Large-scale Face Retrieval System based on Locality-Sensitive Hashing

Ai yongbiao
Student ID :1155114481
1155114481@link.cuhk.edu.hk

Song wenli
Student ID :1155114524
1155114524@link.cuhk.edu.hk

Wang jiaze
Student ID :1155114536
1155114536@link.cuhk.edu.hk

Zhao Yijun
Student ID :1155115559
1155115559@link.cuhk.edu.hk

ABSTRACT

Face retrieval is one of the prime issues in big data analysis and machine learning. With the availability of powerful parallel machines and massive increase in training data, deep convolutional neural networks (CNNs) achieve great progress on face recognition. However, the progress of for face retrieval for large-scale datasets is still a challenging problem due to the huge number of face images. In this project, we proposed a novel system based on deep learning and LSH to achieve large-scale face retrieval.After preprocessing image to extract feature, we took a in-depth research into the state-of-arts LSH frameworks for different similarity metrics and chose the best method to tackle our practical problem.Finally, we implemented a website to give a intuitive interpretation of our results.

1 INTRODUCTION

Face recognition has many important applications in big data analysis and machine learning [14]. In this project, as shown in Fig.1, we proposed a large-scale face retrieval system based on locality-sensitive hashing (LSH), which allow user retrieve the most similar celebrity after upload user-defined face image [20, 21].

Earlier works address the face retrieval [6, 10] by extracting local hand-crafted features and aggregating them into global representations. The convolutional networks first shed lights on deep CNNs based face retrieval which train a spatial network based on only RGB inputs. Recent research mainly focuses on feature representation based on CNNs.

Based on the previous research, we are able to extract features from faces by CNNs easily. As for the practical similarity search problem, it is a heavy cost to find the exact nearest neighbor of certain queries [15], therefore we implement an algorithm based on LSH to do the approximate nearest neighbor search with much smaller space complexity and query time.

In this project, we evaluate our proposed system on the IMDB-WIKI [19]. The IMDB-WIKI is a face database with 523,051 face images which contains 460,723 face images from IMDB and 62,328 face images form Wikipedia. In our system, given a query face image, the system retrieves most similar faces from the IMDB-WIKI dataset.

As what has been mentioned before, our proposed face retrieval system mainly focuses on the retrieval of face features, there are basically two steps:

First step: Feature extraction: We use CNNs to detect faces and extract features in images so that we can get feature vectors to represent faces in the datasets and use the vector to calculate the similarity between query and face images in datasets [9].

Second step: Nearest neighbor search: we propose a novel framework based on locality-sensitive hashing to do the Nearest neighbor search, and output the face results which has the largest similarity with the query image vector.

In this project, our contribution is as follows: **(a)** We propose a novel Large-scale face retrieval system based on locality-sensitive hashing. **(b)** To overcome the difficulties of large-scale data analysis, we use CNNs as well as the state-of-art LSH [8, 17] algorithms. **(c)** A website is deployed to give a more intuitive interpretation of our result project presentation.

The rest of this paper is structured as follows. In Section2, we talk about the image preprocessing and feature extraction. In Section3, the main framework for LSH are discussed. The website technology is presented in Section 4. Section 5 shows the experimental results which leads to obtain the optimal LSH parameters for this project. Finally, we show our results and conclude this paper in Section .

2 DATA PREPROCESSING AND FEATURE EXTRACTION

In this section, we mainly discuss data preprocessing and feature extraction.

First, the image is resized to 256*256. And the we extract face features by Multi-task Cascaded Convolutional Networks. Then each image is compressed to a vector of 256 dimensions.

MTCNN (Multi-task Cascaded Convolutional Networks) [25] implement face detection and alignment at the same time. One the one hand, the author propose a novel cascaded structure,on the other hand, a new way for online hard sample mining is proposed. In conclusion, the main idea is can be summarized the framework adopts a cascaded structure with three stages of carefully designed deep convolutional networks that predict face and landmark location in a coarse-to-fine manner. So cascaded structure is mainly consist of three parts: Proposal Network(P-Net), Refine Network(R-Net), Output Network(O-Net), using three stages to deal with face images is a coarse-to-fine manner. Moreover, multi scale resize method is used in the network. An image pyramid is constructed, and then these images of different scales are trained as input of three stages in order to detect faces of different scales.

P-Net is mainly used to generate some bounding boxes. There are three branches at the top of the network to perform face classification, face frame regression and face key point positioning respectively during training. At the time of testing, the output of

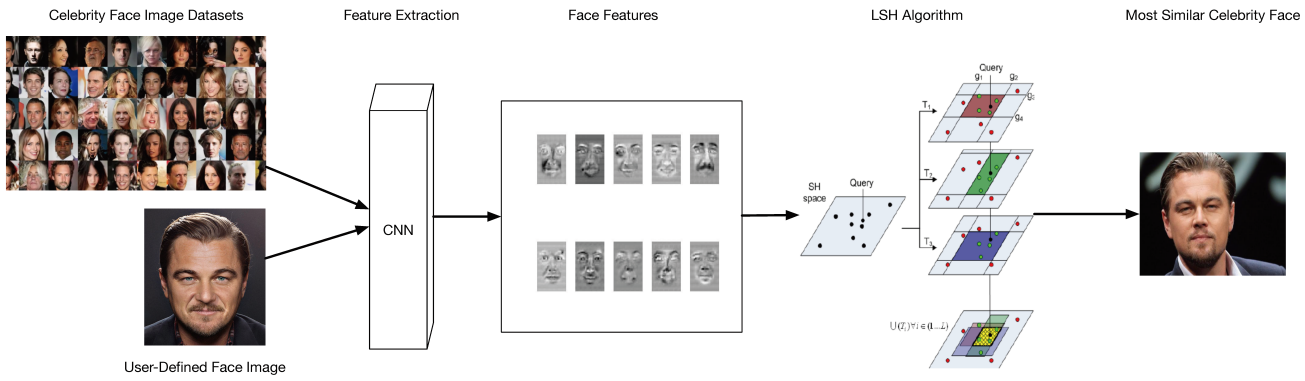

Figure 1: Face Retrieval Model. Extract features from celebrity face images to train the LSH model, then input a face image to the trained model and find the most similar face image from our datasets.

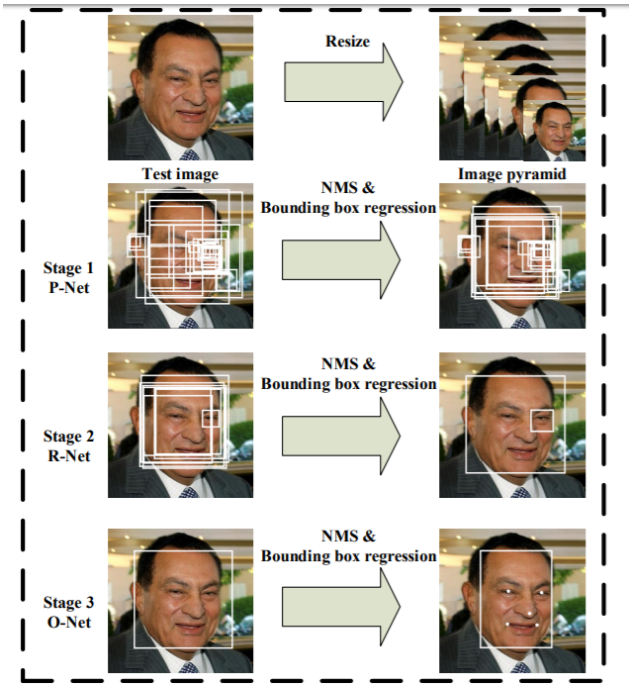

Figure 2: Pipeline of cascaded framework that includes three-stage multi-task deep convolutional networks. Firstly, candidate windows are produced through a fast Proposal Network (P-Net). After that, we refine these candidates in the next stage through a Refinement Network (R-Net). In the third stage, The Output Network (O-Net) produces final bounding box and facial landmarks position.

this step is only four coordinate information and the score of N bounding boxes. Of course, the four coordinates information has been corrected by the output of the regression branch. The four coordinates information has been corrected by the output of the regression branch.

R-Net is mainly used to remove a large number of non-face frames. The input for this step is the bounding box generated by the previous P-Net. Each bounding box is 24*24 in size and can be obtained by resize operation. Similarly, the output of this step is only four coordinate information and score of M bounding boxes, and the four coordinates information is corrected by the output of the regression branch.

O-Net is a bit like R-Net, but this step also adds a regression of landmark location. The input size is adjusted to 48*48, and the output contains four coordinate information, score and key point information of P bounding boxes.

3 LSH FRAMEWORK

3.1 Nearest neighbor search problem

After data preprocessing, we transform the images in dataset into vectors in high dimension space.Given a query image, we are going to find the top-k nearest neighbor vectors of the query vector, and output the corresponding images of these top-k vectors. In this section we explore the best methods to search the nearest neighbor of the query vector.What we mainly discussed is methods based on LSH framework and decision tree methods [11].

3.2 Background and basic idea of LSH

Locality sensitive hash(LSH) is introduced to solve the nearest neighbor search problem (NNS). In order to overcome "the curse of dimension", which means the time/space complexity to solve the NNS problem will have exponential growth based on dimension of the data. people make a trade off between precision and efficiency: instead of finding the exactly nearest neighbor p^*, we reports the C-approximate nearest neighbors p if:

$$Sim(p, q) \leq c * Sim(p^*, q)$$

As for the approximate nearest neighbors(ANN) problems [1, 2, 4],the two popular approaches are dimension reduction and space

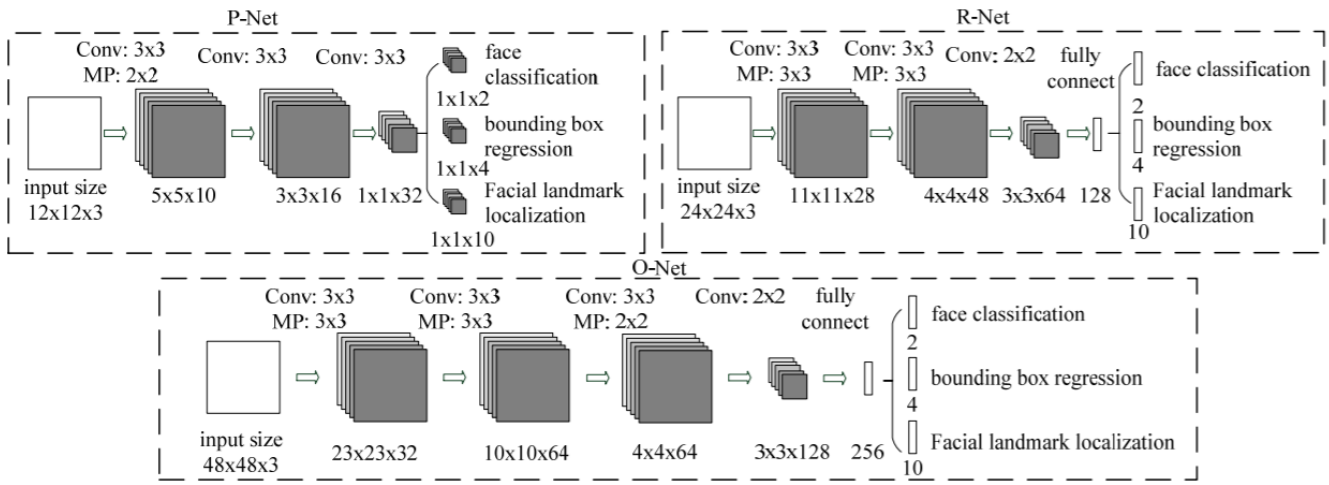

Figure 3: The architectures of P-Net, R-Net, and O-Net, where "MP" means max pooling and "Conv" means convolution. The step size in convolution and pooling is 1 and 2, respectively

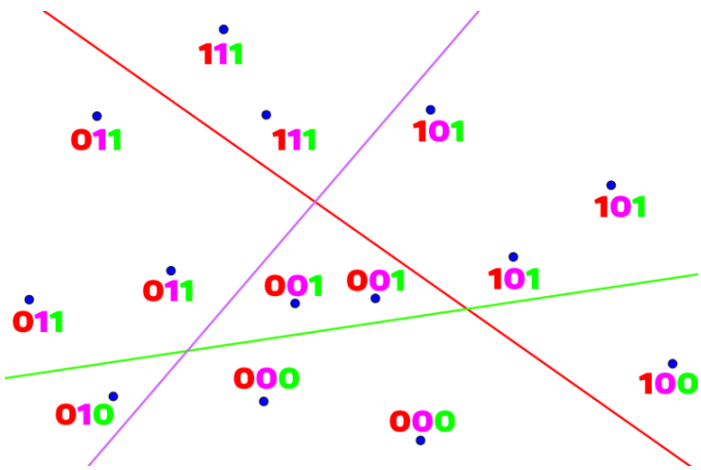

Figure 4: Illustration of space partition using LSH

partition. The first method reduce the space complexity to polynomial space, which is still very expensive in practice when the datasets is very large.[2]

LSH take the idea of space partition to achieve a space complexity linear to n with query time increased to $n^p, p \in (0,1)$. the key idea of LSH is using some random hyperplanes to partition the data space, the smaller the distance between the data point and the query point, the higher probability they been in the same sub-space after the partition progress.

Definition 1 (Locality-Sensitive Hashing (LSH)): a function family H is considered $(S_0, cS_0, p1, p2)$ sensitive if given a randomly select h from H, any points $p, q \in R^d$ satisfy:

$$if\ Sim(p,q) < S_0,\ Pr(h(p) = h(q)) \leq p1 \quad (1)$$

$$if\ Sim(p,q) > cS_0,\ Pr(h(p) = h(q)) \geq p2 \quad (2)$$

one can build data structure based on LSH for c-approximate nearest neighbor with query time $O(n^p logn)$ and space $O(n^{1+p})$. $p = \log p1/\log p2$[13]

Algorithm 1 Locality Sensitive Hashing

Input: number of hash tables L, number of hash function each table M, datasets S, qurey vector q

Output: nearest neighbor of q ,denoted as p^*

1: Given similarity metric, choose the hash family H
2: Generate a group of hash functions G by randomly pick M hash functions h from H
3: **for all** $p \in S$ **do**
4: Calculate $G(p)$
5: $G(p)$ is the concatenation of $(h_1(p), h_2(p), \ldots, h_m(p))$
6: Use hash function to hash $G(q)$ and generate a bucket based on the hash value
7: Ideally we will have buckets number approach n(which means we will put every vector in different buckets)
8: **end for**
9: Calculate $G(q)$ and hash $G(q)$ to a bucket
10: Take out all the vector in the same bucket of q as the candidate p^*
11: Repeat L times,each time using a unique G_k generated randomly
12: Compute the similarity between all the candidate vector and the query
13: **return** p^* which have the top-k similarity with q

3.3 Efficient index of LSH

The accuracy of LSH algorithms depend on the number of hash tables L and the number of hash functions each table holds M [22]. As Figure5 shows, given a vector p and query q such that $Sim(p,q) = c$, then we can compute the probability that p is considered as a

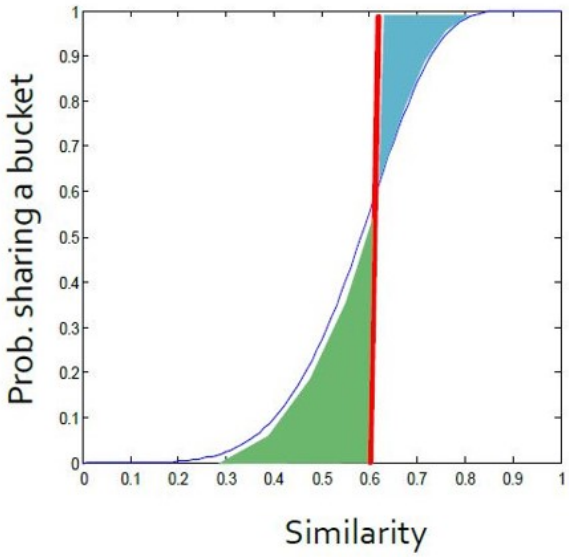

Figure 5: The probability function, the upper blue area shows False negative, the lower green area shows False positive

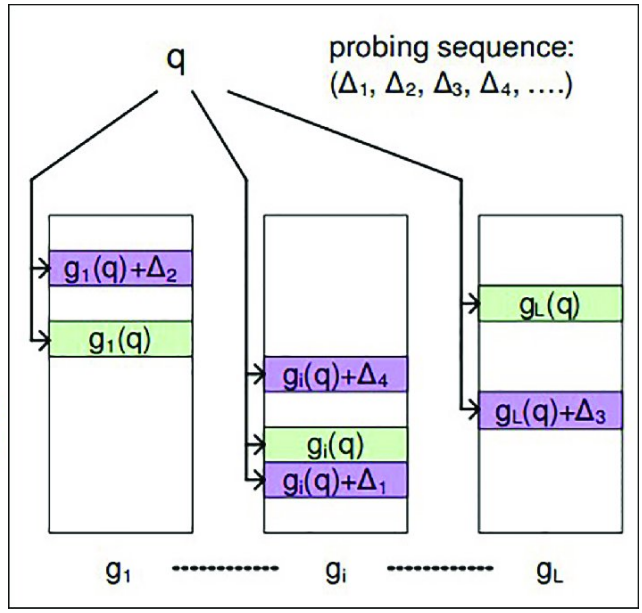

Figure 6: The idea of multi-probe

candidate nearest neighbor of q by:

$$P = 1 - (1 - c^M)^L \tag{3}$$

Figure 5 shows the characteristic of P as well as false positive and false negative. It is our nature to choose large M and L to reduce the false negative so that we will not lose any near neighbor.According to experiments results, in order to have good search performance in high dimension datasetss, hundreds of hash tables are needed[7].

However, as the space complexity of one single hash table is proportional to the number of data in datasets, It is very expensive to build and search so many hash tables [12]. Instead of building hundreds of hash tables, we use the multi-probe method to build index with a small number of hash tables. Multi-probe is based on the characteristic of LSH that the near neighbor of a query will have a high probability of being hashed to a bucket near the query(if they are not hashed to the same bucket).Under this condition, we can generate some distance vector Δ , and then probe the bucket $G(q) + \Delta$ as well as the bucket $G(q)$[16].The idea of multi-probe is showed in the figure 6.

In practice, we rank every buckets according to its Hamming distance between query buckets [23], the buckets with smaller distance will have priority to be probed[24].

3.4 LSH families for different similarity metrics

Based on different similarity/distance metric,people proposed different LSH family accordingly,there are some LSH families [18] which are most common used and the algorithms we implement are also based on them.

Definition 2 (LSH for L2 distance(L2-LSH))

$$h_{a,b}(x) = \left\lfloor (a^T x + b)/r \right\rfloor \tag{4}$$

a is a vector which each coordinate generated from normal distribution: $a_i \sim N(0,1)$, b is a constant randomly generated from uniform distribution: $b \sim U(0,r)$. we can compute the collision probability:

$$\begin{aligned} &\Pr\left(h^{L2}_{a,b}(x) = h^{L2}_{a,b}(y)\right) \\ &= 1 - 2\Phi(-r/d) - \frac{2}{\sqrt{2\pi}(r/d)}\left(1 - e^{-(r/d)^2/2}\right) = F_r(d) \end{aligned} \tag{5}$$

where $\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{\pi^2}{2}} dx$,$d = \|x - y\|_2$

the hash quality of L2-LSH is decided by $\rho(\rho = \frac{\log p_1}{\log p_2})$and we have:

$$\rho_{L2-LSH} = \frac{\log\left(F_r\left(\sqrt{2 - 2S_0}\right)\right)}{\log\left(F_r\left(\sqrt{2 - 2cS_0}\right)\right)} \tag{6}$$

Definition 3 (LSH for correlation(signed random projections))

$$h^{Sign}(x) = sign(a^T x) \tag{7}$$

a is a random vector, $a_i \sim N(0,1)$

Same as above,we can obtain collision probability for signed random projections:

$$\Pr\left(h^{\text{sign}}(x) = h^{\text{sign}}(y)\right) = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{x^T y}{\|x\|\|y\|}\right) \tag{8}$$

and infer the ρ:

$$\rho_{SRP} = \frac{\log\left(1 - \frac{1}{\pi}\cos^{-1}(S_0)\right)}{\log\left(1 - \frac{1}{\pi}\cos^{-1}(cS_0)\right)} \tag{9}$$

In practice, it is very important to choose the right similarity metrics for a specific problem, therefore we can use the corresponding LSH family to solve it. Sometimes we can also transform one similarity to another one under certain conditions.

Definition 4 (LSH for maximum inner product(simple-LSH))

$$P(x) = \left[x; \sqrt{1 - \|x\|_2^2}\right] \tag{10}$$

Inner product similarity metric is very popular in computer vision,especially for face recognition. The state of art solution for approximate maximum inner product problem is simple-LSH [17]. To note that MIPS(Maximum inner product search) is a problem which is a little different from NNS,to apply this simple-LSH framework, we must preprocess data to have them bounded by U, where $U < 1$. The algoritms is showed below: what we actually do is

Algorithm 2 Simple-LSH

Input: datasets S, qurey vector q
Output: p with the top-k inner product of q

```
Preprocess datasets S,divided by max L2-norm
Generate a SRP-LSH family H
Generate mapping function P(x)
Generate a group of hash functions G by randomly pick M hash
  functions h from H
for all p ∈ S do
    Calculate P(p),
    Calculate G(P(p)),
    G(P(p)) is the concatenation of
    (h_1(P(p)), h_2(P(p)), ..., h_m(P(p)))
    Use hash function to hash G(P(p)) and generate a bucket
    based on the hash value
end for
Calculate G(P(q)) and hash G(P(q)) to a bucket
Rank the buckets according to Hamming distance
Probe the buckets and find out the candidate p*
Compute the similarity between all the candidate p* and the
  query
return p* with the top-k similarity with q
```

transform the origin data(bounded) to a higher dimension space by this mapping function$P(x)$, because it is proved that in the origin data space we can not find a efficient LSH family for inner product similarity. After increasing the data space, we can use Sigh-LSH to calculate the angular distance which equals to the inner product in origin data space. therefore, the collision probability and hash quality is as the same as Sign-LSH.

3.5 LSH forest

As mentioned before, LSH can be considered as a method to do space partition. After we build a hash table, we have already partition the space using the hash functions in this certain hash table. Then, we hash the query point using this table, and take out all the points have the same hash value (be paritioned in the same space)as the candidate points. This leads to a connection between decision trees method[5], we can build a decision tree such that:

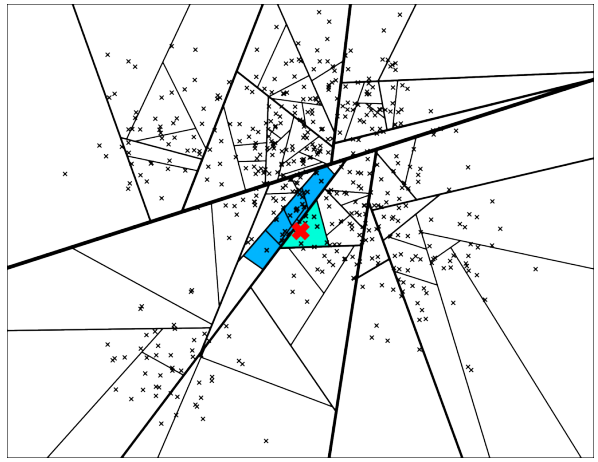

Figure 7: space partition by decision tree

1.Each internal node of the tree denotes a hash function. 2.Points go to the left sub-tree or the right sub-tree depending on the hash value [3]. 3.When a query point comes in, take all those points that go to the same (or a nearby) leaf as the query point as the candidate points.

As showed in figure 8,the leaf node of green path means the green part of space in figure 7.The sets of nodes in blue path means the blue parts of space in figure 8.This reveals two popular methods of LSH framework in practice:

Method 1 (priority query(blue path)): Take the idea of building a hash table and multi-probe the buckets which neighbor to the bucket query hashed into to generate candidate sets.We configure a threshold of the max distance and sort all the nodes of the decision tree within this distance to obtain candidate sets.

Method 2 (random forest(green path)): In LSH framework we build several hash tables and use the points in the same bucket as the query to construct candidate sets.There we build a forest of trees, each one is randomly generated and independent with others.Under this conditions, we search all the forest and focus on the exact node the query is hashed in every tree and the points in these nodes of the forest is treated as candidates.

finally we just compute and sort the candidates depends on the similarity between them and the query, and report the top-k candidates as results.

4 IMPLEMENTATION

Bootstrap Bootstrap is a simple, intuitive, and powerful front-end development framework based on HTML, CSS and JavaScript that speeds up web development. Bootstrap provides elegant HTML and CSS specifications written in the dynamic CSS language Less. Bootstrap provides a basic structure, including grid system, link styles and backgrounds.

Django Django is an open source web application framework written in Python. Adopt MVC framework mode, namely model M, view V and controller C.

Redis Redis is an open source, log-based, Key-Value database that is written in ANSI C, supports networking, and can be persisted

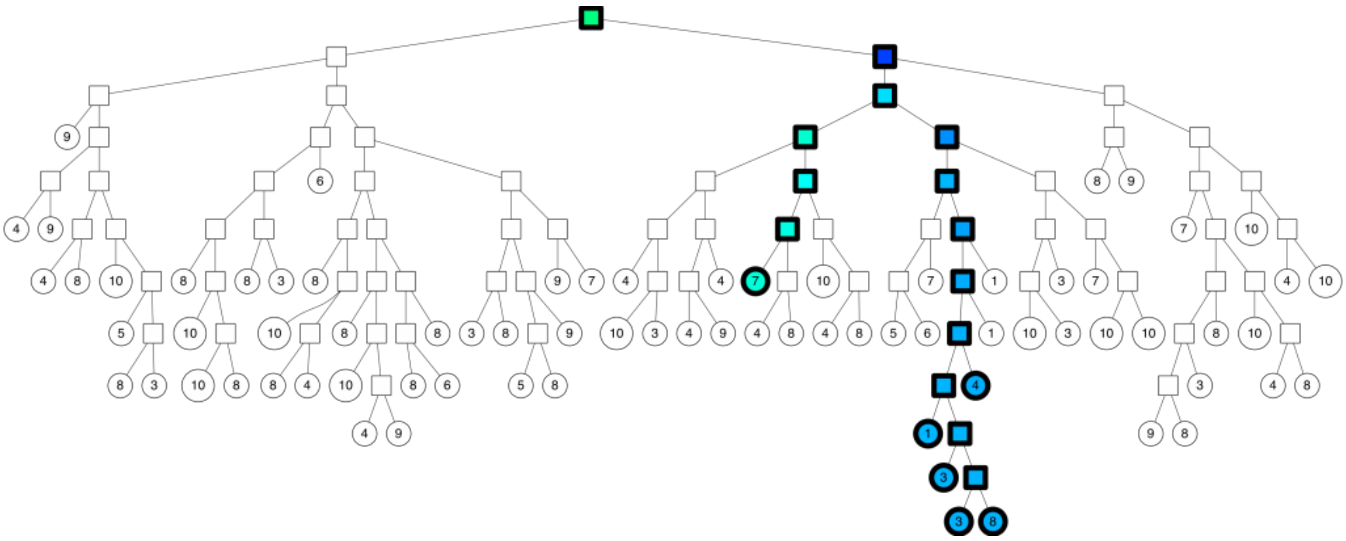

Figure 8: The corresponding tree data structure

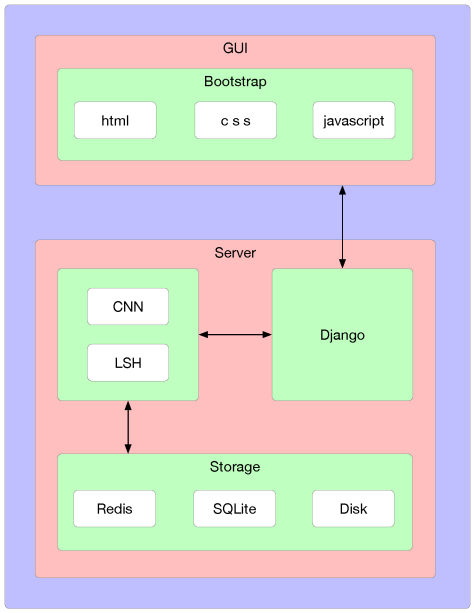

Figure 9: Image of architecture.(a) Bootstrap: show and organize the layout of web GUI. (b)Django: backend server framework used to handle request and return response. (c) CNN: extract face features from the images. (d) LSH: map the feature into different dimension spaces. (e) Redis: store the map of LSH hashing strings, features and image paths. (f) Sqlite: store the Identifier and name of the celebrity. (g) Disk: store celebrity face images.

based on memory, and provides APIs in multiple languages. Redis is a type of NoSQL, NoSQL (Not Only SQL), which refers to a non-relational database. With the rise of the Internet web2.0 website, the traditional relational database is coping with web2.0, especially the ultra-large-scale and high-concurrency SNS type web2.0 pure dynamic website has become incapable, exposing many insurmountable problems, and Non-relational databases have evolved very rapidly due to their own characteristics. The NoSQL database was created to solve the challenges brought by the multiple data types of large-scale data collection, especially the big data application problems, including the storage of very large-scale data.

SQlite SQLite is an open source embedded relational database. it is portable, easy to use, small, efficient and reliable, and takes up less resources. It is embedded in the application that uses it, sharing the same process space, not a single process. From the outside, it is not like an RDBMS, but inside the process, it is a complete, self-contained database engine. One of the big benefits of an SQLite is that you don't need network configuration or management inside your program. Because the client and server are running in the same process space. SQLite's database permissions depend only on the file system and there is no concept of a user account. SQLite has a database level lock and no web server. It requires memory, and other overhead is small, suitable for embedded devices. All you have to do is compile it to your program correctly.

5 EXPERIMENTAL RESULTS

In this section, we use the celebrity faces in the Wikipedia pages as the experimental test dataset, which contain 62,329 people images and 39,173 real faces. We conduct a experiments to show the query time, train time and accruacy results as the change of hash size and hash table count. Based on the results, we balance the accuracy, time cost to choose the most suitable hash size and table count to use in the training dataset.

For the training time evaluation, all the test set will be passed one by one to the program, and all the hash result will be saved on the Redis storage; for the query time evaluation, we choose random 1000 images in the test set to query iteratively, and the accuracy is based on the query results.

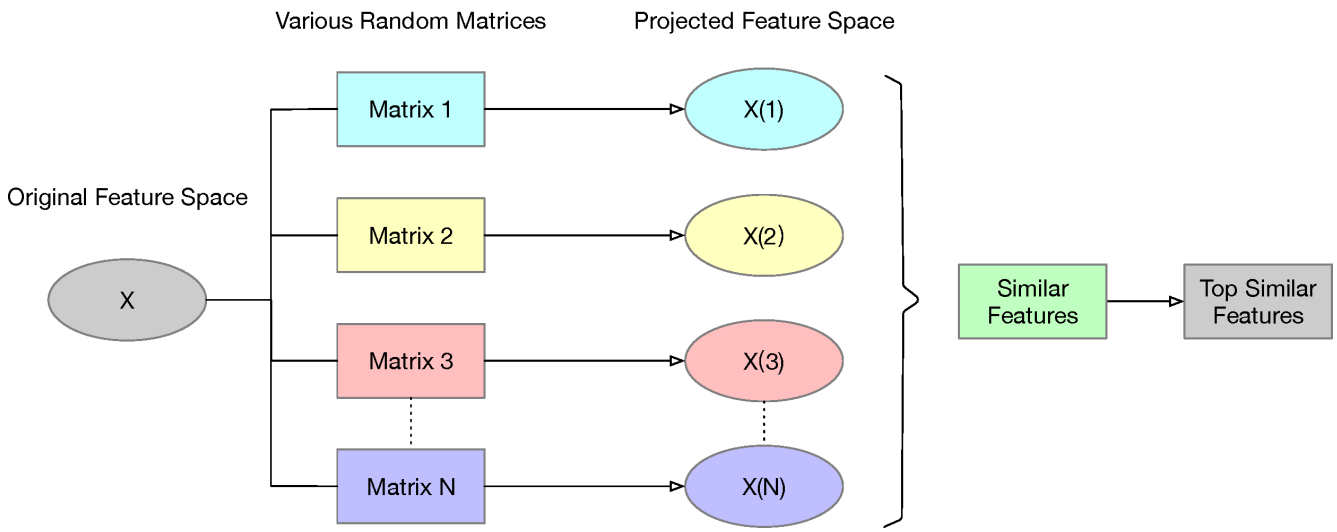

Figure 10: Random projection. Map original feature into other dimension spaces. Find similar features and calculate similarity.

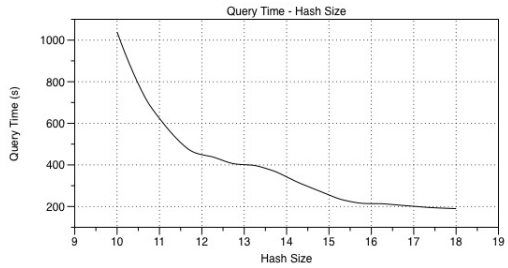

Figure 11: Query time-hash size curve.

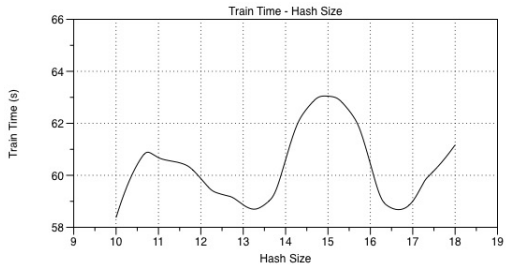

Figure 12: Train time-hash size curve.

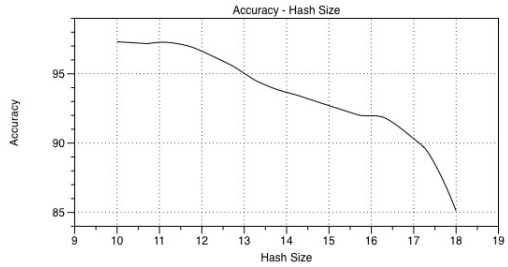

Figure 13: Accuracy-hash size curve.

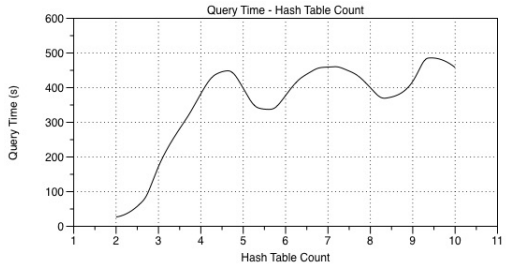

Figure 14: Query time-hash table count curve.

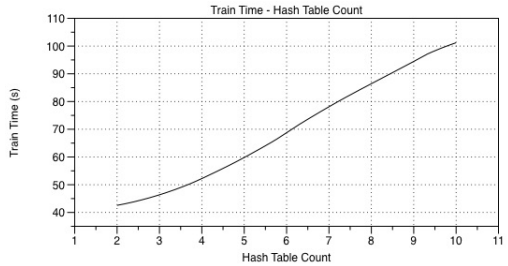

Figure 15: Train time-hash table count curve.

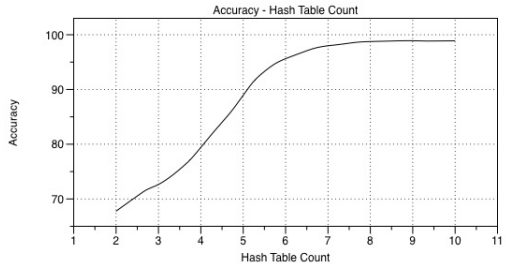

Figure 16: Accuracy-hash table count curve.

The evaluation environment is Amazon EC2 t3.medium model, which have 2.5 GHz Intel scalable processor, 50G EBS optimized disk and enhanced networking.

Tune Hash Size

The output hash of face images will be directly stored on the Redis storage in the training stage, and then will be query in the face retriveal progress. As the hash size is shorter, the query phase will be more time consuming as the there will be more images in the same bucket; the accuracy, however, will be more accurate, because the possibility of similar image in the same bucket will be higher.

As the Figure. [11,12,13] shows, as the hash size increase, the accuracy and query time decrease directly, however, the total training time are around 58-62 seconds. A middle hash size, for example, 12 in the test set will be suitable because the accuracy and query time is balanced.

Tune Hash Table Count

The hash table count is equals to the number of random planes used in the LSH algorithm, in the training phase, the program will iteratively hash the face image's feature into string with different planes, when the training phase complete, there will be different tables in the Redis storage.

As the Figure. [14,15,16] shows, it is obvious that the training time will be more time consuming as the hash table count is larger. The query time increase significantly as the hash table count increase from 2 to 4, but it fluctuate around 400s when the table count continue increase. The hash table count 7 is a perfect choice as the accuracy will not continue increase with larger hash table count.

6 RESULTS AND CONCLUSIONS

Nowadays, face retrieval in large dataset is still a challenge problem given limited space and query time. In this project, we first used deep learning to preprocess and extract feature from the image datasets, then we developed a novel framework based on LSH to find the top-5 similar faces of the query image. We also designed experiments to tune the number of hash tables and hash code length to achieve optimal query time and accuracy. Finally, a website is deployed to give a intuitive interpretation. The results of our project is showed in figure 17 and figure 18.

ACKNOWLEDGMENTS

We would like to thank Prof. Michael R. Lyu for his lectures and instructions. We also appreciate Mr. ZENG Jichuan for his tutorial and help. We would also like to thank the course CMSC 5741 of THe Chinese University of Hong Kong, which grants us the great chance of develop the project.

REFERENCES

[1] Alexandr Andoni and Piotr Indyk. 2017. 43 NEAREST NEIGHBORS IN HIGH-DIMENSIONAL SPACES. (2017).

[2] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. 2018. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823* (2018).

[3] Alexandr Andoni, Ilya Razenshteyn, and Negev Shekel Nosatzki. 2017. LSH forest: Practical algorithms made theoretical. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 67–78.

[4] Artem Babenko and Victor S Lempitsky. 2017. AnnArbor: Approximate Nearest Neighbors Using Arborescence Coding.. In *ICCV*. 4895–4903.

[5] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web*. ACM, 651–660.

[6] Binod Bhattarai, Gaurav Sharma, and Frédéric Jurie. 2016. Cp-mtml: Coupled projection multi-task metric learning for large scale face retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4226–4235.

[7] Jeremy Buhler. 2001. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics* 17, 5 (2001), 419–428.

[8] Bor-Chun Chen, Yin-Hsi Kuo, Yan-Ying Chen, Kuan-Yu Chu, and Winston Hsu. 2011. Semi-supervised face image retrieval using sparse coding with identity constraint. In *Proceedings of the 19th ACM international conference on Multimedia*. ACM, 1369–1372.

[9] Yushi Chen, Hanlu Jiang, Chunyang Li, Xiuping Jia, and Pedram Ghamisi. 2016. Deep feature extraction and classification of hyperspectral images based on convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing* 54, 10 (2016), 6232–6251.

[10] Zhen Dong, Chenchen Jing, Mingtao Pei, and Yunde Jia. 2018. Deep CNN based binary hash video representations for face retrieval. *Pattern Recognition* 81 (2018), 357–369.

[11] Nicholas Frosst and Geoffrey Hinton. 2017. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784* (2017).

[12] Sidney L Hantler, Meir M Laker, Jonathan Lenchner, and Daniel Milch. 2017. Methods and apparatus for performing spelling corrections using one or more variant hash tables. US Patent 9,552,349.

[13] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 604–613.

[14] Animesh Kumar Jha, Rupak Gupta, and Dharmender Saini. 2013. Face Recognition: A Fourier Transform and SVD Based Approach. In *International Conference on Computational Intelligence Communication Networks*.

[15] Xianglong Liu, Cheng Deng, Bo Lang, Dacheng Tao, and Xuelong Li. 2016. Query-adaptive reciprocal hash tables for nearest neighbor search. *IEEE Transactions on Image Processing* 25, 2 (2016), 907–919.

[16] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 950–961.

[17] Behnam Neyshabur and Nathan Srebro. 2014. On symmetric and asymmetric LSHs for inner product search. *arXiv preprint arXiv:1410.5518* (2014).

[18] Ryan OâĂŹDonnell, Yi Wu, and Yuan Zhou. 2014. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory (TOCT)* 6, 1 (2014), 5.

[19] Rasmus Rothe, Radu Timofte, and Luc Van Gool. 2015. Dex: Deep expectation of apparent age from a single image. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 10–15.

[20] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems*. 2321–2329.

[21] Anshumali Shrivastava and Ping Li. 2014. Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). *arXiv preprint arXiv:1410.5410* (2014).

[22] Ryan Spring and Anshumali Shrivastava. 2018. Scalable Estimation via LSH Samplers (LSS). (2018).

[23] Jinhui Tang, Zechao Li, and Xiang Zhu. 2018. Supervised deep hashing for scalable face image retrieval. *Pattern Recognition* 75 (2018), 25–32.

[24] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. 2018. Norm-Ranging LSH for Maximum Inner Product Search. In *Advances in Neural Information Processing Systems*. 2953–2962.

[25] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. 2016. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters* 23, 10 (2016), 1499–1503.

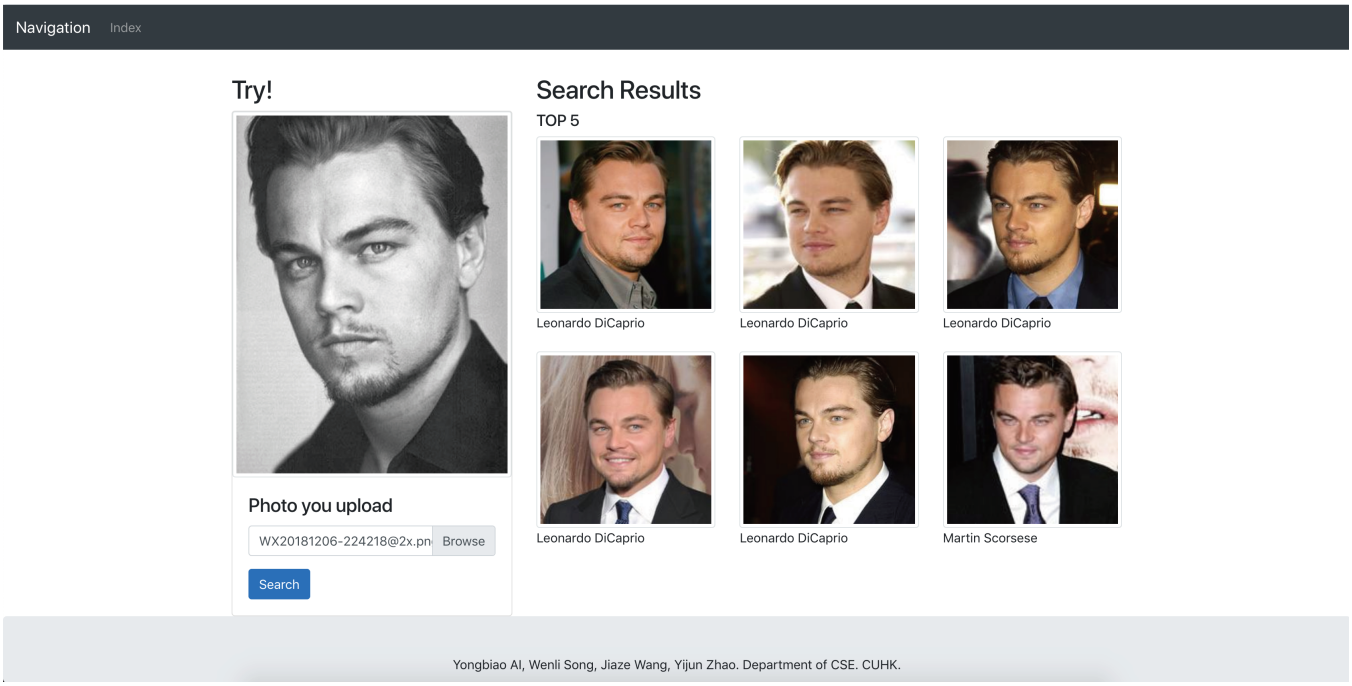

Figure 17: Example result 1 of our website.

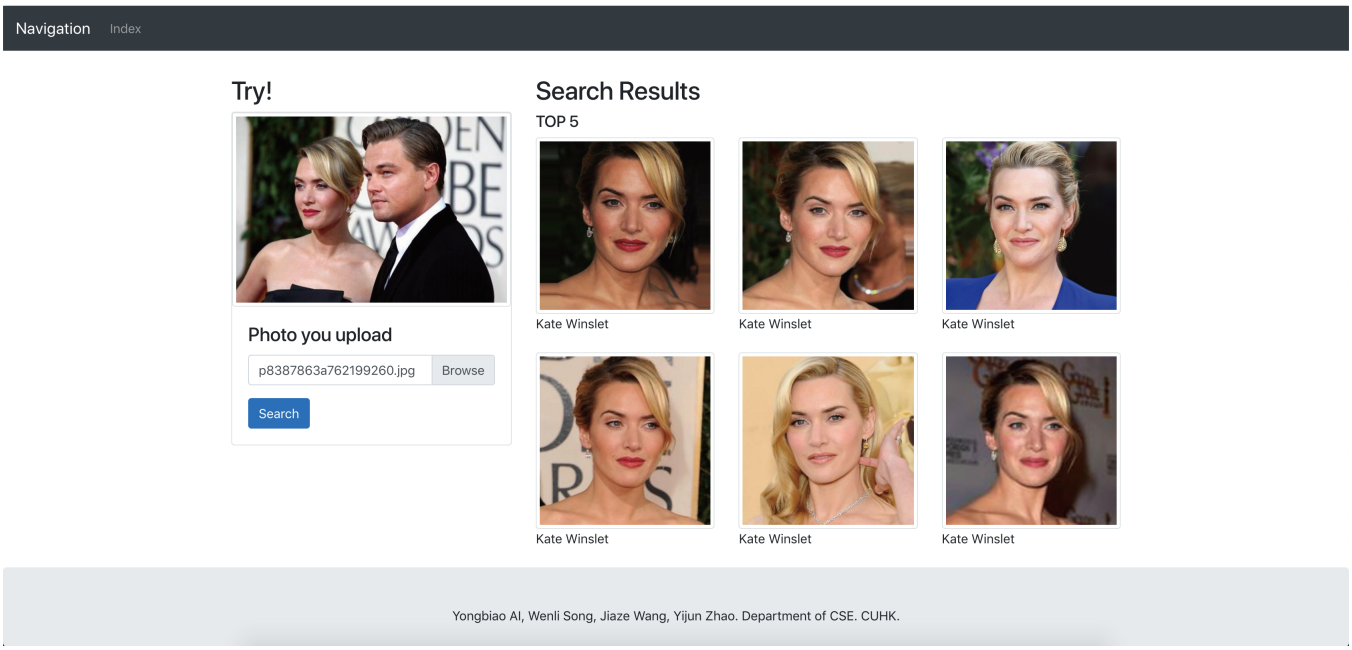

Figure 18: Example result 2 of our website.