Content Based Image Retrieval System Using SIFT Extraction

Deniz Rasim Ulug
School of Computer Engineering
Middle East Technical University
Ankara, Turkey
Email: deniz.ulug@metu.edu.tr

1. Introduction

In this report we try to tackle the Image Retrieval problem using SIFT[3] extraction based on Bag-Of-Words solution. The Image Retrieval problem is about, given an RGB image, returning the query images which are most similar to the given query image, with similarity scores. We also want this retrieval to be based on contents of the pictures themselves, and therefore be as invariant as possible to regular geometric transformations. For a solution, we will build a Visual Dictionary, and then represent each image as a vector which shows how much of each visual word the image contains. We will then use these vector descriptions, or ”feature vectors”, to calculate a basic euclidean distance to show similarity.

2. Architecture

Our architecture has 3 distinct steps in it. First, we extract all SIFT features from all images in database. Then we cluster this features in to K bins using K-means algorithm. With this we end up with K visual words and those K visual words forms a visual dictionary for us. Then using this dictionary, we extract K-size feature vector from each image, we will call these vectors BOF-vectors. Then for a given query image, we compare query image's BOF-vector with each BOF-vector in database, and BOF-vectors with smallest distance gives us our result; the similar images to the query image. Let's elaborate each step one by one.

2.1. Feature Extraction

First we apply the SIFT algorithm and extract some number of features form each image. For this, we either use a regular Sift algorithm, or another approach called Dense Sift.

Regular Sift algorithm first extracts variable number of key-points from an image. The algorithm selects these key-points using Differences of Gaussian method, in English this DoG approach looks for points in image where a movement in any direction results in a high change in pixel values. The algorithm also returns a size for each key-point, which again the size which maximizes a laplacian filter around the key-point is chosen. Tough SIFT algorithm instead again uses Differences of Gradients in size dimension to approximate a laplacian filtering.

For Dense Sift, we select key-points manually, at regular intervals which is defined by step size parameter. So for a 100x100 image with step size as 10, we would have $(100 \times 100)/10 = 1000$ key points.

Then for all key-points for a image, SIFT extracts a ”local descriptor” from each key-point. This ”local descriptors” are extracted from an area with size proportional to the key-point's size. For sift this is extracted, for dense-sift this is equal to step-size.

Each local descriptor is a 128-element vector. Therefore an image with N key-points gives us N feature vector where each feature vector is 128-dimensional. Concatenating all feature vectors from all images gives us, for example with regular sift with default parameters, around 3 million feature vectors.

2.2. Building Visual Dictionary

Now that we have all feature vectors of all images, we need to build a ”visual dictionary” from this 3million x 128 tensor. For this, we cluster these feature vectors in to K bins. Naturally we use a K-means algorithm for this purpose. K-means algorithm groups these feature vectors in to K clusters and returns K centers, a 128-element vector in our feature space. We take those K centers as our ”visual dictionary” where each center is a ”visual word” for us.

2.3. Image-Feature pairs

Now that we have a visual dictionary, where each visual word in it is a 128-element vector; for a given image we may extract what we call a ”BOF-vector”. BOF stands for ”Bag of Features”.

For this purpose, for each image, we apply the following steps:

First we construct an empty BOF-vector, where its each element is 0. This BOF-vector has K elements in it, the K is the K we used for K-means algorithm, equivalently K is the number of visual words in our visual dictionary.

Then we again extract all SIFT-vectors from the image using the same set of parameters we used while building our dictionary. Let's say we have M SIFT-features extracted from the image. For each such SIFT vector, we determine which visual word that SIFT vector is closest to. This actually tells us, which visual word the current SIFT vector belongs to. If the SIFT-vector is closest to Nth visual word, we increment Nth element of the BOF-vector by 1.

Intuitively, what we call BOF-vector is a histogram, keeping track of the number of SIFT-feature from the image which we think belongs to a Visual Word, for each Visual Word. We also at the end normalize this BOF-vector by dividing every value by the total number of features in the image.

Doing these steps, what we have is an image, BOF-vector pairs for all images.

2.4. Image Query

Now that we know the BOF-vector for each image, which is actually a K-element feature vector which we hope describes our image sufficiently ”nicely”; for a given query image we calculate its BOF-vector, and compare query image's BOF-vector with all other image's BOF-vectors in database. To compare two BOF-vectors we simply use Euclidean distance. Close the two BOF-vectors, the more they should resemble each other.

So by this way, for a given query image, we know ”how much similar” each image in database to the query image. Then we can either return the top N similar images, or return images similar above a threshold.

3. Choose Of Parameters

There are 2 parts in our architecture which demands parameters. Those are the SIFT algorithm and the K-means algorithm.

For our implementation, for SIFT algorithm we use the OpenCV[2] library for Python programmin language, and for K-means we use the Sci-kit[1] library, again for Python.

3.1. K-means parameters

K-means implementation of Sci-kit admits 3 important parameters:

1) Batch Size: Number of examples from whole data to sample to use at each iteration. The bigger this value is, more time and memory is used but more accuracy is gained. We set this as bis as we could given our hardware and also for the algorithm to finish in reasonable time, which was 10000 for us.

2) Reassignment Ratio: The fraction of the maximum number of counts for a center to be reassigned. We again set this as big as we could and also for the algorithm to finish in reasonable time, which for our hardware was 0.1.

3) Cluster Count: This is the infamous K parameter. We first tried the values 32, 64, 128, 256 and 512. After seeing that best maP was achieved at 256, we further divided the ranges and settled on XXX as K. More is explained in Experiments section.

3.2. SIFT vs Dense SIFT

As explained, SIFT algorithm admits two parts. First a set of key-points are selected, than for each key-point a local descriptor is extracted. While regular SIFT uses DoG to select key-points, we can also densely select key-points at regular intervals. The latter approach is called Dense SIFT.

Since using Dense SIFT consistently gave us better maP results, we settled for Dense SIFT, and choose 10 as our ”regular interval”, in other words ”step size”.

The results, where Dense SIFT consistently does better, can be intuitively understood as follows:

Even tough we are doing ”content based” image retrieval, we do not work with semantic information from images. We compare their features, albeit invariant at many respects, but still based on image pixels; intensities and gradients. And while semantic information in images usually confined to some particular area in it, like a cat or a baseball stick; non semantic content can be safely assumed to be scattered all across the image rather equivalently. For example, a ”cat” image may have only a portion of pixels relevant to catness, but a ”snowy mountain” image will have ”snowy mountainess” on all the pixels consistently.

So given the nature of our data set, it fares much better to sample the image as densely as possible, and at regular intervals to not bias over particular regions. Had our data set been zoo animals for example, where all images has ”zoo'ness” scattered across an image and ”lion'ness” in only particular regions, regular SIFT may have fared better. I believe that would be not ”Content Based” but ”Semantic Based” image retrieval.

3.3. SIFT parameters

SIFT implementation of open-cv library admits 3 important parameters:

1) Contrast Threshold: The contrast threshold is used to filter out weak features in semi-uniform (low-contrast)

Parameter	Sift	Dense Sift
K	128	256
Contrast Threshold	0.04	-
Edge Threshold	10	-
Sigma	1.6	1.6
Step Size	-	10
maP	0.376	0.496

TABLE 1: Default Parameters

Parameter	Sift	Dense Sift
K	128	256
Contrast Threshold	0.01	-
Edge Threshold	13	-
Sigma	0.95	1.4
Step Size	-	15
maP	0.477	0.522

TABLE 2: Best Parameters

regions. The larger the threshold, the less features are produced by the detector. We set this to a value found in Experiments section, the value which gave best maP.

2) Edge Threshold: The threshold used to filter out edge-like features. Unlike Contrast Threshold, the bigger this value is the less features are filtered out. Intuitively this parameter set's how "corner like" we demand from a key-point to be included. We set this to a value found in Experiments section, the value which gave best maP.

3) Sigma: The sigma of the Gaussian applied to the input image at the first octave. Then the sigma is increased iteratively for further octaves. Refer to original SIFT paper for the explanation for octave. We set this to a value found in Experiments section, the value which gave best maP.

4. Experiments

Do decide on our final parameters, we did a number of experiments. For each experiment, we varied a single parameter, and fixed all others to default value. This gave us a good idea about how each parameter results to our overall maP. The default parameters and their maP scores are given in Table 1.

The first experiment we did on K, the cluster count. All other parameters were set to their default values as stated in their respective libraries documentations. The results are plotted in figure 1. We see that for sift, maP score maximizes around 150 and for dense Sift it maximizes around 250.So we took K to be 128 for Sift and 256 for Dense Sift in rest of the experiments. Also note that from now on, the maP score of any result may fluctuate which is caused by K-mean algorithm's in-determinism, so in this and all other plots, scores may vary ± 0.03

Next, we again varied sigma, all other variables as default. Results are plotted at figure 2. We see that for Sift, maP score maximizes around 1 and for Dense Sift sigma has very little affect on final maP score, the reason is that unlike Sift, Dense Sift does not take sigma in to account during key-point estimation but only in feature extraction. We took sigma to be 1.4 for Dense Sift. We should also note that our hardware with 16gb RAM gave Memory Error for sigma values 0.3, 0.4, 0.5, 0.6 and 0.7 with Sift.

At figure 3 and figure 4, we evaluate Contrast Threshold and Edge Threshold parameters, again taking all other parameters as default and K as 128 for SIFT and 256 for Dense SIFT. We again see that neither parameter has an effect on Dense Sift, since those two parameters are only used for key-point estimation. What is more surprising is that Edge Threshold randomly fluctuates Sift's maP result, and the fluctuations are in the range of K-mean's error. Since in our plot 13 gives the biggest maP and it is closest to default value, 14, we will take 13 as our Edge Threshold parameter for Sift. For Contrast Threshold we take it to be 0.01 for Sift. Even tough 0.003 actually gives the best result, taking it that low increases the estimated key-point count by an order of magnitude without significantly increasing the maP score.

Finally we plot the Step Size versus maP graphics at figure 5. Here the Step Size is the interval which we sample the images for Dense Sift. Result shows that maP maximizes around 15, and quickly drops off after. We should also note that our hardware with 16gb RAM gave Memory Error for step size values 3 and 4

So, to put it all together, in table 2 we give the maP scores for Sift and Dense Sift and the best parameters we choose. As the result show, using Sift algorithm with its best parameters gave us 0.47 maP score while Dense Sift gave us 0.52 maP score. So the result is increased significantly for Sift, but only marginally for Dense Sift. Tough that is because Dense Sift had less important parameters compared to Sift.

Finally at figure 6, we provide some successful and failure cases of image retrieval. Each row has a single query image, followed by ground truth images which then followed by result images. First two rows shows successful cases where returned top 3 images were exactly the ground truth images. Third row is a failure case, and in returned images first ground truth image had been returned as 110th image and the second is as 1147 image. Likewise in fourth row, the first ground truth image returned as 31th most similar image and the second is as 41th. This show that when our system fails, it completely misses the content and not be off by a margin.

5. Future Work

We may add that our research can be immediately extended in three following ways;

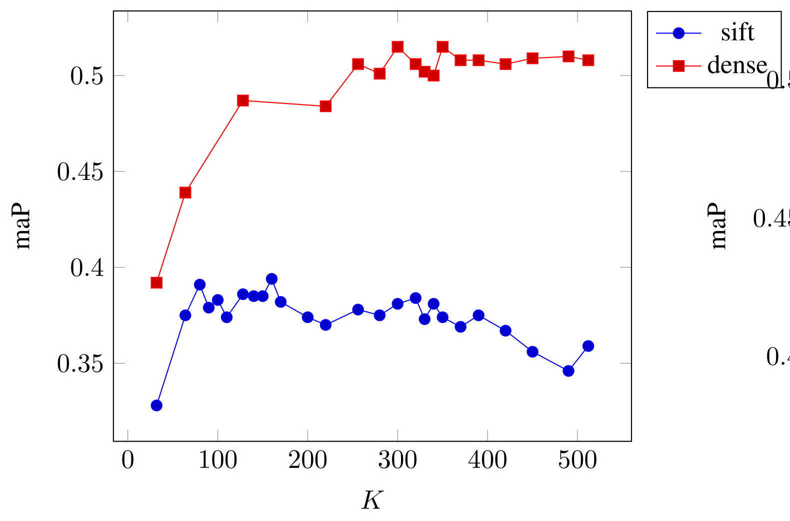

Figure 1: maP versus K

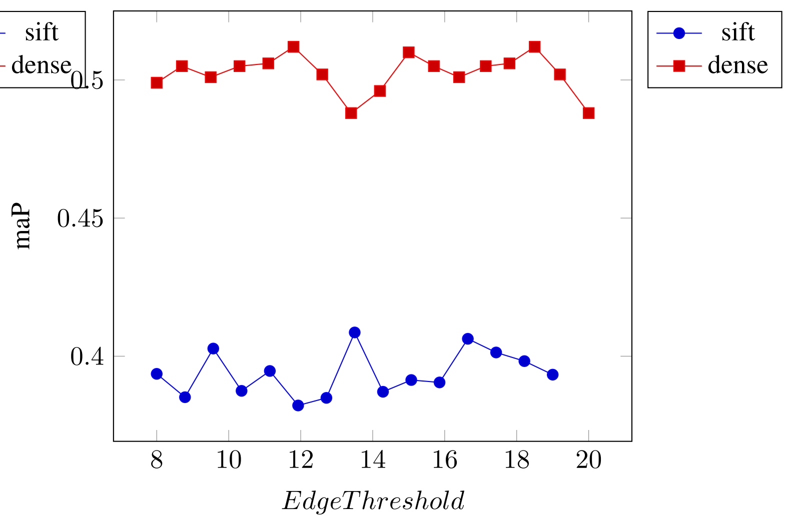

Figure 3: maP versus Edge Threshold

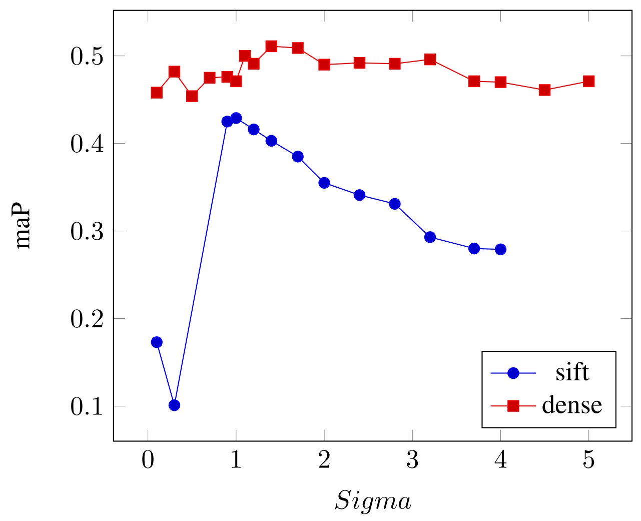

Figure 2: maP versus Sigma

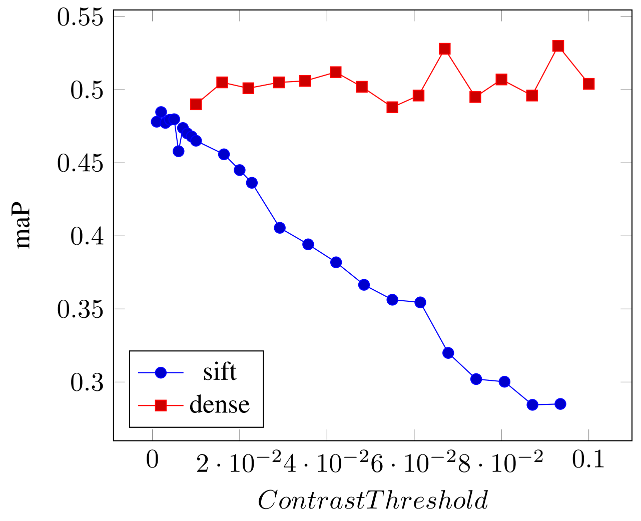

Figure 4: maP versus Contrast Threshold

1) For Dense Sift, our current implementation takes the size of key points as the step size itself. Instead, while keeping sampling the key points in regular intervals, for each key point it size can be determined by applying a laplacian filter as is done in regular Sift.

2) In this paper we searched for best parameters ”linearly”, and assumed that a parameter’s effect to final maP is independent of other parameters. This assumption may be false, so a more sophisticated parameter search may form a co-variance tensor for all parameters and choose the parameters accordingly.

3) Two main differences between Sift and Dense Sift are (a) the number of key point they produce and (b) the independence of the key points. The latter is because regular Sift tried to extract key point from ”possible interesting regions with respect to pixel intensities”. A research that compares Sift and Dense Sift maP’s while keeping the number of key points as close as possible may give better idea about their differences.

6. Conclusion

To conclude, in this report we introduced a Bag of Word solution for Image Retrieval problem which utilizes SIFT detector. Our experiments showed that estimating key points densely increases the results, for reasons we have discussed. We also presented some success and failure cases to visualize our results. On average our system runs poorly. One interesting aspect is that we have almost no partial success cases, so we speculate that our feature detector either successfully captures the image content, or it completely fails, and the results follows accordingly.

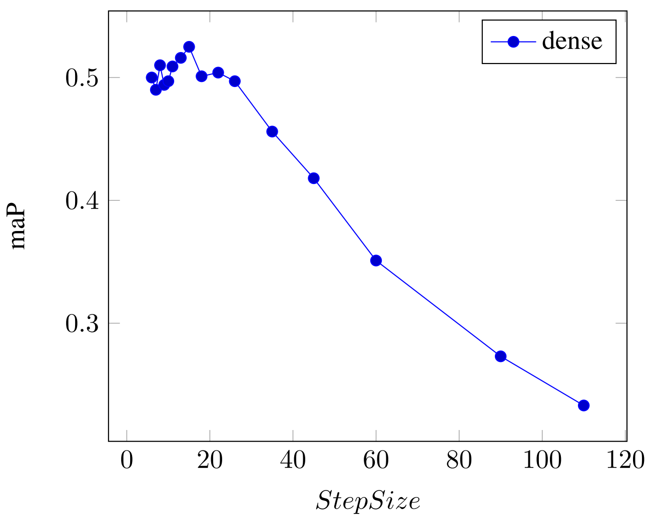

Figure 5: maP versus Step Size

References

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

[2] Bradski, G., & Kaehler, A. (2000). OpenCV. Dr. Dobbs journal of software tools, 3.

[3] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.