision within Hamming radius 2 (P@H≤2) is very important for efficient image retrieval since the Hamming ranking only requires $O(1)$ time cost for each query, which enables really fast pruning. Based on Figures 4.2 (a), (b), and (c), our approach achieves the highest P@H $\leq$ 2 on all three benchmark datasets. This proves that our hashing approach is able to learn more compact hash codes than all comparison methods to establish a more efficient and accurate Hamming ranking. Considering shorter hash codes will assign more images per hamming distance than longer codes, this improvement will be particularly noticeable.

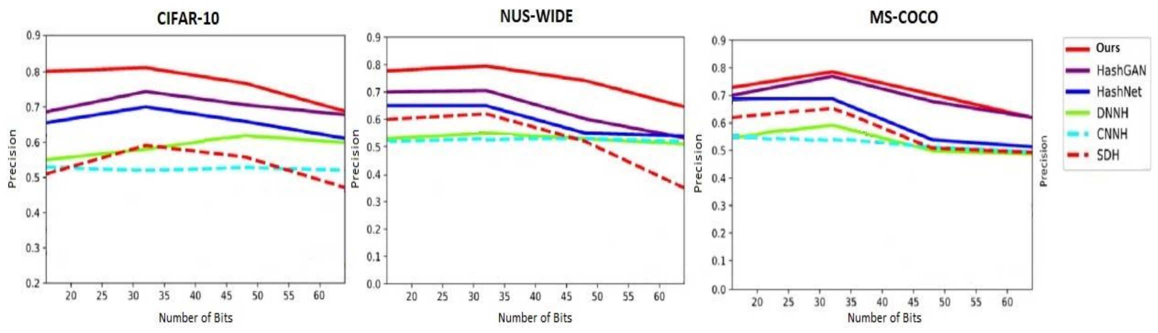

Figure 4.2: Precision within Hamming radius 2; The experimental result of our approach and comparison methods on CIFAR-10, NUS-WIDE, MS-COCO

In Figures 4.3 (a), (b), and (c), we show retrieval performance based on Precision curves with respect to different numbers of top returned samples (P@N), and we see that our proposed approach outperforms all the comparison methods.

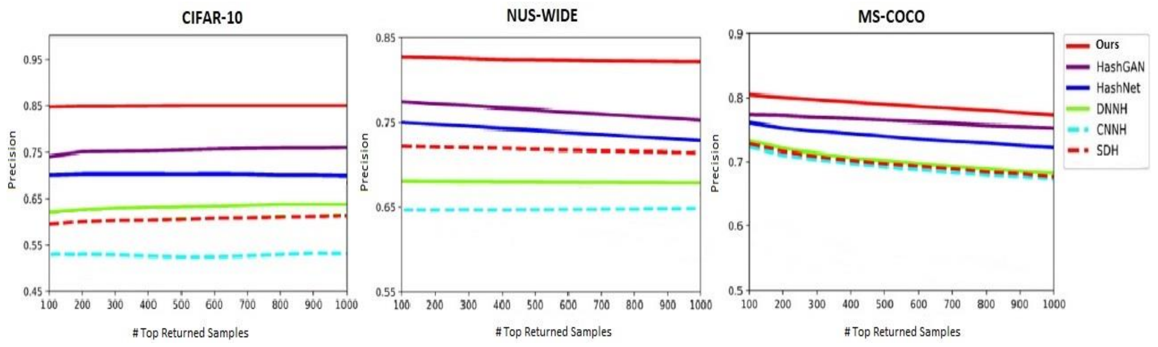

Figure 4.3: Precision curve for top-N @64bits; The experimental result of our approach and comparison methods on CIFAR-10, NUS-WIDE, MS-COCO

We make a comparison with DRLIH in Table 4.2, following the same setting in DRLIH and using VGG 19 as the CNN. DRLIH treats a batch of images as the environment, triplet-loss as the reward, and adjusts a combination of hashing functions to maximize the reward in reinforcement learning. However, the hard binary constraint in DRLIH is compromised, and train/test inconsistency still exists. As a result, we can see that our method somewhat beats DRLIH.

Table 4.2: mAP of Hamming ranking for different number of bits on NUS-WIDE & CIFAR-10 using VGG-19.

Method	***CIFAR-10***				***NUS-WIDE***			
	12 bits	**24 bits**	**32 bits**	**48 bits**	**12 bits**	**24 bits**	**32 bits**	**48 bits**
DRLIH[29]	0.816	0.843	0.855	0.853	0.823	0.846	0.845	0.853
Ours	**0.857**	**0.876**	**0.881**	**0.883**	**0.839**	**0.862**	**0.868**	**0.871**

Results from Table 4.2 show that we are also able to increase the average mAP on CIFAR-10 and NUS-WIDE datasets by 3.9%, and 2.7% compared to DRLIH as a deep reinforcement learning approach to image hashing.

As a further evaluation step for our hashing method based on deep reinforcement learning, the top-5 retrieval results on three benchmark datasets are displayed in Figure 4.4 along with some good and bad instances. These bad examples can be attributed to several reasons, some of which are applicable to all three datasets and others to particular datasets. Because the training set is too small in relation to the database for all three datasets, this poor retrieval example problem is primarily caused by that. The noisy label, or more precisely, the missing label, is a significant issue with NUS-WIDE. The greatest retrieval challenge with CIFAR-10 is the low resolution of the images.

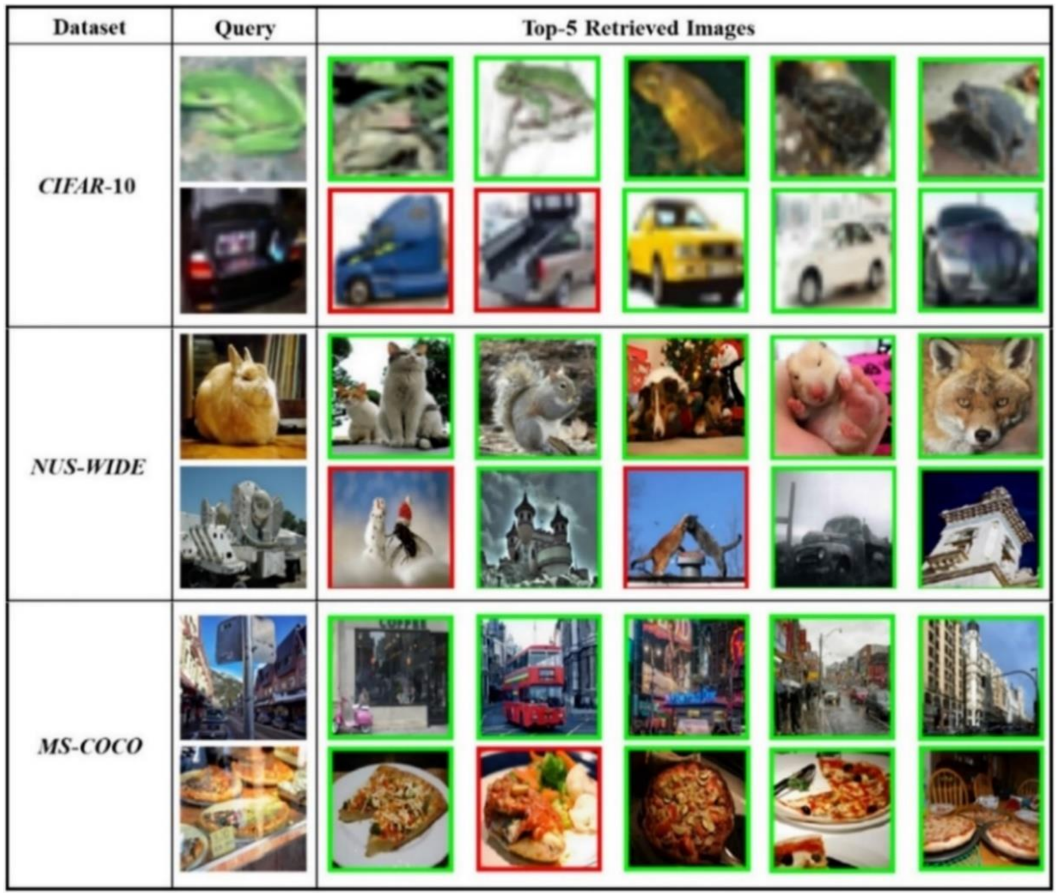

Figure 4.4: Some good and bad examples of the top-5 retrieval results on three benchmark datasets

Chapter 5

Conclusion and Future works

5.1 Conclusion

The concept of image hashing refers to the process of converting images from high-dimensional space into low-dimensional binary codes while preserving their perceptual similarities. Most existing deep learning-based methods are inefficient at the binarization of image data because they are trained in a mini-batch style and can't preserve global similarity relationships. To solve this problem, we propose a deep reinforcement learning-based technique for image hashing. We initially use each image's labels to generate hash codes in a block-wise style. The features retrieved from a picture are next mapped into the binary hash codes of the first stage using AlexNet, a particular kind of CNN. The majority of hashing techniques result in redundant bits in hash codes, which increase the likelihood of collisions, and cannot ensure the uniqueness of the hash codes created. In particular, we learn a hash bit selection policy at this stage, which maximizes mean Average Precision (mAP) during training to identify the most informative bits. We formulate the hash bit selection as an MDP. Therefore, we apply a well-known RL method known as PPO, an actor-critic algorithm style, choosing the informative bits, to remove unneeded bits from the binary hash code we learned. As a result, our binary hash codes are guaranteed to be unique and collision risk is minimized. In our experiments, we demonstrate the effectiveness of our approach using three widely used datasets, CIFAR-10, NUS-WIDE, and MS-COCO.

5.2 Future Work

In this thesis, we encountered some limitations related to the PPO algorithm. One limitation is its sensitivity to hyperparameters, requiring careful tuning for optimal performance and stability. To address this, future work can explore advanced hyperparameter optimization techniques and adaptive learning rate schedules. Another drawback is PPO's sample inefficiency, necessitating a large number of samples for effective learning. To overcome this, future research can focus on efficient exploration strategies, data augmentation techniques, and knowledge transfer methods to improve learning efficiency and reduce data requirements. Furthermore, advancements in optimization methods, such as alternative algorithms or integration of advanced techniques from other fields, can help overcome limitations and enhance PPO's performance and applicability in various domains. By addressing these challenges, future work can improve the effectiveness and efficiency of PPO, contributing to the advancement of reinforcement learning as a powerful decision-making paradigm.

Another possible future work can stand on conducting an ablation study where the policy-based bit selection step is omitted. This study could help quantify the contribution and effectiveness of the policy-based approach in our image hashing method. By comparing the performance of our method with and without the policy-based bit selection, we can gain insights into how much improvement is attributed to this specific component. This analysis would provide a clearer understanding of the role and impact of the policy-based approach in optimizing the binary hash codes and further validate its importance in our proposed approach.

As another avenue for future work, we plan to extend our image hashing method to Video Fingerprinting, enabling the identification of specific frames in videos by considering their temporal differences. Additionally, we aim to explore the integration of our approach with Blockchain technology, providing unique and immutable binary hash codes for transactions and blocks. These advancements would expand the application and enhance the reliability and security of image hashing in video analysis and Blockchain-based systems.

References or Bibliography

[1] Jinjie Zhang, and Rayan Saab, “Faster Binary Embeddings for Preserving Euclidean Distances,” arXiv:2010.00712v2. Mar 10, 2021.

[2] R. Biswas, and P. Blanco, “State of the Art: Image Hashing,” arXiv:2108.11794. Aug 26, 2021.

[3] Shaik, A.S., Karsh, R.K., Islam, M. et al. “A review of hashing based image authentication techniques.” Multimed Tools Appl 81, 2489–2516 (2022).

[4] Pun CM, Yan CP, Yuan XC (2018) “Robust image hashing using progressive feature selection for tampering detection.” Multimed Tools Appl 77(10):11609–11633

[5] Qiu-Yu Zhang and Guo-Rui Wu, “Digital Image Copyright Protection Method Based on Blockchain and Perceptual Hashing,” in International Journal of Network Security, Vol.25, No.1, PP.10-24, Jan 2023.

[6] Srivastava, Mayank & Siddiqui, Jamshed & Ali, Mohammad. (2019). “A Review of Hashing based Image Copy Detection Techniques.” Cybernetics and Information Technologies. 19. 3-27. 10.2478/cait-2019-0012.

[7] A. R. Javed, W. Ahmed, M. Alazab, Z. Jalil, K. Kifayat and T. R. Gadekallu, “A Comprehensive Survey on Computer Forensics: State-of-the-Art, Tools, Techniques, Challenges, and Future Directions,” in IEEE Access, vol. 10, pp. 11065-11089, 2022, doi: 10.1109/ACCESS.2022.3142508.

[8] M. Yu, Z. Tang, X. Zhang, B. Zhong and X. Zhang, “Perceptual Hashing With Complementary Color Wavelet Transform and Compressed Sensing for Reduced-Reference Image Quality Assessment,” in IEEE Transactions on Circuits and Systems for Video Technology, vol. 32, no. 11, pp. 7559-7574, Nov 2022.

[9] H.E. Lee, T. Ermakova, V. Ververis, B. Fabian, “Detecting child sexual abuse material A comprehensive survey,” Forensic Science International: Digital Investigation, Volume 34, 2020.

[10] S. Zhang, J. Li, M. Jiang, and B. Zhang, “Scalable discrete supervised multimedia hash learning with clustering,” IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), 2017.

[11] Chenggang Yan, Biao Gong, Yuxuan Wei, and Yue Gao, “Deep Multi-View Enhancement Hashing for Image Retrieval,” arXiv:2002.00169v2, 16 Jun 2020.

[12] M. Meenalochini, K. Saranya, G.V. Rajkumar, and A. Mahto, “Perceptual Hashing for Content Based image Retrieval,” In IEEE 3rd International Conference on Communication and Electronics Systems (ICCES), 2018.

[13] Liming Xu, Xianhua Zeng, Bochuan Zheng, Weisheng Li, “Multi-Manifold Deep Discriminative Cross-Modal Hashing for Medical Image Retrieval” In IEEE Transactions on Image Processing, 2022.

[14] Du, Ling & Ho, Anthony & Cong, Runmin. (2020). “Perceptual hashing for image authentication: A survey. Signal Processing: Image Communication.” 81. 10.1016/j.image.2019.115713.

[15] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, “Learning to hash for indexing big data: a survey,” Proceedings of the IEEE, vol. 104, no. 1, pp. 34–57, 2016.

[16] Wu, Gengshen, “Learning effective binary representation with deep hashing technique for large-scale multimedia similarity search.”/ Lancaster University, 2020. 180 p.

[17] Robin Landa, “Graphic Design Solutions. Cengage Learning”, 2019.

[18] Anup Singh, Kris Demuynck, Vipul Arora, “Simultaneously Learning Robust Audio Embeddings and balanced Hash codes for Query-by-Example,” arXiv:2211.11060v1 [eess.AS]. 20 Nov 2022

[19] Qiang Ma, Ling Xing, “Perceptual hashing method for video content authentication with maximized robustness “, EURASIP Journal on Image and Video Processing. 21 Nov 2021.

[20] Ou, Yang & Rhee, Kyung Hyune. (2010). “A Survey on Image Hashing for Image Authentication.” IEICE Transactions. 93-D. 1020-1030. 10.1587/transinf. E93.D.1020.

[21] J. Wang, T. Zhang, N. Sebe, H. T. Shen et al., “A survey on learning to hash,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 4, pp. 769–790, 2017.

[22] Alexandr Andoni, Piotr Indyk, Ilya Razenshteyn, “Approximate Nearest Neighbor Search in High Dimensions”, arXiv:1806.09823, 26 Jun 2018.

[23] K. D. Doan, P. Yang and P. Li, "One Loss for Quantization: Deep Hashing with Discrete Wasserstein Distributional Matching," 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 9437-9447, Doi: 10.1109/CVPR52688.2022.00923.

[24] Ibtihaal M. Hameed, Sadiq H. Abdulhussain, and Basheera M. Mahmmod, “Content-based image retrieval: A review of recent trends”, Cogent Engineering, 8:1, 1927469, 2021.

[25] A. Gionis, P. Indyk, R. Motwani et al., “Similarity search in high dimensions via hashing,” in International Conference on Very Large Data Bases (VLDB), vol. 99, no. 6, 1999, pp. 518–529.

[26] Lv, X. (2013). “Robust digital image hashing algorithms for image identification”. University of British Columbia.

[27] Z. Cao, M. Long, J. Wang, and P. S. Yu, “HashNet: Deep learning to hash by continuation,” in The IEEE International Conference on Computer Vision (ICCV), October 2017.

[28] Huajian Liu, Sang-Heon Lee, and Javaan Singh Chahl, “Transformation of a high-dimensional color space for material classification,” Journal of the Optical Society of America A 34(4):523 2017.

[29] Dongmei Mo, W. Wong, Xianjing Liu, Yao Ge, “Concentrated hashing with neighborhood embedding for image retrieval and classification,” In International Journal of Machine Learning and Cybernetic, 03 January 2022.

[30] A. Alzu'bi, A. Abuarqoub, “Deep learning model with low-dimensional random projection for large-scale image search“, Engineering Science and Technology, Pages 911-920, August 2020.

[31] L. Fei-Fei and P. Perona, “A bayesian hierarchical model for learning natural scene categories,” in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005, pp. 524–531.

[32] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” 2009.

[33] Jian Zhang, Yuxin Peng, and Zhaoda Ye. “Deep reinforcement learning for image hashing.” arXiv preprint arXiv:1802.02904, 2018.

[34] Wang, Z., Hong, W., and Yuan, J. “Deep reinforcement learning with label embedding reward for supervised image hashing”. arXiv preprint arXiv:2008.03973 (2020)

[35] Richard S Sutton and Andrew G Barto. “Reinforcement learning: An introduction.” 1998.

[36] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”, arXiv:1712.01815, 5 Dec 2017.

[37] Xin Yuan, Liangliang Ren, Jiwen Lu, and Jie Zhou. “Relaxation-free deep hashing via policy gradient”. In ECCV, 2018.

[38] A. Krizhevsky, I. Sutskever, G.E. Hinton, “ImageNet classification with deep convolutional neural networks”, in: Proceedings of the NIPS, 2012, pp. 1097–1105.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv preprint arXiv:1707.06347, 2017.

[40] T.S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, Y.T. Zheng, “Nus-Wide: a real-world web image database from national university of Singapore”, in: ICMR, 2009, pp. 1–9.

[41] T.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan P. Dollar, C.L. Zitnick, “Microsoft COCO: common objects in context”, in: Proceedings of the ECCV, 2014, pp. 740–755.

[42] Y. Cao, B. Liu, M.S. Long, J.M. Wang, “HashGAN: deep learning to hash with pair conditional Wasserstein GAN”, in: Proceedings of the CVPR, 2018, pp. 1287–1296.

[43] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. “Supervised hashing for image retrieval via image representation learning,” In AAAI, pages 2156–2162. AAAI, 2014.

[44] H. Lai, Y. Pan, Y. Liu, and S. Yan. “Simultaneous feature learning and hash coding with deep neural networks,” In CVPR. IEEE, 2015.

[45] F. Shen, C. Shen, W. Liu, and H. Tao Shen, “Supervised discrete hashing,” in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 37–45.

[46] Z. Cao, M. Long, J. Wang, and P. S. Yu, “HashNet: Deep learning to hash by continuation,” in The IEEE International Conference on Computer Vision (ICCV), Oct 2017.

[47] P. Indyk and R. Motwani. “Approximate Nearest Neighbor-Towards Removing the Curse of Dimensionality. In Proceedings of the 30th Symposium on Theory of Computing”, 1998, pp. 604-613.

[48] Yunhao Tang, Shipra Agrawal “Discretizing Continuous Action Space for On-Policy Optimization.” arXiv:1901.10500v4 [cs.LG] 19 Mar 2020.

[49] Chloe Ching-Yun Hsu Celestine Mendler-Dünner Moritz Hardt. “Revisiting Design Choices in Proximal Policy Optimization”, arXiv:2009.10897v1 [cs.LG] 23 Sep 2020.