Dan Kristensen
Jens Egholm
Tobias Kenhof

Assignment 3: Content based image retrieval

Introduction

Given a database of images and a query image, content based image retrieval is a method for finding the images in the database, which resemble the query image. In the optimal case, such a measure can be ordered so the *most* similar images are listed first.
In this report, we conduct content based image retrieval by representing image features as visual words. This is done via the Bag of Words (BoW) principle, which often is a histogram representation based on independent features in an image. Thus, the BoW model principle represents an image as a document, “visual words” (image patches or fragments) in the image must be defined as well. To create our content based image retrieval system, we followed the pipeline below.

Data set

The data set in our assignment is the Caltech 101 consisting of 9146 images split into 101 different categories. Each category contains between 40 and 800 images and every image is about 300 x 200 pixels. For the purpose of this assignment, we split the Caltech 101 into a training set and test set. SInce the data set is ordered, the split was done by assigning every even image to the training set and odd to the test set. To reduce the runtime of our application we reduced each category to 40 images: 20 in the training set and 20 in the test set. The following categories were chosen for the data set: 'accordion', 'bass', 'brontosaurus', 'pyramid', 'lobster', 'sunflower', 'hedgehog', 'ferry'.

Feature extraction and representation

Interest points on an object in a given image, can be extracted to get a feature representation of the object. This representation is a description, that can be used for identifying similar objects between images. To perform a robust and reliable recognition, it is important that the the detection of the feature representations are invariant to scale, rotation and illumination.
Scale-invariant feature transform (SIFT) is an algorithm that can detect and describe features in images and is invariant to scale, rotation, illumination and translation. SIFT detects keypoints (interest points) by convolving a given image with Gaussian filters at different scales: scale-space extrema detection. The algorithm uses Difference of Gaussians (DoG) since Laplacian of a Gaussian (LoG) takes more computational power. The DoG derivatives are grouped by octave (double of σ-value). Once DoG derivatives are obtained, they are searched for local extrema over scale and space, meaning that each pixel in a given image is compared with its 8 neighbors at the same scale, with 9 pixels in the next scale and 9 pixels in the previous scale. If the pixel

value is a local extrema among all compared pixels, then it is best represented in that scale: a potential keypoint.
Since scale-space extrema detection produces many potential keypoints, they need to be refined to get more accurate results. This is done by eliminating any low-contrast keypoint and edge keypoints so only strong keypoints remain in consideration. Orientations is then assigned to every keypoint making them invariant to image scale and rotation.When a keypoint is created, the final step is to compute every keypoint as a descriptor vector. Around every keypoint, a 16x16 region is taken and for each sub-block a 8 bin orientation histogram is created. Each histograms therefore contains 4x 4 samples from a subregion of the original neighborhood region. The output is then a vector of all the values of the histograms: a distribution of the magnitude gradient over an image patch. Since there are 4 x 4 = 16 histograms with 8 orientations it gives a vector with 128 dimensions.
For this assignment we used, OpenCV which provides the function sift.detectAndCompute() that directly find keypoints and descriptors in a single step using the SIFT algorithm.

Codebook generation

Looking at the eigenvector of 128 dimensions for each descriptor (or word in an individual image), each such vector can be plotted in a 128 dimensional space. Mapping all the vectors from all the images in the training set onto the same virtual space, we have a 128 dimensional map of all the words in all the images.
As a baseline for the comparison, we assume that such vectors describe features which transcend images so that one feature in one image can be compared to a similar feature in another image. Similarity in this regard, simply means measuring the euclidean distance between the two eigenvectors. Exploiting this feature, we can employ k-means clustering on the 128 dimensional space to get our between-image 'words', where each word is represented as the eigenvector of descriptors, which are similar enough to form one word or cluster. The amount of words is given by the amount of clusters (*k*). By grouping descriptors around *k* geometric centers (centroid). Each cluster provides a descriptor, which is the arithmetic mean of all the descriptors in the clusters, thus forming an average of the descriptors in the cluster, i.e. the word.
K-means clustering is highly sensitive to the number of *k*. If *k* is too low, the word that is represented by each cluster will be too abstract and not useful in a comparison. This is because the cluster forces descriptors together which have little resemblance. Conversely, a *k* that is too high will be overfitted to the population and makes a poor predictor for data that have not been fitted to the classifier.
To explore which values for *k* is the most optimal, we choose to generated datasets for values of 10-100, 200, 300, 400, 500, 1000, 1500 and 2000. A *k* value of 90 proved to be most optimal in our case. Since the generation of the codebook took a while, we chose a fixed number of *k* values to test. In more ideal settings, it would be interesting to explore the optimal value of *k* by plotting the predictive value of the cluster algorithm against a test data set. Hypothetically, the most optimal value of *k* occurs when the algorithm no longer improves its predictions (and by time worsens) as *k*

increases. To calculate the clusters we used the k-means clustering classifier from the *sklearn* library.
Employing the k-means clustering algorithm, results in a list of cluster centers. Each cluster is given a label (id) so it can be identified. We define these labels and geometric centers (or words) as our codebook. The codebook represents the visual vocabulary that can be use as a means of comparison across images.

Indexing

For each image in our training set we can now iterate through its features to discover which cluster (word) is closest to the given descriptor. This will give us a list of labels for each image. Each label can be understood as a word contained in that particular image, giving us a BoW for every image. We stored the BoW in a table along with the name of the image, the category and whether it belongs to the training set (True) or test set (False).
In a sense images can now be seen as text documents containing words. The comparison between images can be made by simply looking at how many words are in common; the more words images have in common, the more similar the pictures. We perform this comparison by converting the BoW of each image to a histogram of words. The histogram counts the occurrence of each word in the image, ranging from 0 to *k* (the number of words (or clusters) in the codebook). This provides a vector of length *k*, where each dimension represents the frequency of one word.
By calculating the euclidean distance between two vectors, we now have a measurement of how different two histograms - and thereby how images - are from each other. This score is used to rank the results in the next section on image retrieval.

Retrieving

To test the accuracy of our content based image retrieval, we run two experiments. Both experiments are based on queries from the test data set, since the classifier only has been trained on the training set. This will give a more accurate prediction for search based on images which are not in the training set, and thus make the results more generalisable.
In the first experiment we query each image in the test set to the training set. This metric is interesting because it shows how good the codebook is for comparing descriptors outside the already fitted data. The second experiment runs the test data on the test data set. This result shows how well the algorithm can match predictors which are not fitted, but similar. The higher the number, the better the codebook is to encompass images which were not in the training set.
In the second experiment we expect the algorithm to retrieve the query image as the first result, since the histogram will be the same. This was the case for all query images on all iterations over k and we therefore chose to also include in the table, the ranking statistics for the second closest image in the test data.

Each query is based on an ordered list of the images , which our algorithm believes to be the most similar. For both experiments we report two metrics: the accuracy of the query, measured in how often the top-3 results are in the same category as the query image (in percent) and the mean reciprocal rank, measured in the reciprocal value of the index of the first image from the same category as the query image. These results are shown in table 1.

k-value	Experiment 1: Accuracy on training data within top-3	Experiment 1: Reciprocal mean on training data	Experiment 2: Accuracy on test data within top-3	Experiment 2: Reciprocal mean on test data
80	66%	0.22	100% / 60%	1.0 / 0.15
90	70%	0.22	100% / 65%	1.0 / 0.18
100	65%	0.22	100% / 65%	1.0 / 0.19
190	68%	0.20	100% / 63%	1.0 / 0.17
200	63%	0.19	100% / 66%	1.0 / 0.16
500	46%	0.09	100% / 48%	1.0 / 0.10
1000	44%	0.06	100% / 38%	1.0 / 0.08
1500	28%	0.05	100% / 27%	1.0 / 0.07
2000	27%	0.06	100% / 27%	1.0 / 0.06

Table 1. Metrics from the two experiments, reporting accuracy and reciprocal mean of image queries.

With a data set made from 8 image categories with 40 images from each, the statistics seems to indicate that the optimal *k*-value is 90. As noted earlier, all retrievals from the test set with the test queries gives us a rank of 100% and a reciprocal rank score of 1.0 in all *k*. More interestingly, when retrieving images from the training data the rank is 70%. This means that there's a 70% chance that the right image category will be among the top-3 results when retrieving images from the training set. The mean reciprocal rank is at 0.22 meaning that the right category is usually found within the first 4-5 retrieved images. For the test data, the rank is 65% and the reciprocal rank at 0.18 (right category found within first 5-6 images). These results are promising but not completely satisfactory. With more time, it would have been possible to use a larger data set, making a more precise classifier and thereby getting better results. Had we used a larger and more complex data set, we would have probably seen a higher optimal value for *k,* to account for a higher number of visual words found in the data set.

Conclusion

We showed how to extract feature representations that are invariant to scale, rotation, translation and illumination and how to generate a metric of comparison (BoW histograms) across images. Finally we reported on the results of applying these comparisons between a training and test set.

The results are promising and shows a decent level of accuracy, but are not entirely satisfactory. Given more time and training data, the algorithm would most likely give a higher rank.
To improve the results one could look into a different or supplementary approach to using BoW. For instance the current method completely ignores the spatial relationship among patches. Looking at two patches in an image, the SIFT descriptor captures the individual features of each patch, but does not take their position into consideration. If the relative location of the patches had any meaning for the quality of the image retrieval, then the SIFT description removes this semantic. Including information on the spatial location would probably heighten the accuracy, since the position of the descriptors will influence image similarity.
Furthermore, it would be interesting to explore an automated approach for locating the most efficient value of *k*. The differing accuracy and reciprocal mean values in Table 1 shows how the value indeed has a large impact on successful image retrieval. To reach the best possible solution we would have to systematically try different values at a much more granular level. Lastly, it could be interesting to look at different means of comparison. One alternative to the common words approach is to use the inverse document frequency (tf-idf). This method controls for the length of the document, so unproportionally long documents (images with many features) does not outweigh images with less features. However, we have not observed large variations in the number of features per image, so this method may prove to have a small impact.