Ruoyu. Yang (5011499), Haonan .Liu (5094756)

Implementation of Video Google

***Abstract*—Through utilising the principle of google text retrieval, we developed video preprocessing and object search engine. After inputting the object into the system, video Google system will return all the frames with the outlined object and rank them by relevance with the object. In the start, we represent the object by SIFT descriptors so that objection can be recognised by the system despite their occlusion and illumination in the video. We generated index for descriptors in the each frame of video by means clustering algorithm. Then we calculate a vector for each frame whose components are weighted word frequencies. After users outline an object in movie, search engine also generate a query vector for the outlined object. After calculating the angle between query vector and frame vector, the best matches result are returned.**

The Video Google systems is implemented in python, and demonstrated by two movie clips.

Project Goal

The goal of the project is, after the user selected an object contained in video, the video Google system can allocate the frames including the outlined object and rank them by their relevance at reasonable speed and accuracy. By borrowing ideas from text retrieval, this project explode feasibility of efficient object recognition in video. Although object recognition in image has reached some progress in the past few years. It is still very challenging for retrieval in video, because the outlined object usually is moving or overlapping with other objects. To address this issue, we define the appearance of an object as show up in three continuous frames. If they show up in less than three frames, the program filter its appearance as noise. SIFT descriptor which is invariant to the change of lightness and view angle are computed for the outlined object. Same descriptor is calculated for each frame. By finding nearest vectors among frame vectors for the query vector, video google returns frames containing outline object.

Review of text retrieval method

Text retrieval generally consists of two stages: preprocessing stage and retrieval stage. In the preprocessing stage, firstly, the search engine parses uses' input into words. Secondly, the words are represented with their index. For instance, "runs", "ran", "running" now are represented by "run". Thirdly, removing the generally common words, such as "is", "the" and "an" are removed in sentence "the boy is eating an apple". Since it appears in most sentences user input, so these words are not useful in text search and they will make searching more tedious and with low accuracy. For other infrequent words, they are assigned a unique identifiers. Meanwhile, all the webpages are represented with a vector with component given by the frequency of the word in the website. For example, if "apple" appears three times in the webpage, the vector of this webpage will be assigned 3 in the corresponding location which is also the identification number of the word "apple".

The preprocessing stage generally is completed before any retrieval task, but it always constantly updates when new webpage is created and some webpage is edited.

In the retrieval stage, when a text is input by user, search engine compute the word frequency of input and represent it in vector. Then search engine

calculate the angle between input vector and document vector whose components are weighted word frequencies. Finally, search engine will return search result ranked by angle from small to large.

Paper outline: In the next section, section III we introduce the processes we used in this project. In section IV, we described scope of the project and how we completed it. In the last section, we performed video google in two movies.

We used the ideas of text retrieval into video search. And we implemented the ideas in Python, and used functions from opencv library.

I. Problem Decomposition:

In this chapter, the detailed project plan will be provided, in order to reduce the complexity, lower the difficulty, and deliver a better project.

The basic idea in terms of project implementation is decomposing the whole project into a number of small task. The following contents will give the information of decomposition of the project:

- Extracting frames from the video
- Preprocessing extracted frames
- Detecting feature points and computing the descriptors
- K-means clustering
- Creating stop list
- Building cluster index
- Building query vector
- Comparing the distance between query vector and frame vector

Apparently, splitting the project into several small tasks will make us have deeper understanding of the structure of the project, so that we can make a proper plan for implementing the project, and make everything under control.

II. Scope of the project:

In this chapter, the details of every part in the project will be provided. Basically, the main programming language we used in the project is python 2.7, the following library is what we included:

- Opencv 3.1.0 dev: This library is for video and image processing
- Scikit-learn 0.1.8: This library is for k-means clustering

4.1 Project preliminary work

- Time: Monday 10th Oct (week 11) — Friday 14th Oct (week 11)
- Detail specification:
    - Before implementing the project, it took our group about one-week-time doing literature survey. This includes the main algorithm we used, the right data structure we selected, and the major library we included in the project.

4.2 Extracting frames from the video

- Time: 2 hours, Saturday 15th Oct (week 11)
- Detail specification:
    - The first and foremost thing that we have to do is finding a good video clip, which has high definition of resolution quality, appropriate light condition. The reason for these is to stand out feature points.
    - After finding the right video clip, opencv has video capture and retrieving specific video frame function. In order to avoid high similarity of frames, which may add complexity in image matching, the frames are captured every one second.
    - As the result, we select a video clip from South Park, season 20, episode 2. This is a 3-minute-cartoon, with 23 frames per second. The number of extracted frames is 51.

4.3 Preprocessing extracted frames

- Time: 4 hours, Monday 17th Oct (week 12)
- Detail specification:
    - For each extracted frames, the work of

noise reduction is implemented, by using the opencv library function: "cv2.fastNlMeansDenoisingColored()".

• Apart from denoising frames, unstable regions, which exist no more than three continuous frames, are detected and removed as well.

• As the result, frames with stable regions and without noise features are acquired.

4.4 Detecting scale invariant regions

• Time: 2 hours, Tuesday 18th Oct (week 12)

• Detail specification:

• In this step, a SIFT descriptor from opencv library is created and used to detect SIFT feature points for each extracted frames.

• After the detecting work finished, the compute function which also from opencv library is called, therefore we get SIFT descriptors for every feature points, the SIFT descriptor is a 128 dimension vector.

• As the result, a list of these SIFT descriptors is acquired, the processing time for this step is about 92 seconds.

4.5 K-means clustering

• Time: 6 hours, Wednesday 19th Oct (week 12)

• Detail specification:

• In this step, a k-means classifier is created from skit-learn package, the classifier is used to divide these SIFT descriptors into a number of clusters. According to the analogy of google text retrieval approach. These clusters act as the stem of words.

• In the project, base on limited time conditional, the number of clusters we set is 500. As the result, the processing time in this step is 551 seconds.

4.6 Creating stop list

• Time: 1 hour, Thursday19th Oct (week 12)

• Detail specification:

• In google text retrieval approach, common words, such as "a", "an", and "the" will be removed, since almost every document has these terms. Therefore, in the project, clusters with high frequency are removed as well.

• Since there are 500 clusters in terms of k-means classifier, after calculating frequency for every cluster, top 10% clusters are appended into stop list. In future, when dealing with cluster, operation will omit if the cluster ID is in the stop list.

4.7 Building cluster index

• Time: 3 hour, Thursday19th Oct (week 12)

• Detail specification:

• In this step, cluster-(frameID, times) index and frameID-TDIDF vector are created.

• For the cluster-(frameID, times) index, it is necessary to know the occurrence information for each cluster in every extracted frame. The appear information are stored in tuple format: (frameID, number of occurrence). Therefore, the major of the data structure will be a dictionary, with frameID as the key, and a list, which contains a number of tuples, as the value.

• For frameID - TFIDF index, we need to calculate TFIDF vector for every extracted frame, the detailed algorithm will be provided in the algorithm chapter. As the result, the data structure for this index is also a dictionary ,with frameID as the key, and TF-IDF vector as the value.

4.8 Building query vector

• Time: 1 hour, Thursday19th Oct (week 12)

• Detail specification:

• When inputing a image as the keyword, the system will use SIFT method to detect feature points, and compute the SIFT descriptor.

• After acquiring the 128-dimensional

descriptors, K-means classifier, which we used as the trainer, will be used to predict the labels.

• The result label list will be used to generate cluster appearance vector, for each dimension, it will represent the times of appearance (omitting the labels which are in the stop list).

• After generating the cluster appearance vector, we will use TF-IDF method to weight the vector, the result is a unit vector, which has the same dimension as the cluster appearance vector

4.9 Comparing distance

- Time: 1 hour, Thursday19th Oct (week 12)
- Detail specification:
  - Since the frameID-TFIDF vector index has been created, the vector form one specific frameID could be retrieved.
  - For each TFIDF vector, it multiplies the query vector. Since all vectors are unit vector, the result is the distance between these two unit vectors.
  - Comparing the result, and rank them in descending order. For the top 5 value, its frameID and the value will be displayed.
  - As the result, retrieval and calculation word is easy for the computer, it only takes 0.3 seconds to get the result.

III. ALGORITHM AND DESIGN

In this chapter, detail information about major algorithms and the design in the project will be illustrated.

5.1 Core algorithm

There are three major algorithms we used during the implementation of the project:

5.1.1 Invariant Descriptor: SIFT

SIFT (Scale Invariant feature transform) descriptor is developed by David Lowe in 1999. SIFT is a type of descriptor local features which is invariant to difference in illumination, noise and scale. SFIT descriptors even are able to robustly identify the outlined object in scatter and partial occlusion.

Generally, in video search, SIFT descriptors of outlined objects are first extracted from all frames and stored in a database. The features of outlined object is recognised by individually comparing each feature from the database. By calculating the Euclidean distance of feature vectors, threshold set before will filter out the images with outlined objects.

In details of algorithm implementation part, in order to detect SIFT descriptors, the image is convolved with Gaussian filters at different scales, and then the difference of successive Gaussian-blurred images are taken. Keypoints are then taken as maxima/minima of the Difference of Gaussian that generates at different scales. The Difference of Gaussian of image D.

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma)$$

L is a the convolution f original image I with the Gaussian blur G at scale.

Generally, for our test video, there are generally 500 to 700 SIFT descriptor per frame. The index words are critical regions of frame.

In order to boost the efficiency of video Google.

5.1.2 K-means Clustering

K-means, which is also referred as Lloyd's algorithm, is a clustering method for vectors, it clusters n observations into k partitions. In each partitions, its observation is closet to the centroid of this partitions comparing to centroid of other partitions. This algorithm is very time-consuming,

and the selection of k value is very tricky. Mathematically, k-means algorithm can be described as following:

Given n d-dimensions observation vectors (x1, x2, …, xn), the aim is to divide it into k partitions S = {S1, S2, …, Sk} which minimise the within-cluster sum of squares, sum of distance of each points to the centroid of its partition. And within-cluster sum of squares (WCSS) can be expressed as following equation:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}\in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

For starter, the initial set of k-means are randomly m1(1),…,mk(1) selected. Then algorithm keeps doing following two steps until the partitions S = {S1, S2, …, Sk} no longer change:

[1]. Assign each observation to the cluster whose mean yields the least within-cluster sum of squares (WCSS).

[2]. Calculate out the new means to be the centroid for observation in new clusters.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

The two difficult parts are the selection of k value and initialisation of k cluster centroid. Since these will highly effect running time of k-means algorithm. K value is generally choose based on situation. Conventionally, users usually input a range of k-values for k-means to run independently and compare the clustering sets they got. Actually, if there is no clear requirement about the number of group data need to be divide into in real situation, the perfect k-value is usually uncertain. Users usually make decision based on the distribution of data, scale of data set and scattering level of data. The methods commonly used are Elbow method, x-mean clustering, and cross-validation. The Elbow method plots the percentage of variance against the number of clustering. User choose a number of cluster so that adding another cluster doesn't give significantly higher level of scattering (percentage of variance). In the plot, the marginal gain will drop and give an angle in the graph which looks like "elbow". X-mean clustering repeatedly subdivides until the best resulting splits reached. Cross-validation method partition data set into v subsets. Each subset is used as test set for clustering the model built by the rest v-1 training sets and sum of the squared distances is calculated for test set according to the model.

Two commonly used initialisation method is Forgy and Random Partition. Forgy method randomly pick k observation vectors as the centroids of k partitions. While random partition method randomly assigns a cluster to each observation and then proceeds step 2 to compute the initial mean of observation vector of cluster to be its centroid. After the initialisation method, the algorithm does above two steps continuously until reach the convergence.

It is worth mentioning that k-means algorithm is a heuristic algorithm, so convergent result is not promised as global optimal result and this depends on the choose made at initialisation step. So a conventional way to do this is running k-means algorithm with different options of initialisation.

Although, k-means algorithm can be extremely time consuming at some situation, but time complexity is usually polynomial.

5.1.3 TF-IDF

TF-IDF stands for "term frequency - inverse document frequency", it is a weight calculation algorithm which is often used in document retrieval and text mining (tf-idf: A single-page Tutorial - information retrieval and text mining, no date).

This weight of TF-IDF is a statistical measure used to evaluate the importance of a word to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the TF-IDF weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

To compute the TF-IDF weight, typically, it is necessary to compute the two following part:

- TF: term frequency, which measures the frequency of a term occurs in a document. Since every document is different in terms of its length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length, the formula is: TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
- IDF: Inverse Document Frequency, which measures how important a term is in the whole engine. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following: IDF(t) = log_10(Total number of documents / Number of documents with term t in it).

In the project, each cluster is treated as a stem of word, and each frame is viewed as a document. Therefore, for every frameID, it is needed to calculate both TF and IDF weight vector. Since there are 500 different clusters and top 10% most frequent clusters are in the stop list. The dimension for each TF vector and IDF vector is 450. Then we can calculate the TF-IDF vector by multiplying each dimension, then standardising it, and get a 450-dimensional unit vector for every frame ID.

When inputing a new image, after the labels are predicted by k-means classifier, the query TF-IDF vector is calculated in the same way as the above one. Therefore, by calculating the product of the query vector and each frame vector. Distance of the query image and extracted frame can be identified.

5.2 Project design

According to the association between each task, and the complexity in implementing these works, the whole project is divided into four key parts:

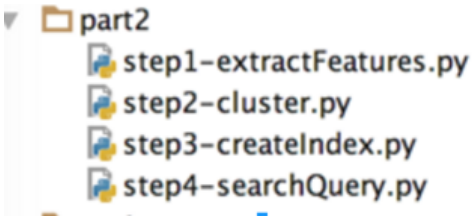
• Feature extraction: In this part, the system extracts frame, reduces noises, removes unstable regions and computes the SIFT descriptors.

• K-means training: This part aims to use k-means classifier to train these SIFT descriptors.

• Index creation: This part uses the k-means classifier to create stop list, and TF-IDF index.

• Query search: This part reads the query image, detects SIFT descriptors, predicts clusters and retrieves the related frames in ranged order.

I. Justification for design decisions



In this chapter, a detailed illustration of design decisions for the project will be provided.

To begin with, it is very time consuming for the system to deal with the first three part. It usually takes 100 seconds, 600 seconds and 70 seconds respectively. On top of that each parts of the project must base on the previous stages' result. Therefore, in order to not impacting the previous work, it is better to write result of each stage into a file, and read these file at the beginning of part2, part3 and part4.

Secondly, base on decomposition of the project, which has been mentioned above. There are some sub-tasks which require the same logic and data structure. Making these sub-task into one stage makes it easier to manage and control the project.

Test and Result

In order to test Video Google developed, we used 2 minutes movie clip of South Park as sample video and tried to search two outlined objects: signboard of South Park Elementary school and "bebe" painting on the wall. It is worth mentioning that these two test objects are both in rectangle shape, this makes them could be represented SFIT descriptors easier and their retrieval easier.

As shown in above images, in the first test, the first-ranked search result perfectly and accurately circle out the outline object in frame 0001 with relevance score 0.682. The second search result is located in frame 1501 with relevance score 0.6954. The searching scope also includes extra areas on the left of outline image.

The second test, searching poster on the wall "bebe", the search result ranked in the first place fails to circle out the outline object accurately. It includes large extra area and is located in frame 0901 with relevance score 0.105. The second search result is located in frame 1101 with relevance score 0.1031 which also contains some extra areas below it. However, for the third test, the outlined object is perfectly and accurately circled. Obviously, the third one is best searching result. We have no clue why it is ranked at third place. In future development, we should further address this issue.

Conclusion

The analogy of text retrieval approach is

meaningful in the project. We have built the index by using the extracted frames. For the query image, it only takes approximately 0.1-0.5 second retrieving the relevant frames. According from our observation, despite there are some mismatch, such

as character's face, the majority of the query image can be matched with high similarity.

Further extension

In terms for future suggestions and improvements, changed can be made from the following parts:

- Technical issue: The only feature we detected in the project was SIFT, it is obvious that different kinds of features, such as maximum stable region should be detected and computed in the project, in order to stand out the features for every frame.
- Hardware issue: The computer we used in the project is MacBook pro 13 (2013), with an i5 CPU and 8GB 1333HZ memory. As the result, it took us more than ten minutes in k-means clustering. If we have the condition to increase the number of clusters, it is likely to get a more precisely result.
- Software issue: The opencv library we used in the project is opencv 3.1.0 dev, which has some bugs in image displaying and MSER feature detection. If we had more time doing this project, we would swap this version into a stable version, such as opencv 3.0.0 stable.

Contribution and Subdivision of tasks

There are two group members in this project, which are Haonan Liu and Ruoyu Yang. Haonan is good at logic and familiar with python. Ruoyu is strong in machine learning, writing documents. Specifically, Haonan is the main programmer, which are data structure designing, algorithm implementation and bug fixing, while Ruoyu is responsible for information searching, data calculation and result analysing.

Overall, both the two teammates have done the equal contributions in the project.

References

1. *J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos", ICCV, 2003.*
2. *L. Shoval, R. Haddad Approximate Nearest Neighbor - Applications to Vision & Matching*
3. *Tf-idf: A single-page Tutorial - information retrieval and text mining (no date) Available at: http://www.tfidf.com/ (Accessed: 23 October 2016).*
4. *K. Sparck Jones. "A statistical interpretation of term specificity and its application in retrieval". Journal of Documentation, 28 (1). 1972.*
5. *G. Salton and Edward Fox and Wu Harry Wu. "Extended Boolean information retrieval". Communications of the ACM, 26 (11). 1983.*
6. *G. Salton and M. J. McGill. "Introduction to modern information retrieval". 1983*
7. *G. Salton and C. Buckley. "Term-weighting approaches in automatic text retrieval". Information Processing & Management, 24 (5).*
8. *H. Wu and R. Luk and K. Wong and K. Kwok. "Interpreting TF-IDF term weights as making relevance decisions". ACM Transactions on Information Systems, 26 (3). 2008.*
9. *Lowe, David G. (1999). "Object recognition from local scale-invariant features" (PDF). Proceedings of the International Conference on Computer Vision.*
10. *Lowe, David G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". International Journal of Computer Vision.*
11. *Lindeberg, Tony (2012). "Scale invariant feature transform". Scholarpedia. 7 (5)*
12. *Richard J. Dean W. (2008). "Applied Multivariate Statistical Analysis". Six Edition*
13. *MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press. pp. 281–297.*
14. *Determining the number of clusters in a data set (no date) Available at: https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set (Accessed: July 2016).*
15. *Catherine A. Sugar; Gareth M. James (2003). "Finding the number of clusters in a data set: An information theoretic approach". Journal of the American Statistical Association. 98 (January): 750–763*