Reverse Image Search

Introduction to Reverse Image Search

Search engines like Google, Bing and Yahoo successfully perform word/text based information search and retrieval. At a basic level the search process performed by a search engine can be understood like this: A ‘query’ word or text, is searched for in a huge data matrix of words/documents. Words/documents which match the query word very closely are then retrieved from the data matrix.

In this lab we will try to perform a similar query based search, the difference now is that the query now is in the form of an image, and the database consists of images. Images are much more complex to process than plain words, so building an algorithm to capture relevant features of the images and then retrieving images that match the query image is much more complex. In this lab, we shall explore the basic concepts used in machine learning for information retrieval such as neighborhood search and see how simple search methods perform for image search. Finally, we shall see how image based search can be performed using concepts of Clustering and Support Vector Machine (SVM) algorithms.

Overview

Suppose we have a database of images and we wish to find a match from this database to a new image that is not in the database. Instead of comparing the images pixel by pixel, we can create a set of relevant features for the images that will help us compare the images efficiently. We can organize the information about our images in the form of features and compare the features of the new image to the features of the images in the database, to finally retrieve the images that have the most similar features.

A. Feature extraction: SIFT

To form the feature vectors for the images, we will use the existing Scale Invariant Feature Transform (SIFT) algorithm. SIFT is an algorithm used in object recognition that extracts descriptors from images. These descriptors can then be used to detect and locate objects of interest in an image. For object recognition methods, a particular variant of SIFT called dense SIFT is used. For this reason, we will use dense SIFT in this lab.

Dense SIFT divides the image into a dense grid, and for each point in the grid it extracts a descriptor. A SIFT descriptor is a vector (of dimension 128x1 in this lab) containing the spatial histogram of three dimensional gradients surrounding the point. Each gradient is a vector pointing towards the direction of greatest change in pixel intensity. The entire SIFT descriptor is weighted with a Gaussian weighting function to give more importance to intensity gradients nearest to the point.
B. Forming the feature vectors

A simple way of forming a feature vector for each image would be to concatenate all its SIFT descriptors. The concatenation of the SIFT descriptors from different images can be compared to find which images have the most similar concatenations of SIFT descriptors. A major limiting constraint of this approach is that this would identify similarity based on the exact orientation of objects within comparable images. Similar images will not be images just with similar descriptors, but with similar descriptors in similar locations. We know that similarity among images is not a function of the orientation of objects within the image alone, so in order to perform better image retrieval, we need to employ a more complex approach.

Let’s recall the document word search example seen in class (example 1.1 in the textbook), where there were several documents in a database, and a query document, and the objective was to retrieve the documents that have similar content to the query document. To do this we would form a data matrix \( A \in \mathbb{R}^{m \times n} \), in which the rows would correspond to the \( m \) words in the dictionary, and each of the \( n \) columns would correspond to a document. Each cell \((i,j)\) in the data matrix would then correspond to how many times the word \( i \) appears in document \( j \).

\[
A = \begin{bmatrix}
    a_{11} & \cdots & a_{1n} \\
    \vdots & \ddots & \vdots \\
    a_{m1} & \cdots & a_{mn}
\end{bmatrix}_{\text{words}}
\]

Now let’s suppose we have a new document, in which each word \( i \) appears \( q_i \) times, with \( i = 1..m \):

\[
q = [q_1 \ q_2 \ \cdots \ q_m]^T
\]

The information retrieval problem will now consist in figuring out which column in the matrix \( A \) is the most similar to the vector \( q \).

For image retrieval, we can take this same approach, taking the descriptors retrieved by SIFT to be the words, and each cell in the matrix corresponding to be how many times the descriptor \( i \) appears in image \( j \).

\[
A = \begin{bmatrix}
    a_{11} & \cdots & a_{1n} \\
    \vdots & \ddots & \vdots \\
    a_{m1} & \cdots & a_{mn}
\end{bmatrix}_{\text{descriptors}}
\]

The query vector will now contain the number of times each descriptor appears in the query image.

\[
q = [q_1 \ q_2 \ \cdots \ q_m]^T
\]

Following the text document example, the image retrieval problem will now be to find the images that correspond to the columns of \( A \) that are most similar to \( q \).

C. Clustering

An interesting challenge that arises when taking this approach is the need to group all the possible descriptors into clusters. In the document word search analogy, this would be similar to considering all the similar words: walk, walking, walked, walks as only one representative word, let’s say ‘walk’.
Clustering can be implemented by any algorithm designed to group similar vectors together into distinct groups. K-means is one particular clustering algorithm, which will be used in this lab. K-means partitions \( n \) vectors into \( k \) chosen number of clusters. Each cluster is created by minimizing the cost \( Q \) of each partition \( \pi \).

\[
Q(\pi) = \sum_{j=1}^{k} \sum_{i \in \pi} \|a_i - m_j\|_2^2
\]

K-means is implemented with Lloyd’s algorithm to converge to the optimal solution. The algorithm starts by initializing partition centers \( m_1^{(1)}, m_2^{(1)}, \ldots, m_k^{(1)} \) cleverly. Then the algorithm repeats the next three steps until some tolerance is met. First, the algorithm assigns vectors to partitions.

\[
\pi_j^{(l)} = \left\{ i; \|a_i - m_j^{(l)}\|_2^2 \leq \|a_i - m_j^{(l')}_j\|_2^2, j' \neq j \right\}
\]

Second, the algorithm updates the cluster centers.

\[
m_j^{(l+1)} = \frac{1}{|\pi_j^{(l)}|} \sum_{i \in \pi_j^{(l)}} a_i
\]

Lastly, the algorithm checks the partition cost is below a predefined threshold.

\[
Q(\pi^{(l+1)}) - Q(\pi^{(l)}) < \text{tolerance}
\]

Once the tolerance is achieved, the algorithm has the most recently updated cluster centers. The vectors closest to each center are clustered together in one partition.

D. Nearest neighbor using distances

Once the data matrix is formed, we would like to see which images are most alike to the query image. One way of comparing similarity of two vectors is to calculate the distance between them. If the distance is small, the two vectors are similar.

There are many types of distance measurements, but depending on the database and the features, some may perform better than others. Some examples are:

L1 distance:

\[
\|a_j - q\|_1 = \sum_{i=1}^{m} |a_{ij} - q_i|
\]

L2 distance (Euclidean distance):

\[
\|a_j - q\|_2 = \left( \sum_{i=1}^{m} |a_{ij} - q_i|^2 \right)^{1/2}
\]

We can see that a practical difference between this two distances is that the L2 distance penalizes large differences between the vector components more than the L1 distance.

For the image retrieval purposes, we can think of an algorithm that calculates the distance between the features of the new image \( q \) and each one of the columns of the data matrix \( A \), and retrieves
the index of the column that has the smallest distance. However, this can be very computationally expensive, especially if the data matrix is very large. This is because each feature vector must be compared to each other feature vector.

E. Support Vector Machines

Another way of determining which images are alike is to use Support Vector Machines (SVMs). SVMs are a class of machine learning algorithms which are used to recognize patterns among data and classify them. Consider that we have a data set with data points which can be separated in two different groups. SVM algorithms are used to build a model which can classify new and existing points into these two groups. Let us look at a pictorial representation of this data set first.

We can see that the two groups of data can be separated visually by drawing a line between the two groups. SVMs help us draw this line in our data sets. The line dividing the two groups is called the decision boundary. The name is suggestive of its function which is that the decision boundary decides in which group a new data points should and places them in either of the two groups. The points close to the decision boundary are called the support vectors.

It is not necessary that the decision boundary be linear. The above example illustrates a classifier in a linear classification problem. Often data though separable, may not be linearly separable. As the dimensionality of the dataset increases linear classification is not always the best way to classify this data. In order to build a classifier which performs successful classification on higher dimensions of data we employ ‘kernels’. A kernel takes two vectors and applies a nonlinear function to these two vectors in and in the process increases the dimensions of the two vectors.

We shall make use of SVMs in our image retrieval problem. We shall learn a classifier on each image present in the database. In our approach we shall employ the kernel most commonly used in image retrieval algorithms. Our choice of kernel is the ‘Multilayer Perceptron’ kernel. We shall then use each of these classifiers to determine if the query image is similar to images in the database to retrieve the most similar images.
Warm Up

We shall now try to see if we can check images for similarity using the basic methods discussed in Overview section.

Download the lab zip file and extract all the contained files. Move the VLFEATROOT file folder, the ImageSearchWarmUp.m file, and the smallcelldatabase.mat file to your Matlab workspace.

Load smallcelldatabase.mat to view the query image and the database images. Use imagesc in MATLAB to display these images. To display the images in grayscale using the colormap function.

load smallcelldatabase.mat

figure(1)
imagesc(query);
colormap('gray');
axis('square');
title('Query Image');

....

Extract the SIFT descriptors for each image using the vl_dsift() function.

%% Dense SIFT
% The vl_dsift function returns the descriptors for each image and stores
% them in d. Each descriptor has length 128.
[~,d1] = vl_dsift(image1);
[~,d2] = vl_dsift(image2);
[~,d3] = vl_dsift(image3);
[~,d4] = vl_dsift(image4);
[~,dq] = vl_dsift(query);

Note that vl_dsift retrieves 19,881 descriptors for each image, each descriptor being a vector of length 128x1.

Finding closest match by comparing the concatenation of the descriptors

For each image, concatenate the descriptors to create a feature vector.

%% Concatenating the descriptors and finding closest match using distance
% Each vector f contains the concatenation of all descriptors in the image
[l,w] = size(dq);
f1 = zeros(l*w,1); f1(:) = d1;
f2 = zeros(l*w,1); f2(:) = d2;
f3 = zeros(l*w,1); f3(:) = d3;
f4 = zeros(l*w,1); f4(:) = d4;
fq = zeros(l*w,1); fq(:) = dq;

1. Use the norm function in MATLAB to calculate the $L_1$ distance between the features of the query image and the images in the data matrix. The $p$-norm/distance can be calculated by using the function norm((), p) in MATLAB. Which image is closest to the query image using the $L_1$ distance?

2. Determine the image closest to the query image by calculating the $L_2$ distances between the query vector and each image in the data matrix.
3. Rank your solutions in terms of ‘best match’ to ‘least match’ and display the images along with the query image.
4. Do you agree with the solutions? Does visual inspection give you different answers? If YES explain your reasons. If NO explain why you think so.

Document word search approach

Now let’s take the document word search approach. To do this, let’s treat the descriptors as words (remember each image has 19,881 descriptors/words, each of length 128). Using the 5 images in the database we can create a dictionary with descriptors (words), and create a data matrix that tells us how many times each descriptor appears in each image.

Since there are too many descriptors (each image has 19,881 and there are 4 images), we are first going to cluster similar descriptors into just one representative descriptor. In document word search, this is similar to considering all the similar words: walk, walking, walked, walks as only one representative word, let’s say ‘walk’. For this example, let’s suppose the dictionary will have 100 descriptors. To determine the representative descriptors, and how many times each one appears in each image, we can use kmeans++.

```matlab
% Visual words and finding closest match using distance
nd = 100; %We will create a dictionary of 100 descriptors
D = [d1 d2 d3 d4 dq];

% Cluster all the descriptors (which are stored in D) into 100 clusters.
[U,L]=vl_kmeans(single(D),nd);

This code will cluster all the descriptors into 100 clusters. U will contain the means and L the labels.

5. In the context of the image search problem we are discussing. What does each column of U represent? And, what does each one of the labels represent?

Now it is time to form the data matrix A. Recall that the rows of A will correspond to all the descriptors in the dictionary (100 descriptors) and the columns will correspond to the images. Each cell (i,j) will tell us how many times the descriptor i appears in image j. Take your time to understand what this part of the code is doing.

```matlab
% Form A matrix
N=size(d1,2);
A=zeros(nd,4); %initialize A to be an nd x 4 zero matrix (number of words x number of images)
for ii=1:nd
    A(ii,1)=sum(L(1:N)==ii);
    A(ii,2)=sum(L(N+1:2*N)==ii);
    A(ii,3)=sum(L(2*N+1:3*N)==ii);
    A(ii,4)=sum(L(3*N+1:4*N)==ii);
end
```

Similarly, each element of the query vector is the number of times each descriptor appears in the query image.

```matlab
% Form query vector
q=zeros(nd,1); %initialize q to be an nd x 1 zero vector (number of words x 1)
```
for ii=1:nd  
    q(ii)=sum(L(4*N+1:5*N)==ii);  
end  

6. Use the norm function in MATLAB to calculate the $L1$ distance between the query vector and each of the columns of the data matrix. The p-norm/distance can be calculated by using the function norm((), p) in MATLAB. Which image is closest to the query image using the $L1$ distance?

7. Determine the image closest to the query image by calculating the $L2$ distances between the query vector and each image in the data matrix.

8. Rank your solutions in terms of ‘best match’ to ‘least match’.

9. Do you agree with the solutions? Does visual inspection give you different answers? If YES explain your reasons. If NO explain why you think so.

Summarize your results from all the methods discussed. Which method do you think worked the best? Which image was the best match to the query image? Do you think these techniques will be sufficient to perform image based search and information retrieval?

**Lab**

We shall now implement the SVM algorithm using ‘svmtrain’ and ‘svmclassify’ in MATLAB on the database formed in the WarmUp. Our task is the same, i.e. retrieve the closest match to the query image from among the images in the database.

1. Learn a classifiers on the database using ‘svmtrain’.

   SVMStruct1 = svmtrain(A',[1 -1 -1 -1]);

   Use this example to build classifier for the first image in the database matrix. Build classifiers for each image in the database similarly.

2. Classify the query image using ‘svmclassify’ to pick the best match among the database images.

   svmclassify(SVMStruct1,q');

   Use this example to check if the query image is similar to the first image in the data matrix.

3. Which image/images are the closest match to your query image? Do you agree with the results?

4. Now implement a kernel to learn a nonlinear classifier on your database. The kernel to be used is ‘mlp’. Follow this example to do this.

   SVMStruct = svmtrain(A',[1 -1 -1 -1],’kernel_function’,’mlp');

5. Classify the query image again using ‘svmclassify’.

6. Which images are closest match to your query image? Are the same as the ones returned using the linear classifier? Display your results. Do you think the nonlinear classifier is an improvement over the linear classifier? Explain your observations.

7. Now change your query image. Experiment with the other images in the database.

8. Change the number of words used to build the database matrix to 10, 300, and 500 (the kmeans algorithm will take a while with 300 and 500 words). How does changing the number of words impact your results?

So far we have looked at image retrieval on a small database. Let us now see how well can be images be retrieved from a slightly larger data base. Load the largecelldatabase.mat file and the
ImageSearchLab.m file. The database now contains 20 database images and a query image. The query image corresponds to the Ebola virus.

9. Form your new database matrix by using the clustering techniques discussed previously. Use 100 descriptors in your dictionary.
10. Now try to find matches to the query image from the database using nonlinear ‘mlp’ kernel classification. What are your observations?
11. Repeat the SVM training and classification process by varying the number of words in your database. For nd=10, 100, and 300, observe and report your errors (number of erroneously matched images and number of images that should have been classified as similar but were not.)
12. Change the kernel to a Gaussian and a polynomial kernel with nd=100. Do you see any change in the performance of your classifier based on the kernel selected? If YES, which kernel do you think works best? Can you think of a reason why?

We can see that there are several images classified as similar to the query image (some that match and some that do not.) To determine the best match, let us run the image retrieval algorithm again, on the images classified as similar. Keep the number of words on 100.

13. Take the images classified as similar to the query image and create a new database with those images and the query image.
14. Repeat the clustering and the SVM training and classification process on this new database. Use the kernel that best performed before.
15. Do you notice the iterative convergence of the solution to only images that should be classified similar to the query image?

We see that iteratively searching the database improves our classification results. Can you think of other ways to improve the search algorithm? How would you improve the clustering of the descriptors, to keep only relevant descriptors that are useful to compare the images? How would you improve the classification process?