COMP480 Final Project: GIF Searching Based on CaPSuLe Algorithm

Xin Hao, Henry Zhang

May 2021

1 Introduction

1.1 Problems

Given efficient application of CaPSuLe algorithm [1] in image matching, we are going to extend the idea of Locality Sensitive Hashing(LSH) into GIF searching. In GIF recognition, the multi-image recognition and classification is computationally hard as the large storage required with neural network such as CNN is sometimes prohibitive [2].

We leverage existing CaPSule algorithm to achieve sequential multi-image searching in GIF with low memory and high efficiency. With cheaper computation cost, it is largely applicable for cloud-independent mobile applications and fast meme matching in the future.

2 Literature Review

2.1 CaPSuLe For Indoor Positioning

"CaPSuLe: A Camera-Based Positioning System Using Learning." Since our project builds on the idea of CaPSuLe, we have explored the original positioning system in detail. In this paper, the authors use CaPSuLe for the image-based device positioning, to improve the efficiency of indoor location process. The authors use 719 training images and 153 query images to build the LSH hash table. In comparison to supervised learning, this process requires much smaller time and energy while providing approximately the same level of accuracy.

2.2 Dataset Exploration

"TGIF: A New Dataset and Benchmark on Animated GIF Description."
To run experiments for GIF classfication, we have explored available dataset for gif images. With over 100K animated GIFs from Tumblr and 120K natural language descriptions obtained via crowdsourcing, TGIF provides a clean and user-friendly gif image library. Also, with all images provided in url remotely, it saves a large amount of space to store images and train LSH table on our machine.

2.3 GIF Classifcation in Machine Learning

"GIFGIF+: Collecting emotional animated GIFs with clustered multi-task learning"
This paper uses 3D CNNs and transfer learning to enable an efficient labeling of a large set of target GIFs in terms of 17 emotion categories (i.e., tasks) and their intensity. It mainly works on the classification and matching instead of image matching. However, it provides us with few potential methods to classify and gifs through different frames. With approximately 55% benchmark AUC, we believe that there is much to improve in the area of gif image matching.

3 Thesis

We hypothesize that we can achieve efficient GIF matching through LSH algorithm by merely looking at retrieved ids from SRP hashed index.

4 Experimental Settings

4.1 Dataset and settings

The programming language we use for experiments is Python. We have used local and server to Yogi server perform the experiment.

TGIF: As mentioned in section 2.2, we use TGIF as the first dataset for our experiments, which covers a wide range of varieties and is highly suitable for a GIF image search engine. Although this data set provides a good description of every GIF, it is not very well categorized and is thus very difficult to evaluate our queries. Hence, in this project we make the test set a subset of the training set, evaluate the retrieval performance.

Self-constructed Dataset: Inspired by the dataset of CaPSuLe, we also

searched for GIFs and constructed a dataset manually. In the dataset of CaP-SuLe, there are a number of categories of data. Here, the category refers to the same location in the mall. Within each category, there are a number of photos taken from different angles and distances at the same sign. During the query stage, if an image in the same "category" is reported, then this query is thought to be accurate. Inspired from this, we manually created our own training and testing dataset with 20 categories of popular meme characters, including SpongeBob, Minions etc. For each category, our training dataset contains 10 similar GIFs of that character and our testing dataset contains GIF for each category.

4.2 Data Preprocessing

For each GIF g in the data set, we use the python PIL library to read in the image for each frame of the GIF. As GIFs vary in the their of frames, in order to prevent some large-frame GIFs to crowd our hash tables, we sample the images belonging to each GIF by removing few duplicates. Therefore, each GIF have the same s number of frames before being inserted.

4.3 Feature Extraction

As pointed out by Moon, et al., modern image matching techniques are far from being cheap, especially those utilizing neural networks. If we were to extend these techniques to GIFs (assume each GIF has i frames on average), this would required ix more resources to compute. We could potentially apply video matching tools on GIFs, but there are also problems associated with it. First, video feature detection is still relatively new and there are very few choices. Secondly, GIFs have far fewer features (such as sounds) to detect than videos. Hence, we decide to apply the widely used OpenCV package for feature extraction.

For each GIF g, we read and sample $x_1, x_2, ...x_s$ images. Then for each x_i, we extract a set of f 64-dimensional SURF features. Then, we shall apply our Locality Sensitive Hashing technique on these SURF features.

4.4 "Training" with Hash Tables

We create L hash tables, each with size 2^K, i.e K-bit keys. At each index, we use a reservoir of length R for storing ids. For every feature I extracted from a GIF, it is hashed into a K-bit hash value $H_j(I)$, for $i \in \{1, ..., L\}$. The hash function we use is Signed Random Projection (SRP). Then, the GIF id is inserted into the reservoirs indexed by $H_j(I)$ in the jth hash table. Hence, each GIF is hashed into $s*f$ values and its ID is inserted in these corresponding reservoirs.

4.5 Query

Given a query image q, we read and sample s frames of images, and extract f SIFT features from each frame of image (same as training process). Then we hash these features into $s * f * L$ keys. We then probe the reservoirs indexed by those keys and retrieve a big set of GIF ids in those reservoirs. For these ids, we rank them based on the number of times it appears with this query, and report the top GIF.

4.6 Hyperparameters

From the property of LSH, K should be at least greater than $log(N)$ (N is the number of elements in total).
As shown in Figure 1, the selection of hyperparameters has significant effect on the performance of classification. For our experiment, we chose k = 14 and 15 for 500 images inserted. As increasing k by 1 means doubling the time for entire training process from 2^14 to 2^15, we only experimented with 14 and 15.
For the selection of L, it we used 15, 30, and 45. As expected, the larger L is, the higher accuracy LSH provides.
In order to maximize time efficiency and correctness, we find that K plays a more important role to improve accuuracy. This is due to the distributive property of our dataset, as decreasing the collision rate in each hash table will provide more accurate query score.

5 Experimental Result

5.1 Accuracy

"Training" Accuracy for TGIF
In the original TGIF dataset, we do not have a clear definition of same "category". As each gif is different with its own character, we can only measure the accuracy of our algorithm by adding the same image into the hash table. As we use the same image, we use "training" accuracy for reference.
We tried different combinations of K and L when running experiments on the TGIF dataset. As we were limited by our availability of machines and time, we decided to observe the performance on the first 500 GIFs in TGIF. First, it is clear to see as the size of our training data increases, a higher K and L are required generally to lower the number of hash collisions in the hash tables. We then observe the accuracy curve as we increase the size of the training set. As shown in Figure 1, either a higher K or a higher L would results in higher accuracy. K has a bigger impact on the accuary: When K is less than or equal to 14, the accuracy drops drastically after the training set reach 250. Overall, the "training" accuracy is really high when k is 15 and l is 30.

"Testing" Accuracy for Self-constructed dataset
With our self-constructed dataset, we checked the result manually comparing

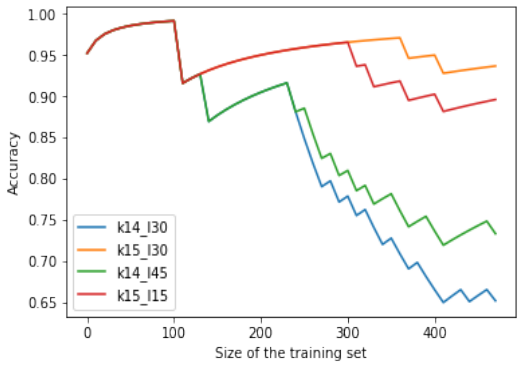
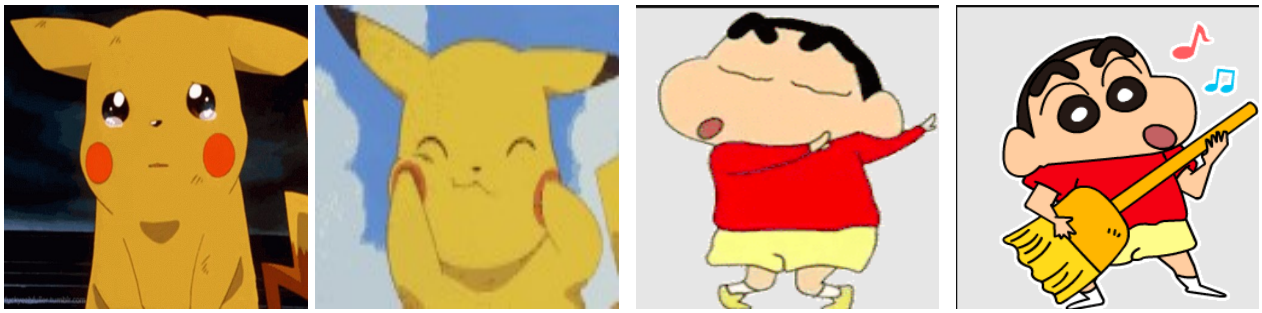


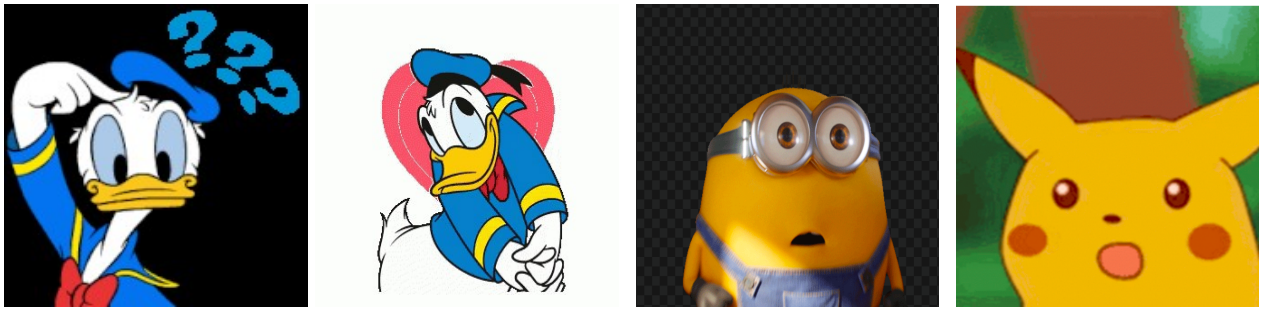
Figure 1: Results on part of TGIF

the top predictions. The predicting behavior varies with different levels of similarity between the query GIFs and training GIFs. For similar GIFs, the features extracted by SIFT will have higher cosine similarities, and are thus more likely to be retrieved. For dissimilar images, their features have quite low cosine similarities, so even the finest parameter tuning cannot help. We know that SIFT features are scale invariant and the key points are also assigned with orientation. Hence, the ideal similar GIFs should contain the same object from the same point of view, but with different scales and orientations. We provide two pairs of examples below. It is easy to see that the two GIFS in Figure 2 are similar, and indeed our LSH reports the desired predicted GIF. However, Figure 3 shows two GIFs that we initially believed to be similar. A closer look shows that one GIF is the front face of the character, and the other is the side face.



(a) Query GIF (b) Predicted GIF (c) Query GIF (d) Predicted GIF

Figure 2: Similar Prediction: Pichaku & Crayon Shin-Chan



(a) Query GIF (b) Predicted GIF (c) Query GIF (d) Predicted GIF

Figure 3: Less similar Prediction

5.2 Time & Space

Overall, our LSH algorithm performs well as we expected.

5.2.1 Time

As we are coding in python, the time performance is inefficient with python built-in data structure. As we run local on CPU devices, with k = 15, l = 20, and 50 SIFT features extracted for each frame, each gif takes approximately 15 to 20 seconds to get inserted.

5.2.2 Space

With non machine learning algorithm, we successfully save a large amount of space by constructing hash tables and storing just the GIF IDs.
As we loaded our hash tables into a csv file, the total size it takes is less than 8 MB for 100 images inserted. This indicates huge scalability to save a large amount space for storing GIFs if needed.

Also, both time and space are largely dependent on the chosen hyperparameters. With larger K and L, both time and space will increase linearly, while the accuracy of prediction increases exponentially as expected. Therefore, to further develop this project into mobile devices, we would further determine the trade-off between time, space and accuracy.

6 Conclusion

6.1 Feature Analysis

As analyzed in section 5.1, image feature extraction plays a key role image/GIF matching. We have shown that with efficient space and time, LSH storing the GIF ids matches similar GIFs with more than 90% accuracy. In future projects,

we can explore different feature extraction algorithms, including those involving neural networks, and evaluate their performance.

6.2 Applications & Future Improvements

There are many future applications for this project. As GIFs are used more frequently, GIF matching is widely applicable to social media, video platform, etc. With LSH based hashing, it largely improves the efficiency and space to run GIF matching, especially on mobile devices.

7 References

[1] Moon, Yongshik, et al. "CaPSuLe: A Camera-Based Positioning System Using Learning." 2016 29th IEEE International System-on-Chip Conference (SOCC), 2016. Crossref, doi:10.1109/socc.2016.7905476.

[2] Li, Yuncheng, et al. "TGIF: A New Dataset and Benchmark on Animated GIF Description." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, doi:10.1109/cvpr.2016.502.

[3] Chen, Weixuan Rudovic, Ognjen Picard, Rosalind. (2017). GIFGIF+: Collecting emotional animated GIFs with clustered multi-task learning. 410-417. 10.1109/ACII.2017.8273647.