Scalable Deep Learning Through Fuzzy-based Clustering in Autonomous Systems

Ganapathy Mani
Department of Computer Science & CERIAS
Purdue University
West Lafayette, USA
manig@purdue.edu

Bharat Bhargava
Department of Computer Science & CERIAS
Purdue University
West Lafayette, USA
bbshail@purdue.edu

Jason Kobes
Research Consortium
Northrop Grumman Corporation
McLean, USA
Jason.Kobes@ngc.com

***Abstract*—Autonomous cyber systems continuously receive large streams of diverse data from numerous entities operating and interacting in their environment. It is imperative that the learning models in autonomous systems to scale up to process the new and unknown data items. Scalable learning is nothing but a method to achieve maximum classification without rejecting any unknown data item that were not present in the training or testing datasets as anomalies. In this paper, we present Bitwise Fuzzy-based Clustering (BFC) technique through error-correcting codes to address the problem. Through BFC, we can approximate the classes of multidimensional features of data items by reversing standard forward error-correction coding. Approximating classes problems generally arise in autonomous systems that are processing fuzzily cataloged data items. These data items can be classified by applying binary vectors to their corresponding features (1: feature is present or 0: feature is absent) to obtain message words. These codewords can be used as cluster centers. In BFC technique, binary vectors of 23 bits are mapped into codewords (labels or indices) of 12 bits. Two different 23-bit binary vectors with the Hamming distance of 2 will have a few common labels. This setting enables the clustering of neighboring 23-bit binary vectors with at most 2-bit variation (mismatch) from a given input. BFC technique has 2^{23} codeword space, which makes it ideal for scalability in clustering of millions of categories and their associated features. With reasonable redundancy, the clustering can be accomplished in $\mathcal{O}(N)$ time.**

***Keywords*-error-correcting codes; clustering; scalable learning; deep learning; fuzzy logic; autonomous systems;**

I. Introduction

Intelligent Autonomous Systems (IAS) have four fundamental characteristics [1]: They are (1) cognizant of their own behavior as well as the environment in which they operate, (2) highly reflexive where they are adapt to unknown situations, (3) effective in knowledge discovery, and (4) trustworthy (i.e.) adheres to the principles of resilient security and privacy. Classification problems generally arise in dynamic environments with many classes [2], in which IAS operate. The large data streams can only be analyzed in indirect ways where they can be either sampled or classified into clusters. But sometimes, sampling may produce very skewed results [3] since it heavily depends on the size of the data and number of samples that can be derived from that data. Hence, clustering can be a right option to organize large amounts of data and learn from it. One well-known example is recognizing thousands of visual data items—one of the biggest challenges in computer vision and big data processing. The vast number of classes make the conventional one-verses-all multi-class paradigm to be very expensive in terms of time and space. The time complexity grows linearly with number of categories, which makes training and testing prohibitive for real-time practical applications such as autonomous robots. These autonomous systems require high throughput with low latency.

The proposed BFC technique's near matching procedure employs hashing method to label the data. The data items are compared and organized in clusters based on the indices obtained through hash transformation of the input data. Hence searching a particular cluster becomes a $\mathcal{O}(1)$ operation. The duplicate indices can be handled with approximate matching, which has a significantly low overhead. The approximation is a two-step process. First, the dimensionality of the data is reduced to expected mismatches. Second, this dimensionality reduced data is sent to the hashing function. A similar approximation technique is proposed in the well-known Soundex Encoding [4] where it gives the dimensionality reduced form of phonetically similar words—fuzzy matching. Soundex retains the first alphabet and drops the vowels from the remaining sequence. But Soundex technique does not have a robust encoding mechanism to encode different type of featured sequences [5]. An effective method for approximation must rely on data items that are represented through binary bit vectors of fixed length. In BFC technique, the binary vectors are clustered together based on the hamming distance from one vector to another, which is accomplished through fuzzy matching by reversing the traditional error-correcting codes.

The rest of the paper is organized as follows. Section II presents the related work, section III summarizes the BFC technique with the explanation of perfect codes and presents proof to show the effectiveness of the encoding, section IV evaluates the scheme based on recall as well as CPU performance and compares the time complexities with other clustering techniques, section V introduces the extension of BFC through CNN application on clustered data, and we finally conclude our work in section VI.

II. Related Work

Employing hamming distance for unsupervised learning is widely used in computer vision based research and applications. K-means Hashing (KMH) simultaneously computes k-means clusters at the same time learns the binary indicies of a particular block of pixels (or cell) [6]. In [7], the authors have developed an algorithm to compute hash functions base on minimizing reconstruction error between the Hamming distances and the original distances of the appropriate binary embeddings. The algorithm perfectly preserves the distances when mapping with Hamming space through hashing. Hashing technique with considerably minimal loss for binary codes has been proposed in [8]. The technique is designed based on structured prediction of latent variables and loss function with Hamming distance parameter that is similar to the hinge loss used in the SVM. Due to the hashing and Hamming space application, the algorithm becomes online, efficient, and scalable. BFC technique employees hashing technique and Hamming space to perform clustering as well.

Categorizing large number of image data items has received significant attention in computer vision after datasets of very large object classes such as ImageNet [9] became publicly available. One category of work focuses on efficient feature categorization and achieving significant performance increases [10] [11]. Another category focuses on optimizations using tree-based models [12] [13]. Recent advances in deep learning has lead to the proposal of state-of-the-art performance challenges [14]. These models assume that there is always a prior probability available for entire training data. These mechanisms work relatively well if the goal is just to store maximum number of classes. Error-correcting codes have played vital part in developing machine learning tools [15]. A fault-tolerant indexing scheme has been proposed in [16] that takes advantage of the perfect codes and our current work is inspired from their project. These codes provide a robust classification mechanisms when the data can only be categorized in a fuzzy manner.

III. Bitwise Fuzzy Clustering (BFC)

In conventional forward error correction techniques, redundant parity bits are joined with a data word to create a codeword. In case of any discrepancies during transmission of the codeword, parity bits are used to restore the initial data word. In BFC scheme, we use a perfect (23, 12, 7) error correction scheme [17]. It has the minimum hamming distance of 7. So up to 3 bits of errors can be corrected if there are discrepancies. Codewords become spheres (clusters) with unique 23-bit binary vector as hash index. The explanation of mapping data word space to codewords can be found in [18].

A. *Perfect Error-correcting Codes*

Hamming bound for error correcting codes is defined as, for any code E,

$$E = (M_n, D_k, H_d) \tag{1}$$

Here M_n is the length of codeword, D_k is the dimension (length of the data word), and H_d is the minimum Hamming distance between two codewords and $H_d \leq 2e + 1$. It satisfies,

$$|E| \sum_{i=0}^{e} \binom{M_n}{i} \leq 2^{M_n} \tag{2}$$

PEC with (23, 12, 7) code satisfies the Hamming bound equality as follows (M_n = 23, D_k = 12, and H_d = 7),

$2^{12} \cdot \sum_{i=0}^{3} \binom{23}{i} = 2^{12} \cdot 2^{11} = 2^{23}$

Here,

$2^{11} = \binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3}$

The creation of hashing can be explained with the Hamming code (7, 4, 3). The code has 16 datawords of size 4-bits. Each dataword correlates to a sphere of diameter 2 in a 7-dimensional binary cube. In order to transmit the 4-bit message, the transmitter adds extra 3 bits to transmit the message as a 7-bit message. For example, a message 1100 will be transformed into 1100101 and it will be the center of the codeword sphere. The message can be decoded from the 7-bit codeword with the tolerance of a 1-bit distortion. In our fault-tolerant hashing technique, the approach is reversed: given a 7-bit message, the hash value (say, index) will be computed by the message decoding method so that it constitutes a dataword. For example, 1100101 will have the hash value of 1100. Lets assume that there is a 1-bit distortion such as 1100100. The decoding method will still output the same dataword (i.e.) hash value: 1100. Hence, with our hashing method, any 7-bit message at the Hamming distance 1 (radius of the sphere) from the given message 1100101 can be recalled through the same hash value 1100.

But in this case, the given message must be the center of the codeword sphere. Otherwise, the Hamming distance 1 can represent any neighboring sphere. So computing a hash value need to be expanded. Usually, if there is a one-bit distortion, it can be corrected by searching all the 8 hash buckets, where first is computed by the hash value of the message and other 7 are computed by one-bit changes. But our hashing scheme, using mod 2 addition, reduces the complexity created by this 8 hash bucket brute force approach. Consider a message 1001 with 3 extra bits 100, yielding 1001100. One bit distortions such as 1001100 $\bigoplus$ 00000010 = 1001110 gives us 1000 as hash value. The combinations one-bit distortions in 7-bit messages gives us just 4 hash values: 1011, 0001, 1000, and 1101. This reduces the brute-force bit bucket size in half.

B. BFC Technique

(23, 12, 7) has one-to-one relationship between codeword and data word. There are 2^{23} codewords and 2^{12} data words. If the codeword space assumed to be a binary cube with 23 dimensions, it can be split equally into 2^{12} spheres. Thus each codeword sphere will consists of 2^{11} = 2048 binary vectors of the size of 23 bits. Since PEC can correct mistakes up to three bits, 23-bit vectors inside the spheres are within H_d = 3 Hamming distance from the centroid 23-bit vector. Since PEC focuses on H_d = 3, the clustering scheme will be interested in $\binom{23}{3}$ = 1771 23-bit binary vectors that are 3 Hamming distance away from center. Other vectors with even just 1 more bit variation will be sent to a different sphere (cluster). $\frac{1771}{2048} \approx 86\%$ of the vectors will stay close to the center where as $\frac{277}{2058} \approx 14$ will be assigned to new cluster's hash index.

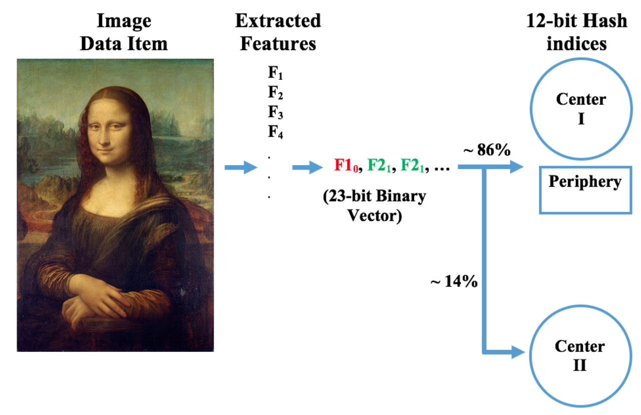

Figure 1. BFC Technique—Fuzzy criteria for clusters

BFC technique (Figure 2) applies 23-bit binary vector to extracted features (1 for presence of feature and 0 for the absence) or predefined features and clusters them in appropriate hash indexes (cluster labels). Predefined feature extraction works as follows: assume a system that is classifying images of the city San Francisco. The user can set the features and program the software to look for those features and apply the 23-bit vector. For example, F_1: Does the image has sharp triangle shaped white pillar?, $F2$: Is there a street sign(s) with an alphabet and number? etc. Based on the detected features, the 23-bit binary vector (e.g. 0, 1, 0, 0, 1, ...) will be created and will be assigned to a specific cluster in the hash indices. The vector does not need to be 23 bits long since the scheme supports even fewer parameters. Another advantage of BFC is that the 23-bit binary vectors can be used for autonomously generating 23-bit labels to be applied for a specific type of data item. If unknown data (that was not present in either training or testing dataset) appears then a new template (label) can be generated autonomously, creating a new cluster.

Since the suggested clustering technique uses (23, 12, 7) cyclic code (a subclass of linear codes), the implementation can be simplified through algebraic computations by generating polynomials instead of parity checks. Thus the code can be computed by the polynomials (this derivation is given in [18] but for the sake of clarity, we recite it here.),

$$H_1(y) = 1 + y^2 + y^4 + y^5 + y^6 + y^{10} + y^{11} \tag{3}$$

$$H_2(y) = 1 + y + y^5 + y^6 + y^7 + y^9 + y^{11} \tag{4}$$

The features labeled from 0 to 22 can be represented by polynomials $\mod y^{23} - 1$ with correlated coefficient in the binary space $H(y)$. The data item can be represented by a 12-bit binary vector and adding 11 parity check bits gives us 23-bit label $< l_0, l_1, ..., l_{22} >$. Let the label be represented by $T(y)$:

$$T(y) = l_{11}y^{11} + ... + l_{22}y^{22} \tag{5}$$

And the parity check polynomial $C(y)$,

$$C(y) = l_0 + l_1 y + l_2 y^2 + ... + l_{10}y^{10} \tag{6}$$

Then the entire label of a data item is represented by,

$$L(y) = T(y) + C(y) \tag{7}$$

A label vector of a data item must be a multiple of generator polynomial such as $H_1(y)$ or $H_2(y)$,

$$L(y) = X(y)H(y) \tag{8}$$

Here $X(y)$ is a polynomial with degree = 11. From this, we can define the encoding procedure with the following steps:

1) Compute the product of the data label polynomial with y^{11} to compute $C(y)$
2) Compute $\frac{T(y)}{H(y)}$ to get the remainder $C(y)$
3) Compute the label with $C(y)$ + $T(y)$

Using (23, 12, 7) code for clustering raw data is based on the following lemma. A direct proof of this lemma will give the elaborate geometrical details in the suggested clustering technique.

Lemma: Given a 23 dimensional binary vector that is partitioned into 2^{12} spheres, then searching a sphere with the hamming diameter of 2 placed in the cube, it either lies (a) totally within the sphere or (b) distributed uniformly across 6 equally partitioned spheres.

Proof: Consider a sphere that is partitioned with center P_c. Based on equation (2), it will have 2048 geometrical points that are divided into 4 categories based on their distance from P_c. Lets assume two cases of $Distance(P_c, S_c)$ where S_c is another sphere to check whether it resides inside the partitioned sphere (cluster) or not.

Case 1: $0 \leq Distance(P_c, S_c) \leq 2$
In this case, the S_c with radius of 1 resides in the partitioned sphere that has the radius of 3.

Case 2: $Distance(P_c, S_c) = 3$
Lets assume that there exists a set of 23-bit binary vectors with the spheres of radius 1: $V = v_0, v_1, ..., v_{22}$ where v_0 = <0,0,0 ... , 0>, v_1 = <0, 0, 0, ... , 1>, ... , v_{23} = <1, 0, 0, ... , 0 >. Then the center of sphere S_c can be computed by 3 fixed unit vectors like v_a, v_b, and v_c.

$$S_c = P_c + v_a + v_b + v_c \qquad (9)$$

Consider a modification for S_c where by adding another vector v_k where $k \neq a, b, c$, or 0 and k = 1, 2, 3, ..., 23. The vector v_k falls into the sphere that is adjacent to P_c at the distance of 3 from the center. Say, S_k constitutes the points of the sphere that is in question,

$$S_k = P_c + v_a + v_b + v_c + v_k \qquad (10)$$

Equation (10) can be rewritten as,

$$P_c + v_a + v_b + v_c + v_k = P_s + v_x + v_y + v_z \qquad (11)$$

where x, y, or $z \neq a, b, c$, or k. This represents the fact that each of the sphere with 4 points with the center in S_c lies adjacent to the partitioned sphere with center P_s, where P_s is another adjacent sphere to P_c. Given V and k, we have 24 points. And in those points, we have identified the location of quadruple of points in P_c—the first partitioned sphere. This procedure can be continuously reused until every point in V is searched. Thus 24 points in v constitute different quadruples and lie in adjacent spheres. Hence, when $Distance(P_c, S_c)$ = 3, we get six spheres of different partitions.

IV. Evaluation

The fuzzy matching depends on comparison operations. It requires time and space. BFC scheme offers various possibilities such as variations in sizes of matching adjacent spheres correlating to different generating polynomials. Precision and recall should be considered in order to measure the effectiveness of the fuzzy classification techniques. We consider recall as one of the metric to measure the performance of the clustering technique. We built BFC with 2^{22} = 4,194,304 for the vector representation.

Figure 3 represents the recall probabilities (recited from [18]) of the suggested BFC technique. The case 1–1 means matching two spheres of size 1 (i.e.) a direct matching of 12-bit hash values. For the higher hamming distances, the matches that occur are relatively small. With 1–6, we get 100% recall for 1 Hamming distance. In the cases of 2–12, 2–2, 12–12, we performed two kinds of (23, 12, 7) partitioning. For example, 2-12 means that 2 adjacent spheres of size 1 are compared with 2 adjacent spheres of size 6.

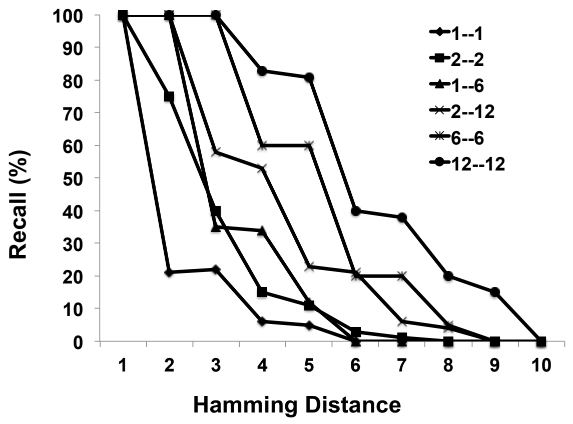

Figure 2. Recall probabilities with (23, 12, 7) partitioning

We also use another metric of measuring CPU performance in terms of number of reference clock cycles used by the BFC technique. Clustering large data requires intensive computing capabilities with systems that can handle the clustering algorithm at the same time processes the streaming data on-the-fly. Figure 4 shows the reference clock cycles required for BFC technique with another encoding software, 7zip. Clock cycles were recorded with a periodic fixed sampling of every 1,000,000 instructions. We used process thread API [19] to obtain clock cycle samples. The header is used to set the sampling period and reference clock cycles were recorded for all the processes that are active and running. During run time, the process id (PID) of the clustering algorithm and 7zip were manually noted. Based on the process id, sampled reference clock cycles were filtered and obtained. The clock cycles show that the

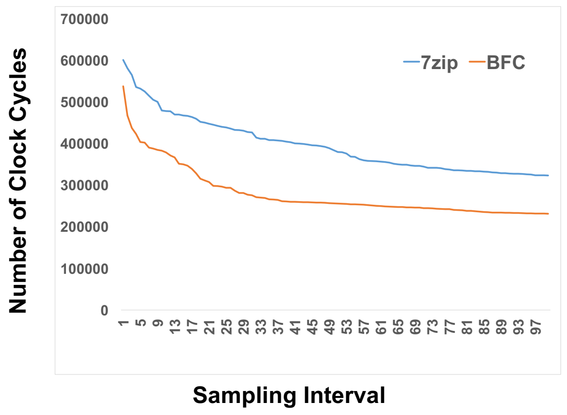

Figure 3. Fixed samples of number of clock cycles required for the execution of BFC

clustering algorithm takes considerably less number of clock cycles to compute the clusters compared to other encoding software 7zip.

Another the main disadvantage of popular clustering algorithms is their time complexity. Conventional clustering algorithms with rich functionalities operate in exponential time. Hence they are prohibitive in preprocessing intense real-time data streams. BFC clustering operates on hash

Table I
COMPARISON OF COMPUTATIONAL COMPLEXITIES OF CLUSTERING ALGORITHMS

Clustering Algorithms	Time Complexities
k-means	$\mathcal{O}(nkd)$
Hierarchical Clustering	$\mathcal{O}(n^2)$
Clustering using REpresentatives (CURE) [20]	$\mathcal{O}(n^2\ log\ n)$
ROCK [21]	$\mathcal{O}(min(n^2, nm_m m_a))$
CLICK [22]	$\mathcal{O}(n\ log\ n)$
BFC	$\mathcal{O}(n)$

indices for cluster assignment. Once the 23-bit binary vector is applied to a data item, the algorithm needs to search the related cluster in the hash and store it. In the best case scenario, hash search can be completed in constant time ($\mathcal{O}(1)$) and in the worst case scenario the hash search can take $\mathcal{O}$(n) time. Thus the PEC clustering takes $\mathcal{O}$(n) time. There are other clustering mechanisms such as Fuzzy c-means clustering [23] and BIRCH clustering [24] that can complete the task in $\mathcal{O}$(n) time but they are not equipped to deal with large continuous stream of data. Comparison with existing clustering methods is given in Table I.

V. EXTENSIONS OF BFC TECHNIQUE

As our future work, we introduce an extension of the BFC technique to increase the learning capability of autonomous systems. Convolutional Neural Networks (CovNets or CNN) are used for image recognition and classification. Recent developments in Natural Language Processing (NLP) are based on CNNs that are used for sentence classification [25].

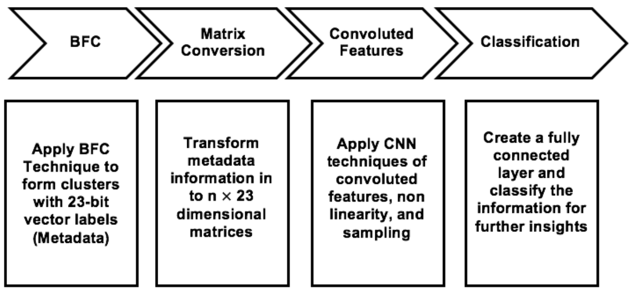
BFC
Matrix Conversion
Convoluted Features
Classification
Apply BFC Technique to form clusters with 23-bit vector labels (Metadata)
Transform metadata information in to n × 23 dimensional matrices
Apply CNN techniques of convoluted features, non linearity, and sampling
Create a fully connected layer and classify the information for further insights

Figure 4. Workflow of applying CNN to BFC classified data

Figure 4 shows the application of CNN on the clustered data. Since these metadata items have common features, they can be transformed into a matrix and we can conduct a sliding window operation [26]. Even through it is a brute force approach, the feature selection is not necessary thus no new training datasets are required, both of which significantly reduce the time and space complexity of the learning process. This is particularly helpful in autonomous systems since they come across unknown data from their dynamic environments and they need to process the data faster, and make decisions based on the results.

Once the small blocks are processed to generate convoluted features, they go through down sampling or pooling to find most interesting features by finding the maximum value in each block. The final values are processed as a new vector to compare and classify. The approach of CNN on clustered data has advantages: (1) the clusters themselves provide valuable information with organized data but applying CNN on top of it can reveal interesting relations between each clusters. Sub clusters can be classified that can provide greater insights into unknown data items that may be wrongly clustered and (2) the errors or mismatches caused by the fuzziness of BFC can be cleared through CNN sub clusters. CNN is costly in very larger datasets [27]. Since the CNN is applied only at the end to required clusters, the time and space complexity of CNN on all data items can be reduced considerably.

VI. CONCLUSION

In this paper, we proposed BFC technique based on (23, 12, 7) perfect error-correcting codes. Reversal of this error-correction scheme results in a robust clustering technique for stream data processing. The scheme offers 23-bit binary vector label for each data item and producing up to 2^{23} combinations of clusters. Hence the scheme can be used for classifying data with thousands of categories. It can correct up to 3 bits of errors in that 23-bit binary vector thus the scheme offers some fault tolerance. One of the most important qualities of BFC clustering is that it operates in $\mathcal{O}(n)$ time complexity. Compared to traditional and rich functionality clustering algorithms, BFC is fast and fault-tolerant. We proposed a framework to enable further insights from clustering through deep learning: applying CNN to clustered data. Since it is applied to only sub clusters, the time and space complexity of learning will be limited. We intend to apply this technique to computer vision problems such as large-scale image classifications in dynamic environments.

ACKNOWLEDGMENT

This research work is supported by Northrop Grumman Corporation Research Consortium. Approved For Public Release #18-1640; Unlimited Distribution.

References

[1] G. Mani, B. Bhargava, B. Shivakumar, and J. Kobes, "Incremental learning through graceful degradations in autonomous systems," in *2018 International Conference on Cognitive Computing (ICCC)*. IEEE, 2018.

[2] J. Xiao, K. A. Ehinger, J. Hays, A. Torralba, and A. Oliva, "Sun database: Exploring a large collection of scene categories," *International Journal of Computer Vision*, vol. 119, no. 1, pp. 3–22, 2016.

[3] G. W. Imbens and M. Kolesar, "Robust standard errors in small samples: Some practical advice," *Review of Economics and Statistics*, vol. 98, no. 4, pp. 701–712, 2016.

[4] J. Fisher, P. Christen, Q. Wang, and E. Rahm, "A clustering-based framework to control block sizes for entity resolution," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 279–288.

[5] A. Karakasidis, V. S. Verykios, and P. Christen, "Fake injection strategies for private phonetic matching," in *Data Privacy Management and Autonomous Spontaneus Security*. Springer, 2012, pp. 9–24.

[6] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 2938–2945.

[7] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Advances in neural information processing systems*, 2009, pp. 1042–1050.

[8] M. Norouzi and D. M. Blei, "Minimal loss hashing for compact binary codes," in *Proceedings of the 28th international conference on machine learning (ICML-11)*. Citeseer, 2011, pp. 353–360.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[10] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang, "Large-scale image classification: fast feature extraction and svm training," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 1689–1696.

[11] K. Yu and T. Zhang, "Improved local coordinate coding using local tangents," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 1215–1222.

[12] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[13] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei, "What does classifying more than 10,000 image categories tell us?" in *European conference on computer vision*. Springer, 2010, pp. 71–84.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[15] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[16] G. Mani, N. Bari, D. Liao, and S. Berkovich, "Organization of knowledge extraction from big data systems," in *Computing for Geospatial Research and Application (COM. Geo), 2014 Fifth International Conference on*. IEEE, 2014, pp. 63–69.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[18] S. Y. Berkovich and E. El-Qawasmeh, "Reversing the error-correction scheme for a fault-tolerant indexing," *The Computer Journal*, vol. 43, no. 1, pp. 54–64, 2000.

[19] "processthreadsapi.h header," https://docs.microsoft.com/en-us/windows/desktop/api/processthreadsapi/, 2018, [Online; accessed 22-July-2018].

[20] S. Guha, R. Rastogi, and K. Shim, "Cure: an efficient clustering algorithm for large databases," in *ACM Sigmod Record*, vol. 27, no. 2. ACM, 1998, pp. 73–84.

[21] ——, "Rock: A robust clustering algorithm for categorical attributes," *Information systems*, vol. 25, no. 5, pp. 345–366, 2000.

[22] M. Peters and M. J. Zaki, "Click: Clustering categorical data using k-partite maximal cliques," *Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY*, vol. 12180, 2004.

[23] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers & Geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984.

[24] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *ACM Sigmod Record*, vol. 25, no. 2. ACM, 1996, pp. 103–114.

[25] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *arXiv preprint arXiv:1702.01923*, 2017.

[26] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[27] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.