Abdur Rafay Saleem
Department Of Computer Science
ab464825@ucf.edu

***Abstract*—This report conducts a thorough review of two papers that introduce improved deep hashing techniques and optimizations for more precise and faster multimedia retrieval. It will comprise four sections: a research problem and the papers' technical contributions to address it, technical details of both new and old techniques, and a critical analysis.**

I. RESEARCH PROBLEM

In recent years, multimedia retrieval has become increasingly important due to the explosive growth of digital content. Efficient techniques for searching and retrieving relevant information from large-scale datasets are crucial for various applications. This comes with some challenges, like fast query processing, substantial storage requirements, and different varieties of media types. Every media type has its structure and meaning; thus, it needs a standardized representation that is easier to store and faster to query across different types.

Hashing methods have gained popularity in this context. They enable compact binary code representation of data, leading to reduced storage requirements and faster similarity search. Several optimizations and extensions of hashing methods have been made to improve the results of multimedia retrieval. However, they are limited because they need user-supplied configurations and can't learn from the features. To solve this, research has been conducted into developing hashing methods that employ deep supervised learning to identify non-linear patterns in data space and form more accurate data clusters. They are more precise than data-independent methods. Among these are methods that perform end-to-end deep hashing by learning binary codes in one stage. This allows for the better utilization of semantic labels in the media and improves performance. However, these methods produce high processing costs and require large hardware memory. Hence, these methods typically select a subset of the training data to update the network in each iteration instead of training on the whole data. However, due to this, some labeled information is not fully utilized. As a result, the obtained binary codes face a loss of helpful information, and the model cannot discriminate effectively.

The first paper focuses on the limitations of existing deep-supervised hashing methods. It introduces a method of hashing that utilizes the graph structure of the data for learning the hashcode. It proposes a deep model that integrates deep supervised learning with optimized hashing of binary code using an anchor graph. The model considers the media samples as anchors. It applies a regression formulation to connect these anchors to all binary codes. This way, the model can obtain the exact binary codes of the training set without any quality loss.

However, the problem of updating the model with newer images still needs to be solved. As the Internet grows, more semantic information is available, and the previous hashcodes need to be recalculated to reflect this. Rehashing existing database media along with additional newer media increases the time costs. The second paper introduces a framework for learning the hashcodes in increments. It learns the query, newer images, and previous hashcodes in such a way that not only are the results obtained, but the hashcodes are also updated without loss. This removes the need to regenerate hashcodes and saves time. It also applies an improved loss function during training to maintain media similarities.

The technical contributions of both papers include efficient hashing frameworks that cover the gaps in existing solutions. They provide sufficient theorems as proof and present algorithms as part of their optimization process. Furthermore, they conduct extensive experimentation against solid benchmarks to test and demonstrate the effectiveness of the results against current state-of-the-art models.

II. FIRST TECHNIQUE

Deep Anchor Graph Hashing (DAGH)

This technique, introduced in the first paper, uses the anchor graph to perform deep supervised hashing. Instead of calculating the affinity matrix, it uses anchor connections among datasets to compute similarities between data points in the local structure.

Problem Formulation

The DAGH framework introduces a joint optimization problem with three main components: a regression term, a deep hashing term, and an efficient binary code learning term.

1. **Regression term**: The previous deep hashing techniques use an affinity matrix to calculate the distances between the data points. However, since they perform the training in small batches, they cannot get binary codes for all training data. Moreover, they perform direct regression of deep features to hashcodes, which takes a lot of time. To solve this, the DAGH technique treats samples as anchors and uses a new regression formula to create connections between them. This new regression term helps create an anchor graph to capture the local structure. Binary code balancing is done by keeping the constraints of having equal 1 and -1 in each row. The anchor graph is defined based on the class labels, ensuring that data points from the same class are connected. By solving the regression term, discrete solutions are obtained efficiently. During training, the DAGH framework uses mini-batch data to update the neural network. On every run, the partial batch of data uses connections established by the anchor graph to approximate interrelated binary codes. The network is better at distinguishing because it receives feedback from the binary codes of the entire training set, not just the mini-batch. This term differs from other techniques because it enables the simultaneous learning of binary codes and updating the deep neural network in a single framework. This framework includes two more components to improve the quality further.
2. **Deep Hashing term**: The deep hashing term is introduced to impose a loss function that converges features learned from neural networks towards the final hashcodes. It maximizes the inner product of deep feature vectors belonging to the same class and minimizes the inner product of vectors from different classes. This results in the same class items being closer in the binary space, and hence, the similarity is maintained.
3. **Efficient Learning Term for Hashcodes**: To improve the binary codes, the authors use something called the "label matrix." The label matrix contains information about each training data point's class. The authors want to use this label matrix to help create the binary codes. They do this by learning a special matrix called the "regression matrix." The regression matrix makes the binary codes bit by bit, using the information from the label matrix. The idea is that data points from the same class should have similar binary codes. To make the binary codes even better, the authors add a special rule called a "constraint." This rule ensures that each bit in the binary code has an equal number of -1s and 1s across all the training data.

Overall, the DAGH framework combines these three terms, weighted by coefficients, and includes constraints to ensure the binary nature and balance of the generated codes.

The Optimization

The optimization of the DAGH framework involves alternately solving for three variables: the regression matrix (W), the binary codes (B_{all}), and the deep features (U):

1. **Solving for W**: Given the binary codes and deep features, the subproblem for W reduces to a linear regression problem. The optimal solution for W can be obtained in closed form using the label matrix and the binary codes.

2. **Solving for B_{all}**: Given W and U, the subproblem becomes a series of regression problems with discrete constraints. The optimal codes are obtained by maximizing a trace function and applying a thresholding rule based on the largest elements of a transformed matrix.
3. **Solving for U**: Given W and B_{all}, the subproblem for U involves minimizing the regression and deep hashing terms. The neural network is updated using back-propagation, where the gradients are computed based on the differences between the deep features, the linear combinations of the binary codes, and the pairwise similarity constraints.

The optimization algorithm iterates between these three steps until convergence or a maximum number of iterations is reached.

The Algorithm

The following steps explain the workings of the DAGH framework algorithm:

1. The algorithm starts by initializing all the binary codes as a random binary matrix.
2. Then, it begins the main iteration loop, which runs T1 times. In each iteration:
    a. It calculates the regression matrix using the label matrix and the binary codes.
    b. It randomly selects partial training data to create the anchor graph, representing the connections between the selected data points and all the binary codes.
    c. It starts an inner loop that runs T2 times. In each inner iteration:
        i. It selects a subset from the training data.
        ii. It computes the deep features of the mini-batch using a neural network.
        iii. It calculates the weights based on the similarity between the deep features.
        iv. It updates the neural network using a derived equation that considers the deep features, the weights, and the binary codes.
    d. After the inner loop, it updates the binary codes using an equation that considers the label matrix and the deep features.
3. Finally, the algorithm outputs the trained hashing function and the hashcodes.

The neural network receives feedback from the binary codes through the anchor graph in each mini-batch iteration. This feedback helps the network learn better features by considering the information from all the training samples, not just the mini-batch. It tries to improve the quality of the binary codes by using deep features and the label matrix together. It uses all available information (deep features, binary codes, and labels) to learn better binary codes for retrieval tasks.

Binary Code Learning

Once the DAGH algorithm has finished training, it produces a deep neural network that has learned to generate good binary codes. This network generates the hash codes for new, unseen testing samples. Every sample from the testing dataset is passed into the network and outputs a continuous value. A sign function is used to convert this into binary codes.

III. SECOND TECHNIQUE

Deep Incremental Hashing Network (DIHN)

This technique, introduced in the second paper, uses an incremental learning methodology to update the image retrieval model with newer information. Instead of recomputing old hashcodes and re-training the model with all data, it adds knowledge from the incremental images to old ones. It uses both in a combined way to process query images. This incremental learning approach significantly reduces the training time compared to retraining the model from scratch.

Problem Definition

We already have a set of images D in the database associated with specific class labels in set L. These images have already been hashed using deep learning to produce binary codes in set B. Then, a set of new images, D', is added to the database. Each new image is associated with a new set of labels, L'. The aim is to hash these new images and produce a set of new binary codes, B'. The new and old binary codes will have the same length, k, and the old ones should be unaffected. There is also a set of images received from the user query, Q, which contains both old and new images. Next, we want to train a CNN to generate hashcodes for this query set. We provide additional information during training to help CNN learn hash codes better. This information is called "pairwise supervised information" and is represented by a matrix S. Each entry in this matrix tells us whether a database image and a query image are semantically similar. This matrix is divided into two parts. The first part contains semantic similarities between the old database and query images. The remaining part contains semantic similarities between the newly added database and query images. This division helps the CNN learn hash codes that preserve the semantic similarities between the query and old and new database images.

Framework Overview

In real-world situations, the query images are usually not available during training. To overcome this, the proposed framework, DIHN, selects a subset of images from the original and incremental databases. DIHN has two components that enable incremental hashing. The first component is responsible for producing hashcodes for query images. It extracts feature vectors from the query images using CNN and then makes their binary codes using deep hashing functions. The second component is responsible for the incremental images. It directly applies hashing to the newer images without a CNN to produce their hashcodes. DIHN uses a specially designed loss function called the incremental hashing loss function to ensure that the generated hash codes are effective for image retrieval. This loss function helps preserve the semantic similarities between the query and database images (both original and incremental) in the hash code space.

Deep Hash Functions

DIHN uses a CNN to learn hash functions that convert images into compact binary hash codes. Here's how it works:

1. The CNN model takes an image as input and processes it through multiple layers, including convolutional, pooling, and fully connected layers.
2. The last fully connected layer of the CNN is designed to have an output size equal to the length of the binary hash codes.
3. The output of the last fully connected layer is a real-valued vector. It contains the query image and the CNN model's learnable parameters.
4. A sign function is applied element-wise to convert the real-valued output to binary hashcode. The sign function maps positive values to +1 and negative values to -1.

Incremental Hashing Loss

Similar images should have similar hash codes, and dissimilar images should have different hash codes. The paper uses the pairwise label matrix S to achieve this, which tells us whether two images are similar or dissimilar.

An incremental loss function is defined to measure the difference between the similarity predicted by the hash codes and the true similarity given by the pairwise labels. It consists of two main parts:

1. The first part deals with the images from the original database and the query. The inner product of their hash codes is computed and compared with the actual similarity for each pair of database and query image. The difference is squared and summed over all pairs.

2. The second part is similar to the first one but deals with incremental database images instead of original ones.

There are two ways of representing each query image's hash codes. The first is a fixed binary hash code from the database set. The second is a hashcode obtained by applying the tanh function to the output of the last layer of the CNN. Tanh is used because the sign function is discontinuous and complex to optimize. Ideally, these two representations should be as close as possible for each query image. Two regularization terms are also added to the loss function:

1. The first term measures the squared difference between the two representations for each query image and sums it over all the query images. The hyperparameter λ controls the strength of this regularization. This term is minimized to bring both representations closer.
2. The second term encourages balanced hash codes with equal 1s and -1.

Optimization/Learning Algorithm

The optimization process alternates between two steps. The θ-step fixes the new hash codes and updates the CNN parameters using back-propagation to minimize the loss function. In B'-step, the CNN parameters are fixed, and the hash codes B' are updated using the discrete cyclic coordinate descent (DCC) algorithm. Hash codes are updated bit by bit, minimizing loss for each bit while keeping the others fixed. Repeat these two steps until convergence or max iterations are reached.

Original Hash Code Learning

DIHN is flexible and can use any existing hashing method to learn the original hash codes. The quality of the original hash codes affects DIHN's performance, so it's best to use state-of-the-art deep hashing methods for better results.

IV. CRITICAL ANALYSIS

Both DAGH and DIHN techniques address critical challenges in multimedia retrieval. DAGH's main strength is utilizing the entire dataset with a lower time complexity than other approaches. On the other hand, DIHN's key advantage is its incremental learning capability. It allows efficient model updates when new images are added to the database. This makes it more suitable for real-world scenarios where the image database is constantly growing. It significantly reduces the computational cost and training time compared to retraining the entire model from scratch.

Despite their strengths, both have some limitations. DAGH's performance depends on the quality of the anchor graph construction, which may be sensitive to the choice of hyperparameters and the presence of noisy or irrelevant anchors. Investigating advanced techniques, such as adaptive anchor selection or multi-level anchor graphs, could improve retrieval performance and robustness. Additionally, its computational complexity may become high when dealing with large-scale datasets. Developing more efficient optimization algorithms or parallelization strategies could help with this scale. In contrast, DIHN's incremental learning approach may lead to suboptimal binary codes, as it only updates the new codes.

Moreover, the quality of the original fixed hash codes, which are generated using existing hashing methods, influences its performance. Also, exploring approaches to update the existing and new hash codes parallelly could lead to better performance. This could involve designing adaptive weighting schemes or incremental fine-tuning of the deep feature representations.

Overall, the authors have significantly contributed to multimedia retrieval by using their techniques to address the key challenges. They have carefully conducted experiments against state-of-the-art techniques and proved the effectiveness of their methods over them. They also provide sufficient algorithmic proofs to sustain their arguments. While each method has its strengths and limitations, they provide valuable insights and pave the way for future research in developing more efficient, effective, and adaptable image retrieval systems.