Large-Scale Image Search

Peiyi Zheng
Department of Computer Science
Johns Hopkins University
pzheng4@jhu.edu

***Abstract*—In this paper, we illustrate the design of a large-scale image search system. Our system uses ResNet, which is proved to perform very well on many computer vision tasks, to extract features from images which ensures good accuracy. What's more, we introduce strategies like iterative quantization and multi-index hashing to provide excellent space and time complexity. Our experiments show that the system can return acceptable result in a reasonable time.**

I. Introduction

In present-day Internet, image recognition and search technology are widely used in applications, and are expanding theirs influence. The key of improving user experience of such search application is to find a high-efficient and outstanding search technique. Most of the current mainstream search engines have the functionality to search images by a given image, which greatly meets the demand of many users. By processing the pictures in the knowledge base, the performance of searching similar images has improved a lot compared with simply using the raw image.

One of the popular approaches in previous years is using normalized color histogram as feature of images and the earth mover's distance as comparison metric for image retrieval [1]. This approach only performs well when the given image has similar color distribution with some images in the database. Hence it can not provide accuracy good enough for reliable image search.

Another common approach in recent years is using Bag-of-Visual-Words as feature representations [2]. The basic idea is to detect key points in the image and cluster key points to generate a visual-word dictionary. Then we can consider each image in the database as a bag of visual words and transform them into some visual-word vectors. By comparing the distance between visual-word vectors, we are able to determine the similarity between original images. [3] evaluates the performance of this model which shows that it is better than color histogram but still can be improved.

In order to achieve better accuracy, we decide to introduce convolutional neural network for image features extraction.

II. Feature Extraction

Deep convolutional neural network has been proved to have great performance on many computer vision related tasks. AlexNet shows its power in ImageNet classification task in ILSVRC 2012 [4]. After that, with the growth of network depth, the top-5 error rate on ImageNet classification task keeps decreasing. VGG with 19 layers is the winner of ILSVRC 2014 [5] and GoogLeNet with 22 layers wins 2015's ILSVRC [6].

In our system, we use deep residual networks(ResNet) [7] to extract features from images for image search. ResNets is a very deep network which has state-of-art performance for image classification and object detection. According to the experiment in [7], ResNet-152 has a top-5 error rate 4.49% on the ImageNet validation set [8] which is better than VGG(8.43%) and GoogLeNet(7.89%). This excellent result shows that ResNet is very suitable for our system.

III. Iterative Quantization

Our experiment uses 150000 images from test set and validation set of ImageNet [8]. The total size of all feature vectors extracted by ResNet is 3.6 GB, which doest not satisfy the requirement of space complexity for image search. Therefore we introduce a new strategy, iterative quantization, to transform the long vector of float numbers into a short binary code while preserving the similarity between the corresponding images [9].

The basic idea of iterative quantization is to reduce the dimensions of features and iteratively find a good binary representation for original features. The first thing we need to do is to reduce the dimensions of the original feature matrix. The most common tool for dimensionality reduction is principal components analysis (PCA).

PCA was invented by Karl Pearson long time ago but it is still very popular now [10]. PCA maps data in n dimensions space to k dimensions space ($k < n$) while maximizing the variances for each principle components. The reason of maximizing variance is that in signal processing we usually consider signal to have higher variance than noise. Therefore PCA is keeping the important information in signal while removing the noises. This property helps us get away from the irrelevant noises in image features.

Therefore firstly we apply dimensionality reduction on the $n \times k$ features matrix via principal components analysis to get a new matrix V with dimension c. Then we define $\widetilde{V}$ as VR where R is an orthogonal $c \times c$ matrix. The binary quantization matrix B is determined by $\text{sgn}(\widetilde{V})$. We introduce rotation matrix R because V is fixed. If we do not modify V, then B is also fixed. So we are unable to minimize the quantization error. But if we iteratively optimize R to help us transform V into a better representation $\widetilde{V}$, we can reduce the quantization error gradually.

Our goal is to minimize the quantization error because small quantization error means we preserve more information from original features:

$$Q(B,R) = \|B - VR\|_F^2 = \|B - \widetilde{V}\|_F^2$$

Where $\|\|_F$ denotes the Frobenius norm. Expanding $Q(B,R)$, we have:

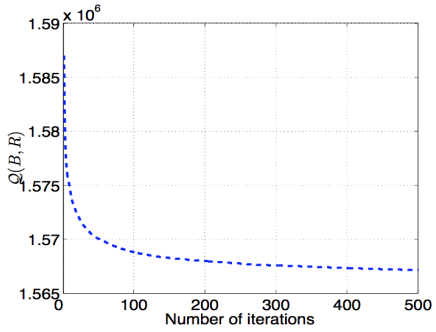
$$Q(B,R) = nc + \|V\|_F^2 - 2tr(BR^TV^T)$$

Hence minimizing $Q(B,R)$ is equivalent to maximize $tr(BR^TV^T) = \sum_{i=1}^{n}\sum_{j=1}^{c} B_{ij}\widetilde{V}_{ij}$. According to the algorithm in [9], we begin with initializing *R* to a random orthogonal matrix. Then in each iteration, we do the following two steps:

Fix *R* and optimize *B*. In order to maximize $\sum_{i=1}^{n}\sum_{j=1}^{c} B_{ij}\widetilde{V}_{ij}$, obviously we should let $B_{ij} = -1$ when $\widetilde{V}_{ij} < 0$ and $B_{ij} = 1$ otherwise. It's clear if we set B_{ij} in this way, every item $B_{ij}\widetilde{V}_{ij}$ will be nonnegative. Otherwise some $B_{ij}\widetilde{V}_{ij} < 0$ exist which can only increase the quantization error.

Fix *B* and optimize *R*. After we update *B*, we should fix it and continue to update *R*. When *B* is fixed, minimizing $Q(B,R)$ is equal to find a rotation matrix that minimizes the distance between points set *B* and points set *V*. This is the orthogonal Procrustes problem [11] which can be solved using SVD. Schonemann gave the solution of this problem based on SVD [11]. So according to Schonemann's strategy, we can find the SVD $X\Omega Y$ of BV^T and optimize *R* to *XY*.

We can iterate the two steps above until the quantization error is small enough. The following data comes from the experiment in [9] which shows that 100 iterations are enough for minimizing quantization error.



If we let all $B_{ij} = -1$ to be 0, each row in B is a binary code. After we transform the whole features database into 256 bits binary codes, the space it needs decreases to 36MB which is absolutely better than 3.6GB. For a given query image, we can represent it as a feature vector using ResNet. Then we should reduce its dimensions and multiply the rotation matrix *R*. At the end we set each negative element to zero so the query image is transformed into a binary code.

IV. Muliti-Index Hashing

For a given binary code, if we want to find out the top-k similar images in the database, we should firstly calculate the distance between it and each codes in the database. After that we do a K nearest neighbors search to get the top-k nearest result. When we try this approach on our dataset, it takes about 10 seconds to finish the query. Usually we don't want to wait for such a long time. Hence we need to do better than linear search.

A simple idea of accelerating the search is to transform the binary codes into integers so that we can use bit operations to quickly compute the hamming distance between binary codes. Unfortunately this method will not work in our case since the length of binary codes is larger than 64. Therefore we should try some data structures like hash table in order to beat linear search.

[12] presents an useful method to do fast search in hamming space with multi-index hashing. According to their algorithm, we do not use KNN search directly. Instead, now we try to find all images within hamming distance r from the query and increase r in each iteration until we find out more than K candidates. Then we do KNN search in this small dataset. The problem is for a given binary code with length b and a hamming distance r, the number of possible candidates is:

$$N(b,r) = \sum_{k=0}^{r} \begin{pmatrix} b \\ k \end{pmatrix}$$

For example, when $b = 64$ and $r = 7$, $N(b,r)$ is about one billion. Therefore if we want to precompute and save all possible candidates in the hash table for any key (b,r), the hash table will become too large.

To overcome this issue, [12] reminds us to notice the fact that we can divide the query binary string into several disjoint substrings, for instance, m substrings. If two binary strings x and y differ by less than r bits, then at least one of their substrings must differ by less than $\lfloor \frac{r}{m} \rfloor$, otherwise according to the Pigeonhole Principle, the sum of the distances of substrings must be larger than r.

Hence, for each binary codes in the dataset, we divide them into m disjoint parts and construct hash tables for each part. Now we only need to construct m hash tables with size $N(\frac{b}{m}, \frac{r}{m})$ for radius r.

And for a query q, we also divide it into m substrings and search them in corresponding hash tables. For a substring q_i, the key $(q_i, \frac{r}{m})$ for i-th hash table gives the possible candidates whose full string might has a hamming distance less than r between q and itself.

So we can add those candidates into a candidate set and compute the full hamming distance for each candidate. We remove those candidates which really have hamming distances less than r and add them into the neighbor set. When the size of neighbor set is more than K, we do a KNN search in this small data set. Otherwise we increase r by m and start another iteration.

[12] also analyses time complexity for this algorithm. When we set $m = \frac{b}{log_2 n}$, the time complexity is $O(\frac{b\sqrt{n}}{log_2 n})$ which is better than $O(n)$. Also, when $m = \frac{b}{log_2 n}$, the extra space complexity for hash table is $O(nlog_2 n)$. Comparing to the original space complexity $O(nb)$, the hash table doesn't cost us too much storage but greatly improve the search speed.

Since $\frac{256}{log_2 150000} \approx 15$, we can just set $m = 16$ so that the lengths of substrings are the same. After our image search

system includes this algorithm, now we only need 1.5 to 2 seconds to do a KNN search. What's more, we can use m threads to parallel search in each hash table to accelerate the system.

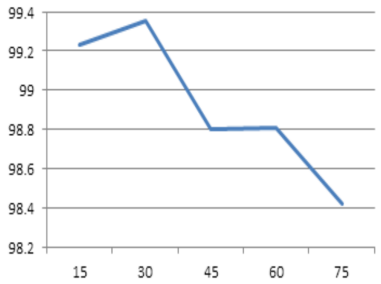
V. Experiment

To implement this system, we use Caffe [13] with a pre-trained model of ResNet-152. We run the iterative quantization algorithm for 100 times to ensure the quantization error small enough. We set m to 16 so that each substring also has same length 16.

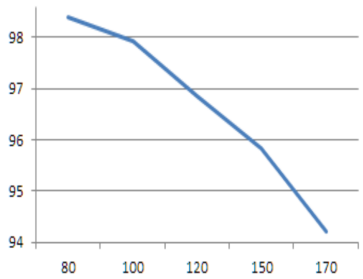
Previously the size of compressed database and query runtime has showed that our system can guarantee reasonable space complexity and time complexity. But we haven't test the accuracy. The difficulty we face is that our image database contains images from both test set and validation set of ImageNet. However we don't have the labels for test set. In order to solve this issue, we decide only to test the accuracy on validation set.

Our test procedure includes the following 3 steps. The first step is to randomly pick a image from validation set and remember its label. Then we search in the validation set for the top-K result. Since we know the labels of every image in validation set, in the last step we let the top-K result to vote the label of the query image according to the count of their own labels.

The label appears most in top-K result is the label return from our system. Hence we can check if this label matches the label of query image to calculate the accuracy. For different values of K, we run the test procedure for 1000 times and get the following result:



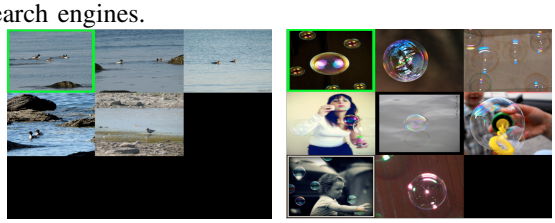
With the growth of candidates, more and more irrelevant images will influence the voting result so the system begins to pick wrong label and the accuracy begins to decrease. We also test it in another way, fix K to 30 and evaluate the accuracy based on different maximum radius:



Sadly this test is not perfect because in fact we are evaluating the classification result of objects in the images, not the image similarity. But we haven't find out any other ways to test the system.

The following results show how our system performs. The top left image with green border is the query image.

Sometimes we do not get enough reasonable result because the validation set of ImageNet contains only 50000 images which is very small comparing to the real dataset of some search engines.



VI. Conclusion

The most important thing we learn from this project is that most of the times we are unable to solve a problem just with a single algorithm. Usually we need to combine many strategies to make things really work.

In this project, in order to extract better image features, we include deep neural network in the system. To provide good space complexity and time complexity, we use iterative quantization and multi-index hashing. With the ideas from some many researchers we can finally build an useful image search system.

Computer vision is really interesting.

References

[1] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.

[2] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2007, pp. 1–8.

[3] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo, "Evaluating bag-of-visual-words representations in scene classification," in *Proceedings of the international workshop on Workshop on multimedia information retrieval*. ACM, 2007, pp. 197–206.

[4] 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.

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

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[8] 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.

[9] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 817–824.

[10] K. Peason, "On lines and planes of closest fit to systems of point in space," *Philosophical Magazine*, vol. 2, pp. 559–572, 1901.

[11] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[12] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast search in hamming space with multi-index hashing," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3108–3115.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.