CBIR

April 18, 2021

1 Content Based Image Retrieval

CBIR follows basic following steps

- Getting the Dataset *(referenceImages)*
- Extracting features from those reference images
- Get the query image *(queryImage dataset)* and extracts its features same way as of reference images
- Calculate similarity between extracted features of each reference image and given query image
- Display the most similar image, that is image having least distance between features

1.1 Extracting features of reference images

There are few ways in which we can extract key features and descriptors from an image.

- SIFT
- SURF
- KAZE
- ORB

OpenCv library is used to perform image reading as well as extracting the features. Using SURF was not possible as it patented and removed from OpenCV 3.0+. Hence, I have kept more emphasis on using SIFT and comparing run time with different algorithms, that is, training the images.

For Extracting features first we need to detect keypoints on image. The number of detected keypoints can be different depending upon image, so we need to add some clause to make our feature vector always same size(this is needed for calculation, because we can't compare vectors for comparing similarity of different dimensions)

Following is the image of descriptors*(using 128 keypoints)* extracted from particular beer bottle image using SIFT, ORB and KAZE respectively. This indicates that SIFT is better in extracting descriptors

Following is the table comparing runtime of training 755 reference images on my local machine (intel i7-7th gen, 8GB ram):

SIFT	ORB	KAZE
18 mins	2.5 mins	80 mins

This indicates clearly KAZE takes huge amount of time, whereas ORB appears to be really fast.

1.2 Calculating Similarity

With obtained descriptors, there are many approaches for comparing similarity between query image and reference images. Some of them are -

- Calculating the basic cosine distance obtained matrix of descriptors of reference images and descriptors of query images
- Using BruteForce Matching
- Using FLANN based Matcher

In first approach it is considered that least the cosine between two images, better is the match. The latter one is considered to be working more faster than BruteForce method for large datasets.

For Flann based matcher, function 'knnmatch' is used. This function finds the 'k' best matches of number for each of reference descriptor and query descriptor. This computation is performed for each reference image.

To keep only the strong matches we can use David Lowe's ratio test. Lowe proposed this ratio test in order to increase the robustness of the SIFT algorithm. The goal of this test is to get rid of the points that are not distinct enough.

Applied Lowe's test is as follows:

```
# store all the good matches as per Lowe's ratio test.
good_matches = 0

for m,n in matches:
    if m.distance < 0.5*n.distance:
        good_matches+=1
good_matches_list.append(good_matches)
```

Above is the created loop that defines which matches will be discarded and which matches will be preserved. So, if m distance is less than 50% of n distance, then the descriptor for that particular row will be preserved. On the other hand, if m distance is greater than 50% of n distance, then it's probably not a good match, and the descriptor for that particular row will be discarded. Therefore less distance means a better match.

Following this only index of reference image with maximum number of good matches is retrieved and displayed.

```
print("Matched Reference Image")
#Index of reference image with maximum number of good matches is retrieved and displayed.
show_img(ref_names[good_matches_list.index(max(good_matches_list))])
```

Following are the pair of the reference image retrieved when given query image

- Using SIFT with FLANN based matcher. Same Spirit bottle is retrieved based on query image

- Using KAZE with FLANN based matcher. The retrieved image seems to be very different as compared to query image

1.3 Changes that can be done:

- We can increase number of keypoints vector *(currently 128)* that are initially extracted from feature extraction algorithm
- In order to try to increase the precision or the quality of the matches, we can do tweaks with FLANN parameters.
- We can change number of FLANN trees, by default it is 5
- We can also increase FLANN index parameter checks, by default it is 50. But there is great possibility of increasing computation with very little change in quality of matches

1.4 References

1) https://towardsdatascience.com/bibirra-beer-label-recognition-8546c233d6f4

2) https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html

3) https://michellelynngill.com/presentations/2017_GillMichelle_SciPy_WINE-O_AI.pdf

4) https://towardsdatascience.com/build-an-image-search-engine-using-python-ad181e76441b