Computer Vision
Homework 1

Scene Recognition

CS 188.2
October 5, 2019

Overview

The goal of this assignment is to build a set of visual recognition systems that classify scenes in different categories. The dataset [2] you will work on contains 15 categories (office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, suburb), with 100 images per category. Every image has around 200×300 pixels. After development, your systems will be given **new** images, and will have to assign them a label.

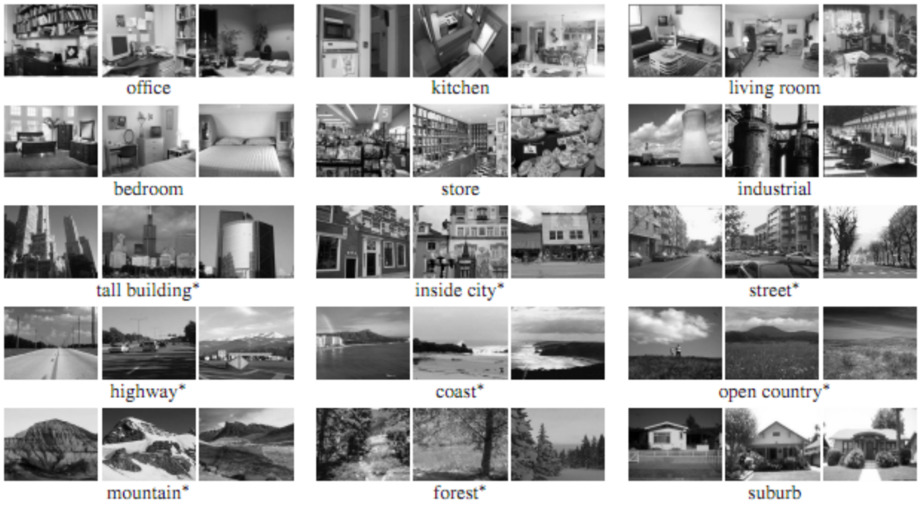

Figure 1: The dataset contains several hundred images belonging to 15 different categories.

The dataset has been divided between a *training* set and a *testing* set for you. During development of your recognition pipeline, you may only use *training* images and their labels, and will evaluate the performance of your system on new images from the *testing* set. You won't provide your system with the labels of testing images, but will compare the true labels with the ones produced by your pipeline. This allows us to define accuracy as:

$$Accuracy = \frac{Number\ \ of\ \ Correct\ \ Predictions}{Number\ \ of\ \ Predictions}$$

This idea is ubiquitous in machine learning.

Development will be done in Python. The list of packages needed will be specified in the next section.

Logistics

Deadline

The project is due Friday, October 18, at 11.59pm. **This is a hard deadline.** Your project will be submitted on CCLE, the contents of which are detailed in section 4. You may submit partial versions of your project early, and update this submission as you go. Avoiding submissions in the last hour is a good way to make sure that your project is handed-in on time. This project may be completed in pairs, in which case only one member of the group needs to submit.

Organization

We provided you with 3 things: this document, some starter code, and the dataset.

Starter code

The starter code organizes some of the work for you. It references the libraries that you need to complete this project, and will therefore throw an error if they are not installed. The required libraries are:

- OpenCV: A computer vision library
- Scikit-learn: A machine learning library (called sklearn for short)
- Numpy: Arrays in Python (the python language doesn't have a representation for arrays, the Numpy library introduces that)

Follow the links for installation information, which is made very simple using **pip** (found here). Pip is the recommended tool to install Python packages. You may also choose to manually install Python packages, but this will be less practical.

The starter code also details the names of the functions that you need to implement, as well as the types of arguments these functions should accept, and their return type. Do not change the signature of these functions, as we will call them during grading. You may have these functions call other user-defined sub-functions that we didn't specify, and organize them in other files.

Note that you are not expected to implement any algorithms from scratch in this homework, but rather to leverage the aforementioned libraries. This will involve some research through the API; this is an important skill. This is also a good opportunity to (re)view different machine learning algorithms.

Scene Recognition

Dataset

The dataset has been split in a training and testing set, contained in subdirectories. You may only use the training set to train your algorithms, and will use the testing set to report their accuracy. While a standard machine learning pipeline would involve random sampling of the test and train sets, you are not expected to do so here.

1 Baseline: Tiny image features

Your first task will be to build a very simple scene recognition system. It will not work very well, but will allow you to notice some of its flows, which you will fix in subsequent sections. Note that a first baseline could be to simply guess at random - given an image, assign it uniformly to a category. The expected accuracy is 7% (you would be correct on average once every 15 times, as there are 15 categories in the dataset). Let's do a little better.

The first idea will be to compare images (almost) directly. After all, a human being can tell that an image of a kitchen looks nothing like an image of a forest, so why not use pixel values directly? If we could define distance in image space, we could say something like: "That new image I_2 is *closest* to my training image I_1, which I know is an image of a kitchen. Therefore, I_2 is also an image of a kitchen." Let's develop a pipeline that follows this logic.

We could define a distance in image space as a difference in pixel intensities:

$$d(I_1, I_2) = \sqrt{\sum_{i,j \in \Omega} (I_1(i,j) - I_2(i,j))^2} \tag{1}$$

where $I_1(i,j)$ is the intensity value in image I_1 at pixel (i,j). Therefore, similar images will be 'closer' together (the idea of 'proximity' is a result of the definition of distance).

We calculate the distance of any new image (of the testing set, for which we pretend not to have a label) to all images in the training set (for which we have labels). We find the minimum distance, and assign the corresponding label.

Computing distances over images of $60,000$ values is inefficient, at best. Therefore, as a first step, we will resize all images in the training set to a smaller size. This will throw away a lot of information, but make the system usable. We will compare the resized version of those images to assign labels.

Tasks:

- Write a function imResize that resizes an image to a fixed scale. Normalize the output to be zero-mean in the range [-1, 1]. (*Hint:* OpenCV has a resize and a normalize function.)

- Write a function reportAccuracy that returns the accuracy, given predicted and true labels.

- Write a function KNN_Classifier that uses a nearest neighbor classifier to build the recognition system. (*Hint:* sklearn has an implementation.)

- Write a function tinyImages that:
  - Resizes images to 8x8, 16x16 and 32x32 scales
  - Runs a classifier using 1, 3, and 6 neighbors
  - Outputs 18 values, the accuracy and runtime for each configuration (*Hint:* Checkout the timeit package)

Your goal should be to exceed 15% accuracy in this section.

2 Building a Vocabulary of Visual Words

In section 1, you built a pipeline that performed better than guessing, but it's still far from an optimal system. The problem is that tiny image features aren't very good features. Note that:

1. Shifting the image by 1 pixel in any direction incurs a huge change in the distance measure, even though the scene represented in the image is largely the same.

2. Shifting the intensity of pixels up or down (illumination change) also causes a huge change in distance, even though the scene remains unchanged.

3. Images of the same scene from different viewpoints would yield very different images, that therefore wouldn't be 'close' to each other.

A good feature for scene recognition should be invariant to viewpoint, scale, and illumination changes. This is why SIFT features are so popular in computer vision, as well as SURF and ORB that were developed more recently. Let's modify the previous pipeline by using these features, and measure the performance gain. The next paragraphs give an overview of the pipeline.

SIFT [3] descriptors (or SURF [1] or ORB [4]) capture *local* information in an image. To represent the image as a whole, a single of these features is therefore not enough: we will represent an image with a collection of such features. If we extract SIFT (or SURF or ORB) features from all images in the training set, we will have a large collection that we can then group by similarity. Extracting one 'representative' feature for each group will allow us to build a *vocabulary* of features, or *visual words*. Doing so is done through a *clustering* algorithm, in this homework you will use K-means and hierarchical agglomerative clustering.

Once this vocabulary is built, we can compute a new image representation: a histogram of visual words. This representation is called a *Bag Of (visual) Words (BOW)*, and comes from the field of Natural Language Processing. For each test image we will densely sample many feature descriptors. Instead of storing these hundreds of descriptors, we simply count how many descriptors fall into each cluster in our visual word vocabulary. This is done by finding the nearest neighbor centroid for every feature. Thus, if we have a vocabulary of 50 visual words, and we detect 220 features in an image, our BOW representation will be a histogram of 50 dimensions where each bin counts how many times a descriptor was assigned to that cluster; and sums to 220. The histogram should be normalized so that image size does not dramatically change the bag of feature magnitude.

Tasks:

- Review the K-means algorithm
- Review Agglomerative Hierarchical Clustering
- Write a function buildDict that samples the specified features from the training images (SIFT, SURF or ORB), and outputs a vocabulary of the specified size, by clustering them using either K-means or hierarchical agglomerative clustering. Cluster centroids will be the words in your vocabulary. Use an Euclidean metric to compute distances. (*Hint:* OpenCV has implementations for SIFT/SURF/ORB feature detection. Sklean has implementations for Kmeans and Hierarchical Agglomerative Clustering)

3 Bag-Of-Words Recognition system

Images from the test set can now be classified using the vocabulary generated in section 2. The features you extract from test images have to be the same than the features you used to generate the vocabulary, otherwise the comparison would not make sense.

Tasks:

- Write a function computeBow that computes a BOW representation for an image, given the descriptor type.

- Use your KNN-Classifier to:
  - Classify images represented by a BOW, using 1, 3, and 6 neighbors.
  - Do so for different dictionary sizes and feature types. Consider dictionaries of size 20 and 50 using SIFT, SURF and ORB features
  - Return 36 values (accuracy and runtime for all configurations)

You should exceed 55% accuracy.

4 BoW + SVM

While the BOW model we used is powerful, the classifier remains very simple. To improve overall performance, you will train an SVM on BOW features. SVMs are (by default) linear, binary classifiers, and you will therefore train 15 1-vs-all SVMs (eg: 1 SVM for kitchen vs not kitchen, 1 for street vs not street, ...).

The feature space will be partitioned by a learned hyperplane, and test cases will be assigned a category dependent on their relative position to the hyperplane. The advantage SVMs have over a nearest neighbor classifier is that they can learn which visual words are more or less informative for categorization, and assign them a weight to reflect that fact. For example, maybe in our bag of SIFT representation 40 of the 50 visual words are uninformative. They simply don't help us make a decision about whether an image is a 'forest' or a 'bedroom'. While a KNN classifier will still be influenced by these frequent visual words, an SVM can learn that those dimensions of the feature vector are less relevant, and thus disregard them when making a decision.

Finally, you will then further improve results by using a nonlinear SVM with a Radial Basis Function kernel. This allows the feature space to be better warped before being split, allowing to better separate data. Nonlinear SVMs are still binary, you will still have to train them in 1-vs-all fashion.

- Write a function svm_classifier that:
  - Trains 15 binary, 1-vs-all SVMs (one for each category of images)
  - For each test case, run all 15 SVMs, and assign the label of the SVM performing with highest confidence
  - Returns the accuracy of the classification and its runtime
  - Implements the option of using nonlinear SVMs with an RBF kernel

Submission

Your submission will consist of a single tarball, "*UID*.tar.gz" where *UID* is the university ID of the submitter. It will be submitted on CCLE. Your tarball will consist of several files, listed below. Please respect the filenames and formatting **exactly**. You tarball should include:

- README: a .txt file
  - Line 1: Full name, UID, email address of first group member (comma separated)
  - Line 2: Full name, UID, email address of second group member (comma separated) if any, empty otherwise
  - Use the rest of the file to list the sources you used to complete this project
- code/: a directory containing all the .py files that you used for your project. Your main function should be called in a file called homework1.py, that imports all functions in other files. This spec specifies the functions you are expected to program; you may choose to include more in several different files.

Note: Your tarball should not include the data we provided.

Grading Criteria

The total score is out of 100, with the following breakdown:

- Build tiny image features for scene recognition - (**10 points**)
- Implement a Nearest Neighbor Classifier - (**10 points**)
- Build a vocabulary from a random set of training features - (**20 points**)
- Use Hierarchical Agglomerative Clustering to build the dictionary - (**10 points**)
- Build a BOW representation using SIFT features - (**10 points**)
- Build a BOW representation using SURF features - (**10 points**)
- Build a BOW representation using ORB features - (**10 points**)
- Train 1-vs-all linear SVMs on your bag of words model - (**15 points**)
- Train 1-vs-all nonlinear SVMs on your bag of words model - (**15 points**)
- Error in the submission format - (**-5 points/error**)

References

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[2] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2169–2178. IEEE, 2006.

[3] David G Lowe et al. Object recognition from local scale-invariant features. In *iccv*, volume 99, pages 1150–1157, 1999.

[4] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R Bradski. Orb: An efficient alternative to sift or surf. In *ICCV*, volume 11, page 2. Citeseer, 2011.