EECS 545 Final Report: Implementing Deep Hashing for Compact Binary Codes Learning Implementation Track: 4 Team Members

Anonymous Author(s)
Affiliation
Address
email

Abstract

Image hashing is the one-way process of converting an image into a binary hash such that similar images have similar hashes. This has promising applications in speeding up approximate nearest neighbor search when trying to retrieve similar images from a database as well as in security for verifying an image hasn't been perceptually modified. Utilizing deep learning, we implement a model that learns these binary hashes under three primary constraints. First, we minimize the loss in information between the continuous model output and the quantized binary hash. Second, we make sure the binary values are distributed evenly on each bit. Third, we ensure different bits are as independent as possible through a relaxed orthogonality constraint on each fully connected layer of the model. In addition, we implement a variant of the same model that takes advantage of training data labeled for classification tasks in order to generate hashes that are near one another for images of the same class and far away for images of different classes. We evaulate the supervised and unsupervised variants of this model on the MNIST and CIFAR-10 datasets, as was done in the original paper, as well as a recent malaria diagnosis dataset from the NLM.

1 Introduction

Image hashing is the process converting an image into a short binary code called a hash, which in theory encodes the most important perceptual information about the associated image. Since hashes are highly space-efficient and computationally simple to compare, they have many applications in the domain of image retrieval. For instance, when searching a database of images for an entry that is most similar to a query image, it would be much faster to compare the hash of each image rather than look at every pixel. Some form of distance metric, most commonly the Hamming Distance, is used to determine the degree of similarity between hashes. Image retrieval applications motivate multiple important properties of hashes, including reduced size, which enables low search latencies, parametric computation, which makes it easy to obtain the hash of a new image, and finally, a good correspondence between inter-hash distance and perceptual similarity of the corresponding images.

One prominent approach to image hashing for retrieval tasks is Learning-Based Hashing (LBH), where labeled or unlabeled training data is leveraged in order to learn a series of hash functions that convert a given image to a hash with the desirable properties described above. Many existing LBH methods are linear and cannot capture the nonlinear, diverse structure of images. The goal of our project is to implement a state-of-the-art, non-linear LBH model (1) which utilizes deep neural networks to improve on past approaches. We feel it is interesting as it has critical applications to multiple relevant tasks, including retrieval of similar images and verification of image integrity. This model represents a nontrivial contribution to research on image hashing as it achieves better results than previous models on many image similarity benchmarks. Additionally, the paper describing the model introduces a novel objective function for image hashing that performs well in both supervised and unsupervised settings. As this is an implementation project, the novelty portion would be the investigation of a model that has a novel approach towards image hashing and provides state-of-the-art accuracy.

2 Related Work

2.1 Overview

Existing methods for image hashing can largely be broken down into two sub-categories: data-independent and data-dependent (1), (2). The former do not make use of training data, instead employing random linear projections, where as the latter leverage training data to learn useful transformations for hashing. One prominent data-dependent hashing algorithm is Locality Sensitive Hashing (LSH), which we explain below, and its many extensions (1), (3). Data-dependent hashing algorithms include Spectral Hashing (SH), Binary Reconstructive Embedding (BRE), Iterative Quantization (ITQ), K-Means Hashing (KMH), Minimum Loss Hashing (MLH), and Sequential Projection Learning Hashing (SPLH) (1). We will go into further details on a subset of these below. Additionally, we will briefly address one of the first applications of deep learning to image hashing in an algorithm called Semantic Hashing. The inventors of the method we implement in this project compared their algorithm to six unsupervised and three other supervised state-of-the-art hashing methods in their paper (1). The methods are the following: PCA-ITQ, KMH, Spherical Hashing, SH, PCAH, LSH, SPLH, MLH, and BRE, where the former six are unsupervised methods and the latter three are supervised.

2.2 Data-Independent Hashing

One of the most foundational data-independent hashing methods is LSH, which is designed specifically for retrieval tasks (3). For a set of points $P \subset \mathbb{R}^d$ under the l_1 norm, LSH begins by approximating each $p \in P$ as a binary vector by first multiplying all p by some constant and translating such that all of p's coordinates are nearly positive integers, then converting each integer coordinate $z_{i,j}$ into a binary string with $z_{i,j}$ ones followed by $(C = \max\{z_{k,l} | z_{k,l}$ is the l'th coordinate of $p_k \forall p_k \in P\}) - z_{i,j}$ zeros, and finally concatenating these binary strings together to form a binary representation for p, which we will call p'. Note that the hamming distance between any two p' is the same as the l_1 distance between the corresponding p's except scaled by the constant we used when making the coordinates of all p fall on near-integer values. Subsequently, LSH entails building l hashing functions for some $l \in \mathbb{N}$, each corresponding to a set $I_k \subset \{1, 2, \ldots, C\}, 1 \leq k \leq l$. Then, the hash function g_k maps p' to its bits in positions specified by the set I_k. Ultimately, when comparing data points for a retrieval task, given a query q, the authors of the paper (3) show that only comparing q to $p \in P$ for which some $g_k(p) = g_k(q)$ speeds up computation by lessening the number of necessary comparisons without a substantial decrease in precision. This method may also be kernelized and performed without explicitly computing the lengthy binary vectors for p'.

2.3 Data-Dependent Hashing

Many different data-dependent hashing methods have been proposed in the literature. An Iterative Quantization method called PCA-ITQ was proposed in (4). In this work, the authors describe a hashing method in which data is first projected into a lower-dimensional space via PCA and then the projected data is rotated so as to minimize its quantization error. They also show this method surpasses the performance of other PCA-based binary coding schemes. Another data-dependent hashing method called K-means Hashing (KMH), is proposed in (5). This method leverages a novel version of K-means clustering to assign data points to centroids in the conventional manner while simultaneously learning distance-preserving binary representations for each centroid. The algorithm is optimized through the expectation maximization algorithm with an added loss term in the M-step that helps to develop the binary representations for each centroid (5). As with LSH, KMH does not learn a unique hash for each data point, instead opting to divide the data into discrete buckets with binary codes for each bucket that may be compared to one another via hamming distance to determine the similarity or difference between buckets.

We will now summarize a few remaining data-dependent hashing methods in greater brevity. One hashing method called Binary Reconstructive Embedding (BRE) from 2009 is based on minimization of the difference between the original distances between data points and Hamming distances of binary codes (6). Another similar hash learning method known as Minimal Loss Hashing (MLH) maps high dimensional data to binary codes using a model based on structural support vector machines (SVMs) (7). The loss function used in this paper is similar to the hinge loss function used in SVMs, and encourages hashes to be near one another for similar training data points, like in BRE. Finally, in (8) a hashing method called Sequential Projection Learning for Hashing (SPLH) is proposed. In this work, supervised and unsupervised hashing methods are implemented to learn hash codes using linear mapping that handles similarity within the data.

2.4 Deep Hashing

As explained in (1), the earliest application of deep learning techniques for hashing is said to be semantic hashing. In this early method, Salakhutdinov and Hinton (9) used stacked Restricted Boltzmann Machines that learn compact binary codes for image classification. This model was not proven to be practical due to cost efficiency and complexity issues. The paper we implemented proposes a novel and more functional approach to deep hashing.

3 Problem Framing

3.1 Linear Learning Based Hashing

We assume $[x_1, x_2, \dots, x_n] \in \mathbb{R}^{d \times N}$ is the training set with N samples and $x_n \in \mathbb{R}^d$ is the nth sample. A series of hash functions are used to map and quantize each sample to a binary vector of size K. Therefore each x_n is mapped to a $K-$ bit binary vector as $b_n = [b_{n_1}, b_{n_2}, \dots, b_{n_K}] \in [-1, 1]^{K \times 1}$. Each element of b_n is calculated as follows.

$$b_{n_k} = f_k(x_n) = \mathrm{sgn}(g_k(x_n)) = \mathrm{sgn}({w_k}^T x_n)$$

where f_k is the k^{th} hashing function, sgn is the sign function and $w_k \in \mathbb{R}^d$ is the projection vector of the k^{th} hashing function. Assuming $W = [w_1, w_2, \dots, w_K] \in \mathbb{R}^{d \times K}$, binary codes of the image samples can be calculated as

$$b_n = \mathrm{sgn}(W^T x_n)$$

The goal of such a linear hashing algorithm would be to learn a matrix W that results in a succinct, informative hash. In the next sections, the nonlinear methods of deep hashing and supervised deep hashing are explained for generating binary code vectors from sample images.

4 Model

4.1 Unsupervised Deep Hashing

Now, we begin to describe the algorithm we actually implemented, as proposed in (1). In this method, each image sample is passed through M layers of a deep neural network. For each layer $1 \leq m \leq M$ of the network there are p^m units (this is not an exponent, merely a superscript for indexing purposes). For any sample image x_n the output of layer 1 is equal to:

$$h_n^1 = f(W^1 x_n + c^1)$$

where $W^1 \in \mathbb{R}^{p^1 \times d}$ and $c^1 \in \mathbb{R}^d$ are the linear transformation matrix and bias vector of the first layer and f is a nonlinear activation function such as tanh, relu, or sigmoid. The output of the m^{th} layer is equal to:

$$h_n^m = f(W^m h_n^{m-1} + c^m)$$

The binary vector output of the M^{th} layer for each image sample of x_n is equal to:

$$b_n = \mathrm{sgn}(h_n^M)$$

where sgn is the sign function, which we use to quantize h_n^M into bits. We denote $B = [b_1, b_2, \dots, b_N]$ and $H^M = [h_1^M, h_2^M, \dots, h_N^M]$ to be the matrix of binary codes and the output of M^{th} layer for all sample images respectively. Both matrices belong to $\mathbb{R}^{p^M \times N}$. The following loss function (LF) based on the three given criterion is used to learn the parameters of the network.

$$\min_{\mathbf{W},\mathbf{c}} LF = \frac{1}{2}\left\|B - H^M\right\|_F^2 - \frac{\lambda_1}{2N} tr(\tilde{H}^M(\tilde{H}^M)^T)$$
$$+ \frac{\lambda_2}{2}\sum_{m=1}^{M}\left\|W^m(W^m)^T - I\right\|_F^2 + \frac{\lambda_3}{2}\sum_{m=1}^{M}(\|(W^m)\|_F^2 + \|(c^m)\|_2^2)$$

Note that $\tilde{H}^M = H^M - \bar{H}^M$ where $\bar{H}^M$ is the row-wise mean of H^M, or rather the mean of H^M for each bit. In the loss function, the first term minimizes the quantization error between the output of the network and binary vectors for all image samples. The second term maximizes the variance of output binary vectors to encourage even distribution of the binary bits. The third term maximizes the orthogonality of the weight matrices to make them as independent as possible. The last term is a standard l_2 regularization term to control the scale of the parameters.

We use gradient descent optimization to minimize the loss function, requiring that we take the derivative of LF with respect to W^m and c^m for $m = 1, \ldots, M$. We then update the parameters according to the following formulas.

$$W^m = W^m - \eta\frac{\partial LF}{\partial W^m}$$
$$c^m = c^m - \eta\frac{\partial LF}{\partial c^m}$$

where η is the step size and the derivatives are calculated as follows.

$$\frac{\partial LF}{\partial W^m} = \Delta^m(H^{m-1})^T + \lambda_2 W^m(W^m(W^m)^T - I) + \lambda_3 W^m$$

where

$$\Delta^M = (-(B - H^M) - \lambda_1 H^M) \odot f^{'}(Z^M)$$
$$\Delta^m = ((W_1^{m+1})^T\Delta^{m+1}) \odot f^{'}(Z^m)$$

where $\odot$ denotes element-wise matrix multiplication and $Z^m = W^m H^{m-1} + c^m$.

4.2 Supervised Deep Hashing

The authors of the paper we implement also propose a supervised variant of their model that can learn from class-labeled training data to improve its hashing. The model architecture is exactly the same as the unsupervised version, with the one exception being an additional loss term in LF (1). The new loss term also requires input data of a slightly different format which takes advantage of the class labels in training data. With the goal of performing well on image retrieval tasks, the model utilizes "positive samples," or pairs of images belonging to the same class, and "negative samples," which are pairs of images belonging to different classes. Formally, pick $N_S, N_D \in \mathbb{N}$, which will be the number of positive and negative samples respectively to train on for each training iteration. We then sample from the training images to build S, the set of pairs of training samples (x_i, x_j) that belong to the same class, and D, the set of pairs of training samples (x_i, x_j) that belong to different classes. With the aforementioned setup, we minimize the following loss function:

$$\min_{\mathbf{W},\mathbf{c}} LF = \frac{1}{2}\left\|B - H^M\right\|_F^2 - \frac{\lambda_1}{2N} tr(\tilde{H}^M(\tilde{H}^M)^T) + \alpha tr(\Sigma_B - \Sigma_W)$$
$$+ \frac{\lambda_2}{2}\sum_{m=1}^{M}\left\|W^m(W^m)^T - I\right\|_F^2 + \frac{\lambda_3}{2}\sum_{m=1}^{M}(\|(W^m)\|_F^2 + \|(c^m)\|_2^2)$$

where

$$\Sigma_B = \frac{1}{N_D} tr((H_{d_1}^M - H_{d_2}^M)(H_{d_1}^M - H_{d_2}^M)^T)$$
$$\Sigma_W = \frac{1}{N_S} tr((H_{s_1}^M - H_{s_2}^M)(H_{s_1}^M - H_{s_2}^M)^T)$$

Here, $H^M_{s_1}$ is the model output when executed on the first training sample in each pair in S. Similarly, $H^M_{s_2}$ corresponds to the second sample of each pair in S, and $H^M_{d_1}$, $H^M_{d_2}$ represent the same values with respect to D instead of S. The extra term added in this loss function (compared to the one for unsupervised hashing), $\alpha tr(\Sigma_B - \Sigma_W)$, minimizes the intra-class variations and maximizes the inter-class variations in hashes. α is used to scale the impact of this term in the loss function. By using gradient descent optimization and taking the derivative of LF with respect to W^m and c^m for $m = 1, \ldots, M$ we update the parameters in a similar manner to the way we did for unsupervised deep hashing.

$$\begin{aligned}\frac{\partial LF}{\partial W^m} &= \Delta^m H^{m-1} + \frac{\alpha}{\lambda_1}(\Delta^m_{s_1} H^{m-1}_{s_1} - \Delta^m_{s_2} H^{m-1}_{s_2} - \Delta^m_{d_1} H^{m-1}_{d_1} + \Delta^m_{d_2} H^{m-1}_{d_2}) \\ &\quad + \lambda_3 W^m + \lambda_2 W^m (W^m (W^m)^T - I) \\ \frac{\partial LF}{\partial c^m} &= \Delta^m + \frac{\alpha}{\lambda_1}(\Delta^m_{s_1} - \Delta^m_{s_2} - \Delta^m_{d_1} + \Delta^m_{d_1}) + \lambda_3 c^m\end{aligned}$$

where the Δ terms for hidden layers and the top layer are calculated as follows:

$$\begin{aligned}\Delta^m_{s_1} &= ((W^{m+1})^T \Delta^{m+1}_{s_1}) \odot f'(Z^m_{s_1}), & \Delta^M_{s_1} &= \tfrac{1}{N_S}(H^M_{s_1} - H^M_{s_2}) \odot f'(Z^M_{s_1}) \\ \Delta^m_{s_2} &= ((W^{m+1})^T \Delta^{m+1}_{s_2}) \odot f'(Z^m_{s_2}), & \Delta^M_{s_2} &= \tfrac{1}{N_S}(H^M_{s_1} - H^M_{s_2}) \odot f'(Z^M_{s_2}) \\ \Delta^m_{d_1} &= ((W^{m+1})^T \Delta^{m+1}_{d_1}) \odot f'(Z^m_{d_1}), & \Delta^M_{d_1} &= \tfrac{1}{N_S}(H^M_{d_1} - H^M_{d_2}) \odot f'(Z^M_{d_1}) \\ \Delta^m_{d_2} &= ((W^{m+1})^T \Delta^{m+1}_{d_2}) \odot f'(Z^m_{d_2}), & \Delta^M_{d_2} &= \tfrac{1}{N_S}(H^M_{d_1} - H^M_{d_2}) \odot f'(Z^M_{d_2})\end{aligned}$$

4.3 Training Procedure

The training procedures for both the supervised and unsupervised variants of this model are very simple. We have already explicitly shown how the weights of the model are updated on each iteration. One aspect of the procedure that is important to note though, is the weight initialization. Following the methodology in the paper (1), for both supervised and unsupervised variants, we initialize the columns of W^1 to be the top p^1 eigenvectors of the data covariance matrix. All other weight matrices are initialized to the identity matrix with zero padding as needed to match dimensionality. The biases are all initialized to ones. After initialization, training proceeds via standard gradient descent as described above. The training process is terminated after the absolute change in loss from one iteration to the next falls below a certain threshold $\epsilon > 0$.

4.4 Evaluation Metrics

We evaluate both the supervised and unsupervised variants of the model on the same three metrics used in the original paper: Mean Average Precision (mAP), precision at N samples, and Hamming Lookup precision for a fixed radius r. mAP computes the area under the precision-recall curve for retrieval tasks. Precision at N samples simply measures what portion of the top N documents with hashes most similar (by hamming distance) to a query are of the same class as the query. The Hamming Lookup score measures a similar quantity, except instead of considering the closest N hashes to the query it looks at all images with hashes within hamming distance r of the query.

4.5 Variants

In our implementation we experimented with using convolutions instead of fully connected layers for the neural network. This was very easy to implement and did not alter the fundamental logic of the algorithm, so we thought it would be worth a try. Furthermore, given that the data involved are images, convolutions seemed like an obvious choice. However, we found that training times increased drastically when using convolutions, so we were unable to evaluate the performance of the model with this addition. We noticed that the loss decreased as the model trained, but we could not execute enough training iterations to get any meaningful evaluation results.

5 Methodology

We implemented the core model described in (1) entirely in Tensorflow. This allowed us to take advantage of its automatic differentiation capabilities, so we did not manually write out the gradient computations. We did specify the different layers of the network with custom losses and initializations and wrote our own training loop to ensure that updates to the weights were performed as specified above. For testing, we built Python programs to clean the datasets discussed in the following section and then prepare them for training and evaluation with the model. This mostly involved loading, saving, reshaping, and sorting the data. However, the paper used GIST feature vectors instead of raw pixels as inputs on some of the larger datasets, so we had to build infrastructure to convert RGB images to GIST vectors efficiently. Ultimately, we used a mix of our own code and sklearn functions to calculate the three evaluation metrics on the results of each dataset.

6 Performance Evaluation and Comparison

	Mean Average Precision (%)	Precision (%) at $N = 100$	Precision (%) at $r = 2$
MNIST	28.06	29.99	30.84
	37.59	38.83	42.72
CIFAR-10	12.67	12.55	13.18
	13.83	13.51	14.43
Malaria	51.38	52.17	52.03
	54.59	55.91	55.92

Figure 1: Our evaluation metrics for the model on the MNIST, CIFAR-10, and Malaria datasets. The top section corresponds to unsupervised models and the bottom results correspond to the supervised.

	Mean Average Precision (%)	Precision (%) at $N = 100$	Precision (%) at $r = 2$
PCA-ITQ	15.67	22.46	22.60
KMH	13.59	20.28	22.08
Spherical	13.98	20.13	20.96
SH	12.55	18.83	18.52
Semantic	12.95	14.79	11.49
PCAH	12.91	18.89	21.29
LSH	12.55	16.21	16.73
DH	16.17	23.79	23.33
DH Actual	12.67	12.55	13.18
SPLH	17.61	25.32	23.05
MLH	18.37	24.43	23.53
BRE	14.42	20.68	20.89
SDH	18.80	26.32	23.26
SDH Actual	13.83	13.51	14.43

Figure 2: Evaluation metrics from (1) for the CIFAR-10 dataset. The top section corresponds to unsupervised models and the bottom results correspond to the supervised. Bolded are the paper's results and our results.

We evaluate both the supervised and unsupervised versions of the deep hashing algorithm on three image classification datasets: CIFAR-10, MNIST, and a malaria diagnosis dataset from the NIH. For reference, MNIST has 10 distinct classes, CIFAR-10 also has 10, and the malaria dataset has 2. The metrics we computed with these datasets are the three described in the Evaluation Metrics section: mean average precision, precision at $N = 100$ samples, and hamming lookup precision for a radius $r = 2$. Just like the authors of the paper, we split each dataset into a gallery set, which is used to train the model, and query set, from which queries are sampled to compute evaluation metrics. Although the paper does not specify how exactly the two subsets of data are used, we infer that the gallery set was used for precision calculations as well as training, since it would otherwise be mathematically impossible to achieve the scores the original authors present given their small query set of 1000 data points. We, on the other hand, decided to use the smaller query set for precision calculations as well for the following reasons: 1) The computations could be performed much faster 2) In practice, it would be difficult to train the algorithm on every image it will ever need to retrieve 3) When we attempted to use the gallery set for precision calculations, we found it had a minimal impact on outcomes. For each model, we used layer sizes as given in the paper: $[60, 30, 16]$, yielding a hash size of 16. We also used the hyperbolic tangent function for layer activations, just like the original

	Mean Average Precision (%)	Precision (%) at $N = 100$	Precision (%) at $r = 2$
PCA-ITQ	41.18	66.39	65.73
KMH	32.12	60.43	61.88
Spherical	25.81	49.48	51.71
SH	26.64	56.29	57.52
PCAH	27.33	56.56	36.36
LSH	20.88	37.77	25.10
DH	43.14	67.89	66.10
DH Actual	28.06	29.99	30.84
SPLH	44.20	62.98	63.71
BRE	33.34	60.72	34.09
SDH	46.75	65.19	63.92
SDH Actual	37.59	38.83	42.72

Figure 3: Evaluation metrics from (1) for the MNIST dataset. The top section corresponds to unsupervised models and the bottom results correspond to the supervised. Bolded are the paper's results and our results.

authors. For other hyper-parameters, such as $\lambda_1, \lambda_2, \lambda_3$ in the loss function and the learning rate, we tried training the models multiple times on each dataset with different sets of parameters until we found some that worked well. The scores we report below are the best versions we managed to train for each model. Note that the results varied somewhat from one train of the model to another, but generally were within 10% of reported results. We also expand on our specific preprocessing and training procedures for each dataset after the results table.

6.1 MNIST

The MNIST dataset required minimal preprocessing before being sent as input to the model. We flattened each image into a 784-dimensional vector, and passed the entire 69000×784 gallery matrix to the model each iteration to compute the unsupervised loss. For the supervised variant, we generated 5000 pairs of images from the same class and 5000 pairs from different classes in order to calculate the supervised loss term. After much experimentation, we found that the weight and bias initializations used in the paper performed well, but the hyperparameters given by the paper did not. The set of hyperparameters that we found worked best, and which generated the results in the table above, were as follows: $\lambda_1 = 100$, $\lambda_2 = 0.1$, $\lambda_3 = 0.1$, and $\alpha = 3$. We also set our learning rate to .001 and trained for 300 epochs. Throughout the parameter tuning procedure, we found that hyperparameters and initialization methods had a very large effect on model performance, so we believe the difference in scores between our implementation and that of the paper may be partially attributed to imperfect hyperparameters. Furthermore, there were multiple typos and vague definitions in the paper in areas spanning the loss function definitions, scoring methodology, and hyperparameter choice, so it is quite possible that our implementation deviates from the original in these areas. Ultimately, our choice to use the query set for evaluation with a value of N as large as the number of total samples from each class should result in harsher evaluations than those given in the paper; that said, we believe our methods of evaluation are more relevant to real-world use cases than the original.

6.2 CIFAR-10

The CIFAR-10 dataset required some preprocessing before being sent as input to the model. First, each image was converted to grayscale and then converted into a 512-D GIST vector as per the paper. Then, the entire 59000 x 512 gallery matrix was similarly sent to the model each iteration to compute the unsupervised loss. For the supervised variant, 5000 intra and inter class pairs were generated just as in MNIST. Similar to MNIST, we found that the optimal combination of bias and weight intiializations was in fact a zero initalization for the bias as well as hyperparameters of $\lambda_1 = 1000, \lambda_2 = 0, \lambda_3 = 0, \alpha = 3$ with a learning rate of 0.01 trained for 100 epochs. In effect, the model performed better if the last two terms of the objective function were removed. This may have to do with the fact intra-class images within CIFAR often had images that were significantly different in terms of the positioning of each object and of their color, causing significantly different GIST vectors, however had common objects which required the same higher level processing. As a result, different hashing functions for images of the same class would be similar, and as a result, the λ_2 and λ_3 terms would run decrease prediction accuracy as they encouraged independence between hashing functions. With the results we generated, it can be seen that while the paper claims their model to give better measurements than most all the other baseline models, our results show that it in fact gives nearly the same results as all the other baseline models.

6.3 Malaria Dataset

The Malaria dataset, first used in (10), consists of 27558 images of blood smear slides produced at Chittagong Medical College Hospital, Bangladesh which are categorized by whether the cells are infected with malaria or not. This dataset required some preprocessing before being sent as input into the model. First, each image was resized with interpolation to a square using OpenCV since the dataset contained images with variable sizes and converted to grayscale. Then, the images were converted into 512-D GIST vectors, as in the original paper, and the entire 27358×512 gallery matrix was sent to the model each iteration to compute the unsupervised loss. In the supervised variant, we generated 5000 inter and intra class pairs as in the MNIST and CIFAR-10 datasets. The optimal initialization of the model for the malaria dataset was to initialize every bias to zero with the hyperparameters $\lambda_1 = 300, \lambda_2 = 0.1, \lambda_3 = 0.1$, and $\alpha = 3$. Additionally, we use a learning rate of $.001$ and trained for 100 epochs. Similar to the MNIST dataset, slight alterations of the initializations and choice of hyperparameters caused large changes in the model performance, so better metrics could be achieved with additional tuning. Furthermore, the original images range from 2200 to 123760 pixels so a sizable amount of information is potentially lost in the preprocessing. Hence, better model performance could be achieved by creating larger GIST vectors in the preprocessing to reduce the information loss.

7 Summary and Conclusions

Overall, we find the supervised and unsupervised deep hashing algorithms proposed by (1) to be moderately effective for image retrieval tasks. Both variants of the algorithm perform significantly better than a random retrieval system in all three metrics on every dataset we tested. The supervised variant performs slightly better than the unsupervised one, as should be expected given that it has more information with which to learn its embeddings. However, the scores that we achieve and even the higher ones reached in the paper are very far from the precision of a supervised classifier. There is likely substantial room for improvement on both the consistency and maximum precision of this model, at least for image retrieval tasks. In addition, the heavy reliance on good hyperparameter tuning and initialization makes this model difficult to work with and apply in varied settings. If there were heuristic rules for setting hyperparameters that would certainly help, but we did not notice any distinct patterns in how to set parameters for the three datasets. Regardless of performance on the evaluation datasets, we did confirm the authors' claim that their network trains and executes very quickly; we were able to train all of our models referenced here in a few minutes on home computers with minimal code optimization.

Similarly, while the paper says that their model produces much better results than other baseline models, our results have shown that for both CIFAR-10 and MNIST, their results are in fact similar to the other baseline models. As we discussed above, this could be due to our implementation details, but it also suggests that the paper's results may not be as reproducible as the authors claim, as we tried many variations on the original structure to no avail. One other interesting feature we noticed was the disparity in scores between a simpler dataset like MNIST and a more complex one like CIFAR-10. We believe this demonstrates that the learned hashes may not be very effective at encapsulating complex, higher order information about image content. The original paper indicates a similar disparity, so this is probably not just a feature of our implementation. This opens up an area for future research: is it possible to build hashing models that capture high-level features of complex images, even going so far as to be relevant to semantic tasks like segmentation or object recognition?

References

[1] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, “Deep hashing for compact binary codes learning,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[2] K. G. Dizaji, F. Zheng, N. S. Nourabadi, Y. Yang, C. Deng, and H. Huang, “Unsupervised deep generative adversarial hashing network,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3664–3673.

[3] A. Gionis, P. Indyk, and R. Motwani, “Similarity search in high dimensions via hashing,” *Proceeding VLDB ’99 Proceedings of the 25th International Conference on Very Large Data Bases*, vol. 99, 05 2000.

[4] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, “Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, 2013.

[5] K. He, F. Wen, and J. Sun, “K-means hashing: An affinity-preserving quantization method for learning binary compact codes,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[6] B. Kulis and D. Trevor, “Learning to hash with binary reconstructive embeddings,” in *23rd Annual Conference on Neural Information Processing Systems*, 2009.

[7] M. E. Norouzi and D. Fleet, “Minimal loss hashing for compact binary codes,” in *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[8] J. Wang, S. Jumar, and S.-F. Chang, “Semi-supervised hashing for large-scale search,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, 2012.

[9] R. Salakhutdinov and G. Hinton, “Semantic hashing,” *International Journal of Approximate Reasoning*, vol. 50, no. 7, 2009.

[10] S. Rajaraman, S. K. Antani, M. Poostchi, K. Silamut, M. A. Hossain, R. J. Maude, S. Jaeger, and G. R. Thomas, “Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images,” *PeerJ*, 2018.