2019 IEEE International Conference on Smart Internet of Things (SmartIoT)

An Effective Network Intrusion Detection Framework Based on Learning to Hash

Wenrui Zhou, Yuan Cao, Heng Qi[1], Junxiao Wang
School of Computer Science and Technology
Dalian University of Technology
Dalian, China

***Abstract*—Nowadays, the network intrusion detection has been an important issue in IoT. Although machine learning based methods seem to be promising in traditional network intrusion detection, these methods can hardly meet some demands of IoT. For example, unknown classes of flows are produced frequently in IoT, leading to classifiers training repeatly. To address this issue, we proposed a network intrusion detection framework based on learning to hash in this paper, which can reduce computation overhead significantly while avoiding frequent training of classifiers. The proposed framework consists of a hashing encoding module and an anomaly detection module with optimized *k*-NN classifier based on data distribution ratio. Moreover, the multi-index hashing is applied for fast and accurate search in Hamming space. Experimental results show that the proposed framework can detect various attacks and outperform the traditional intrusion detector.**

Keywords—network intrusion detection, learning to hash, k-nearest neighbors, similarity retrieval

I. INTRODUCTION

The increasing use of the network in various fields makes user privacy and the security of key data vulnerable to attack. Intrusion Detection System (IDS) [1] is provided to detect attacks on computer networks. Generally speaking, IDS is used to identify and judge any security vulnerabilities in computers or networks. In recent years, the commonly used intrusion detection techniques include: machine learning based techniques [2], and data mining based techniques [3] etc. The supervised and unsupervised intrusion detection techniques based on machine learning have been hot topics.

Existing work prefers to focus on supervised learning. These methods rely on training classifiers to achieve classification, even training neural networks to learn the characteristics of existing samples for classification [4-5]. However, the number of attacks on networks has been increasing over the years, leading to unknown classes of flows being produced frequently [6], especially in IoT. When a new type of intrusion traffic occurs, the classifier needs to be re-trained. Otherwise, the correct classification cannot be obtained, and it is necessary to ensure that the model is constantly updated manually in order to obtain an excellent detection effect.

To solve the above problems, some work [7] put forward to using "lazy" classification algorithm to avoid repeated training of intrusion detection classifiers, such as *k*-Nearest Neighbor (*k*-NN) algorithm [8]. The *k*-NN algorithm is a simple and effective supervised learning technique because it doesn't require classifier training. It assigns a class label to an unlabeled object based on the class labels of its k nearest neighbors. Consider a class-labeled dataset $X = \{x_1, x_2, \ldots, x_m\}, x_i \in R^N$ and an unlabeled query q. To predict the label of q, k-NN figures out the distance between q and all samples in X to decide the k nearest neighbors of X. Then q is labeled as per the majority class of its k nearest neighbors. As seen from the above procedure, it does not have an explicit learning process. Therefore, in the face of the ever-changing scenario of network traffic, the algorithm is very applicable. Even if new types of intrusion data are constantly added, the algorithm can correctly predict and classify. But because of the complexity of computing Euclidean distance, *k*-NN model based IDS cannot be widely used, and when the amount of data is huge, the storage and computing costs are very high. In this paper, we resort to the state of the art learning to hash method to convert network traffic data to binary codes before *k*-NN classification process to solve the above bottlenecks.

Learning to hash is a widely-studied solution to the approximate nearest neighbor search, especially for many large-scale analysis tasks due to its fast query speed and drastically reduced storage[9]. Hash coding aims to transforms the points in the database (high-dimensional vectors) into binary vectors consisting of 0 and 1, known as hash codes, while keeping the similarity between the points in the original space as far as possible. Specifically, given a sample $x \in R^D$, which is used to represent the vector with dimension D, the following mapping procedure is performed using the set of hash functions $H = \{h_1, \ldots, h_K\}$, that is, $h_K: R^D \rightarrow B$. A binary code with K bits $y = \{y_1, \ldots, y_K\}$ can be calculated for vector X, and y is a hash code of X. There are two main categories of hashing algorithms: data-independent hashing [10] and learning to hash [11]. Learning to hash, the interest of our paper, is a data-dependent hashing approach which aims to learn hash functions from a specific dataset so that the nearest neighbor search result in the hash coding space is as close as possible to the search result in the original space, and the search cost as well as the space cost are also small.

In this work, we propose a novel network intrusion detection framework based on learning to hash. Our specific contributions are: (i)In order to save lots of resources and computing power to train classifiers and avoid repeating training classifiers to detect unknown classes, a network intrusion detection framework based on learning to hash is proposed. This framework consists of online real-time data gathering, data preprocessing, hash coding, anomaly detection and output of detection results modules. (ii)We introduce an anomaly detection module with optimized *k*-NN classifier based on data distribution ratio to improve the detection

[1] Corresponding author: hengqi@dlut.edu.cn

978-1-7281-3488-8/19/$31.00 ©2019 IEEE
DOI 10.1109/SmartIoT.2019.00089

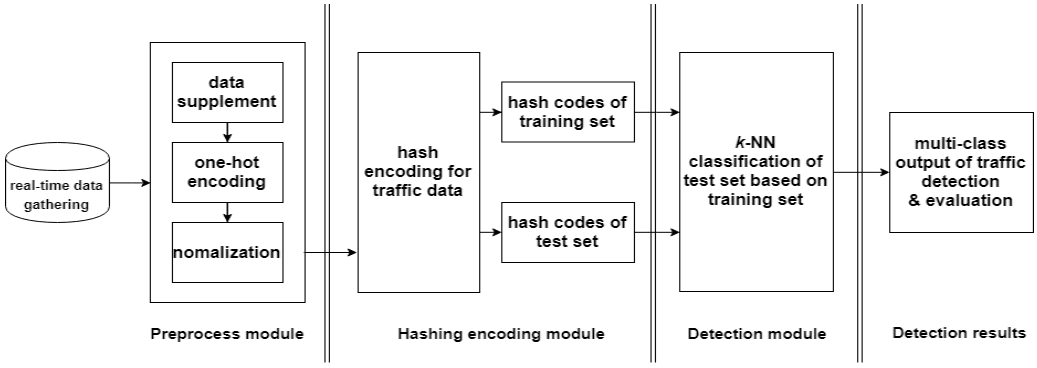


Fig. 1. Network intrusion detection framework based on learning to hash.

performance. The combination of multi-index hashing for fast and accurate search in Hamming space further accelerates the rate of intrusion detection. (iii)The solid experiments on NSL-KDD dataset show that the proposed method can detect various attacks and outperform the *k*-NN intrusion detector, especially the high-speed detection rate. This demonstrates that our optimization is feasible and promising for network intrusion detection.

The remaining of this paper is organized as follows. Section II introduces the proposed algorithm and implementation. Section III presents experimental results. Section IV concludes the paper.

II. Proposed Framework and Implementation

This section describes the proposed network intrusion detection framework based on learning to hash. We firstly introduce the basic framework, and then mainly explain hash encoding module and anomaly detection module according to their innovations.

A. Basic Framework

As illustrated in Fig. 1, it depicts the overview of detection procedure of our proposed framework, including online real-time data gathering, data preprocessing, hash coding, anomaly detection and output of detection results modules etc.

In this framework, we need to follow the steps below:

- Firstly, obtain network traffic data through the real-time data gathering module.
- Secondly, extract the statistical characteristics of network traffic and preprocess them.
- Thirdly, after obtaining the characteristic data, construct an unsupervised hash encoder for network traffic and get its hash codes.
- Finally, classify the hash codes by machine learning method, namely *k*-NN classification method, and obtain the detection results.

Notice that the preprocess module conducts three steps: data supplement, numerization and normalization. For the convenience of later data processing, we employ one-hot encoding for discrete features during the process of numerization. The hot-encoding method is how many values of the attribute are selected, then how many independent bit state registers are used to represent the state, only one bit is valid at any time. Normalization is to scale the data according to a certain method, so that the characteristic data can fall into a specific numerical space, so as to facilitate subsequent calculation. We adopt z-score normalization to convert data from different dimensions into the same dimension. The formula is as follows:

$$z = \frac{x-\mu}{\sigma} \tag{1}$$

Where μ and σ are the mean and standard deviation of the original data *x*, respectively.

The advantage of this framework is that the hash encoder is used to convert network traffic into hash code. At present, classification algorithms usually consume a lot of storage and computing resources in large data scenarios, while hash codes are binary vectors and usually only have tens to hundreds of dimensions, so they occupy very little space; usually, the comparison between binary codes can be completed only with a small number of machine instructions, which greatly reduces the computational resources. In addition, the *k*-NN algorithm is adopted in this method. For other machine learning classification algorithms, every time there is a new kind of traffic data, they need to be retrained. In the face of the ever-changing network environment, new types of attacks will continue to emerge, so these classification algorithms have poor scalability and they can only be used in the experimental stage. But for *k*-NN algorithm, there is no need for repetitive training, so it can avoid the problem of repetitive training classifier faced by other classification algorithms.

Next we'll focus on hash encoding module design and anomaly detection module improvement.

B. Hash Encoding Module Design

To avoid repetitive training in the framework, we still need to pay attention to the selection of hashing algorithm for hash encoding module. It needs to fulfill two requirements, that is, unsupervised and data-dependent (learning to hash). After comprehensive consideration, the hash encoding model in this paper uses the unsupervised RBA (Relaxed Binary Autoencoder) algorithm [12] proposed by Thanh-Toan Do to get the hash code of network traffic. This algorithm can train the hash encoding model according to the data itself without the need to provide labels. RBA algorithm solves the following optimization function:

$$\min_{\{W_i,c_i\}_{i=1}^{2}} J = \frac{1}{2}\|X - (W_2(W_1X + c_1 1^T) + c_2 1^T)\|^2 + \frac{\beta}{2}(\|W_1\|^2 + \|W_2\|^2) \tag{2}$$

$$s.t.\ W_1X + c_1 1^T \in \{-1,1\}^{L\cdot m} \tag{3}$$

Algorithm 1: Relaxed Binary Autoencoder (RBA)

Input: X: *training data; L: code length;*
T_1: *maximum iteration number; parameters* λ, β
Output: *Parameters* $\mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2$
Initialize $\mathbf{B}^{(0)} \in \{-1,1\}^{L\times m}$ *using* ITQ;
Initialize $\boldsymbol{c}_1^{(0)} = 0, \boldsymbol{c}_2^{(0)} = 0$;
for $t = 1 \to T_1$ ***do***
  Fix $\mathbf{B}^{(t-1)}, \mathbf{c}_1{}^{(t-1)}, \mathbf{c}_2{}^{(t-1)}$, *solve* $\mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}$
  Fix $\mathbf{B}^{(t-1)}, \mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}$, *solve* $\mathbf{c}_1{}^{(t)}, \mathbf{c}_2{}^{(t)}$
  Fix $\mathbf{W}_1^{(t)}, \mathbf{W}_2^{(t)}, \mathbf{c}_1{}^{(t)}, \mathbf{c}_2{}^{(t)}$, *solve* $\mathbf{B}^{(t)}$
end for
return $\mathbf{W}_1^{(T_1)}, \mathbf{W}_2^{(T_1)}, \mathbf{c}_1{}^{(T_1)}, \mathbf{c}_2{}^{(T_1)}$

Solving (2) under (3) is difficult due to the binary constraint, so the algorithm introduces a new auxiliary variable B and solve the following optimization:

$$\min_{\{W_i,c_i\}_{i=1}^2,B} J = \frac{1}{2}\|X - (W_2B + c_2 1^T)\|^2 + \frac{\lambda}{2}\|B - (W_1X + c_1 1^T)\|^2 + \frac{\beta}{2} \quad (4)$$

$$\text{s.t. } B \in \{-1,1\}^{L\cdot m} \quad (5)$$

The above optimization functions are solved by matrix derivation and coordinate descent algorithm. The input is training data set X, encoding length L, maximum iteration times T1 and parameters λ and β. The specific training steps are summarized in Algorithm 1.

From Algorithm 1, it can be found that the initialization method of RBA for intermediate variable B is ITQ [13], which is an iterative quantization hashing method. The problem is that the length of hash code for X cannot exceed the limit of original data dimension due to the inherent quality of ITQ. In this paper, in order to make the hash code length of the traffic data unrestricted, LSH [14] is used to replace the original initialization method of RBA hash algorithm.

C. Detection Module Improvement

In the anomaly detection module, the intrusion detection classification process based on machine learning algorithm needs to be implemented. This paper adopts *k*-NN (*k*-Nearest Neighbor) algorithm, which is a classification algorithm proposed by Hart and Over in 1968. It's a "no learning" algorithm, that is, it doesn't have an explicit learning process. Therefore, in the face of the ever-changing scenario of network traffic, the algorithm is very applicable. Even if new types of intrusion data are constantly added, the algorithm can correctly predict and classify.

But for the actual Internet environment, traffic data usually has the following characteristics: there are many types of abnormal traffic, but the number is far less than normal traffic data, that is, the distribution of normal traffic data and intrusion data is unbalanced. Most of the current algorithms do not take into account the imbalance of data distribution, but simply classify the data through the trained model, resulting in the detection effect is not optimal. In this paper, in order to solve the above problem, we propose an optimized *k*-NN classifier based on data distribution ratio to replace the original *k*-NN classification algorithm.

In the *k*-NN classification algorithm, the category with the most frequent occurrences in the *k* nearest neighbors in the training set is simply calculated as the category of test data. If a category has a high occurrence frequency, such as the normal category, then the category data with a very low occurrence frequency will be identified as the category with a high occurrence frequency. After the data distribution is taken into account, in *k*-NN classification method based on data distribution proportion, the category with the most frequent occurrences in the *k* nearest neighbor is not used as the category of test data. But according to the *k* nearest neighbor set of query point q, the proportion of the number of *k* nearest neighbors in each category to the total number of data in the training set is obtained, and the category with the highest proportion is identified as the category of test data points. This trick avoids the problem of poor detection results due to the uneven distribution of data.

D. Speedup

In the basic framework of intrusion detection, hash encoding has already speeded up intrusion detection to a great extent. In order to further accelerate the rate of detection, we employ the multi-index hashing (MIH) [15] in anomaly detection module. As an accurate *k*-NN search algorithm for binary encoding space, multi-index hashing is faster than exhaustive linear search. This algorithm shows excellent performance of sub-linear search time, saving storage space and simple execution. The basic idea of multi-index hashing is as follows:

- Firstly, decompose hash codes into non-overlapping *m* blocks, that is, *m* substrings, each with *s* bits. A hash table is needed for each binary substring.
- Secondly, use the idea of divide and conquer, for each block, the binary codes within the Hamming distance of *r'* are queried and used as candidate binary codes..
- Finally, the candidate binary codes of all blocks are combined to eliminate binary codes whose final Hamming distance is greater than *r*.

Specific steps can be found in [15]. Through incorporating multi-index hashing into our proposed framework, the detection rate can be speeded up to a large extent.

III. Experimental Results

To evaluate the performance of our proposed method, our experiments were performed using the NSL-KDD dataset [16]. The NSL-KDD dataset provides different training sets and testing sets, we used the training set KDDTrain$^+$20% and the testing set KDDTest$^+$ in our experiments. Data records consist of 42-dimensional features, of which 41-dimensional features are data features, and the last one is a class label. Table I lists the numbers of instances in the training set and the testing set. As seen from Table I, attacks in NSL-KDD are grouped into four types: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), Probe.

TABLE I. Number of Instances in NSL-KDD

Data sets	Amount					
	Total	***Normal***	***DoS***	***Probe***	***U2R***	***R2L***
KDD Train$^+$ 20%	25192	13449 53.39%	9234 36.65%	2289 9.09%	11 0.04%	209 0.83%
KDD Test$^+$	22544	9711 43.08%	7458 33.08%	2421 10.74%	200 0.89%	2754 12.22%

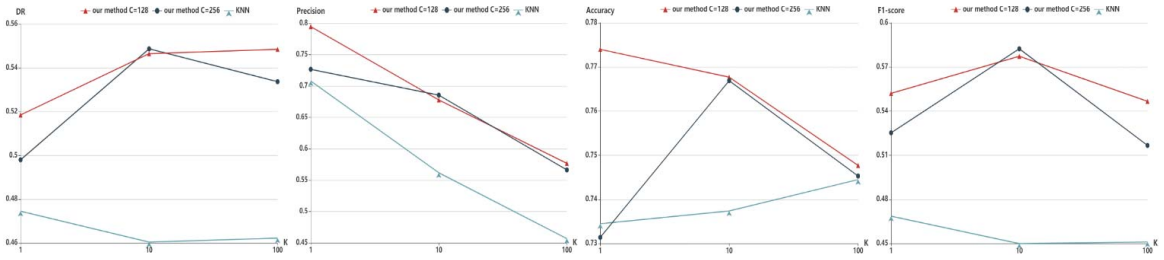


Fig. 2. Comparison of multi-classification evaluation indexes of our method and *k*-NN intrusion detector on NSL-KDD. (a) Detection rate. (b) Precision. (c) Accuracy. (d) F1-score.

TABLE II. CONFUSION MATRIX

		Predicted	
		Attack	***Normal***
Actual	***Attack***	TP	FN
	Normal	FP	TN

Four measures are used to evaluate the intrusion detection models, namely, precision, accuracy, detection rate (DR), and F1-score. The definitions of these measures are based on the confusion matrix, as shown in Table II. The formulas are as follows:

$$Precision = \frac{TP}{TP+FP} \tag{6}$$

$$Accuracy = \frac{TP+TN}{TP+FN+FP+TN} \tag{7}$$

$$DR = \frac{TP}{TP+FN} \tag{8}$$

$$F1-score = \frac{2*PR*DR}{PR+DR} \tag{9}$$

For the experiments we used Intel(R) Core(TM) i7-8700K CPU @3.70GHz, running on the ubuntu16.04.1 LTS.

A. Detection Performance

1) Overall detection performance: Our experiments are mainly multi-class experiments because they involve multiple types of attacks. We compare the performance of our method against *k*-NN intrusion detectors, and the results are depicted in Fig. 2. There are two parameters in our experiments: code length and number of nearest neighbors returned *k*. The experimental parameters are set as shown in the table III. The reason for choosing 128 bits and 256 bits is that the method under the code length performs best in the intrusion detection experiment before optimization.

As can be seen from the Fig. 2, the evaluation indexes of our method are all higher than those of directly using *k*-NN detector for intrusion detection classification, especially the F1-score index, which takes into account both detection rate and precision. At the best performance of our method, F1-score index is 13% higher than that of *k*-NN detector. In conclusion, the proposed method can detect various attacks and outperform the *k*-NN intrusion detector, so the network intrusion detection framework combining learning to hash and optimized *k*-NN classifier based on data distribution ratio is effective and desirable.

TABLE III. SETTINGS OF EXPERIMENTAL PARAMETERS

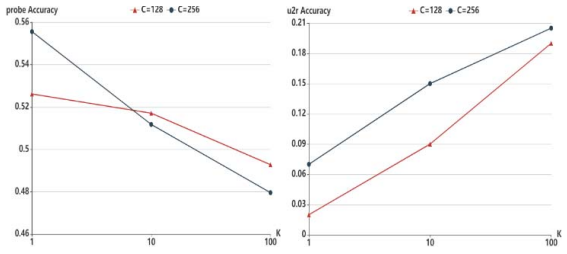
Experimental Parameters	**Value**
Code length	128bits and 256bits
Number of nearest neighbors returned *k*	1, 10 and 100

2) New attacks detection performance: The framework proposed in this paper has one advantage, that is, it can classify new arrival attacks types correctly even without repeated training. The idea of experimental verification is firstly to remove Probe and U2R two types of attack traffic of KDDTrain$^+$20% training set, and then use it to train RBA hash encoder to get model parameters. In the process of intrusion detection classification, the testing set KDDTest$^+$ remains unchanged, so the attack traffic of Probe and U2R in the testing set corresponds to the new types of attack traffic. When the traffic of new attack types appear, the hash encoder will not be trained again, and the traffic will be encoded directly using the original hash encoder. KDDTrain$^+$20% training set and KDDTest$^+$ testing set are input into the hash encoder for hash codes, and then the hash codes of testing set are classified by using *k*-NN classification model.

After setting the parameters, the experimental results are shown in Fig. 3. We mainly calculate the accuracy corresponding to the new types of Probe and U2R attacks in the experiment. From the results, we can see that the accuracy of the Probe attack type can basically reach more than fifty percent, while the accuracy of the U2R attack type is very low because it's very rare. So there is still a lot of room for improvement and optimization of the U2R attack type. It can be concluded that when a new type of attack appears, the supervised intrusion detection method can correctly detect the traffic of this type only after retraining process, while the proposed framework in this paper can detect the traffic of the new type without repeated training, which greatly reduces the time and resources consumed in the process of manual training model.

B. Time Performance

This part of the experiment is mainly to compare the intrusion detection classification time between our method and using *k*-NN intrusion detector directly, as shown in Fig. 4. It should be noted that the training time of hash encoder is offline, so it isn't included into the comparison time. As for parameters settings, the code length is set to 2^n ($3 < n < 9$), and the number of nearest neighbors returned *k*, is set to be 1, 10, 100, 1000, 2000, 5000, respectively.



(a) (b)

Fig. 3. Probe and U2R classification accuracy of new attacks detection experiments. (a) Probe accuracy. (b) U2R accuracy.

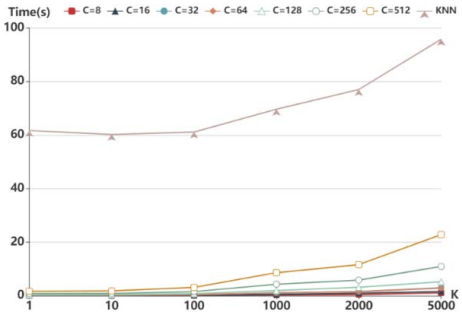


Fig. 4. Comparison of multi-classification time of our method and *k*-NN intrusion detector on NSL-KDD.

From Fig. 4, it can be seen that the classification time of intrusion detection in both methods increases with the increase of code length and *k*, that is, when the code length and the number of nearest neighbors returned are minimum, the classification time will be minimum. Our method costs only 0.15s to complete the classification. However, directly using *k*-NN intrusion detector to classify, the classification time will be all more than one minute, that is, it takes about 400 times the time of our method to complete all intrusion detection classification. Therefore, the proposed framework takes less time to achieve a classification effect comparable to or even better than *k*-NN intrusion detector, which can prove the high rate of detection and effectiveness of the proposed framework.

IV. Conclusion

This paper proposed a network intrusion detection based on learning to hash, . The above exploration shows that our method not only has good detection results for intrusion detection data, but also significantly improves the detection speed through the combination of learning to hash and optimized *k*-NN classifier based on data distribution ratio. In addition, it effectively avoids the problem of repeating training classifiers to detect unknown classes faced by others intrusion detection algorithms and saves lots of resources and computing power.

Acknowledgment

This work was supported in part by the NSFC under Grants 61772112, 61672379, and 61602226.

References

[1] D. Denning, “An intrusion-detection model,” IEEE Trans.Softw.Eng, 1987, 13(2):222-232.

[2] Buczak. A. L. , Member, IEEE, et al, “A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection,” IEEE Communications Surveys & Tutorials, 2017, 18(2):1153-1176.

[3] Panigrahi C. R., Tiwary M., Pati B., et al, “Big Data and Cyber Foraging: Future Scope and Challenges,” Techniques and Environments for Big Data Analysis, Springer International Publishing, 2016.

[4] Kim J., Thu H. L. T., et al, “Long short term memory recurrent neural network classifier for intrusion detection,” Platform Technology and Service (PlatCon), International Conference on. IEEE, 2016: 1-5.

[5] Staudemeyer R. C., Omlin C. W., “Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data,” Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference, ACM, 2013: 218-224.

[6] Marshall A Kuypers, Thomas Maillart, and Elisabeth Paté-Cornell, “An empirical analysis of cyber security incidents at a large organization,” Department of Management Science and Engineering, Stanford University, School of Information, UC Berkeley, 30, 2016.

[7] Li L., Yu Y., Bai S., et al, “An Effective Two-Step Intrusion Detection Approach Based on Binary Classification and k-NN,” IEEE ACCESS, 2018, 6(3):12060-12073.

[8] D. T. Larose, “k-Nearest Neighbor Algorithm,” Hoboken, NJ, USA: Wiley, 2005, pp. 90–106.

[9] Wang J., Shen H. T., Song J., et al, “Hashing for Similarity Search: A Survey,” Computer Science, 2014.

[10] Charikar M., “Similarity estimation techniques from rounding algorithms,” ACM Symposium on Theory of Computing, 2002. 380-388.

[11] Wang J., Liu W., Kumar S., et al, “Learning to Hash for Indexing Big Data - A Survey,” Proceedings of the IEEE, 2015, 104(1):34-57.

[12] Do T. T., Tan D. K. L., Pham T. T., et al, “Simultaneous Feature Aggregating and Hashing for Large-scale Image Search,” 2017.

[13] Gong Y., Lazebnik S., “Iterative quantization: A procrustean approach to learning binary codes,” Computer Vision&Pattern Recognition, 2011.

[14] Charikar M., “Similarity estimation techniques from rounding algorithms,” ACM Symposium on Theory of Computing, 2002. 380-388.

[15] Punjani A., “Fast search in Hamming space with multi-index hashing,” Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[16] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in Proc. 2nd IEEE Symp. Comput. Intell. Secur. Defence Appl., Jul. 2009, pp. 1-6.