Chennai Mathematical Institute
Computer Vision
March 17, 2020

Due Date: Sunday, April 5, 2020 at 10:00PM

Assignment 4: Object Recognition with Bag of Words

Instructions:

1. Integrity and collaboration: Students are encouraged to work in groups, but each student must submit their own work. If you work as a group, include the names of your collaborators in your write-up. Code should NOT be shared or copied. Please DO NOT use external code unless permitted. Plagiarism is strongly prohibited and may lead to failure of this course.

2. Start early! Constructing the dictionary in Part 1, as well as converting all the images to visual words in Part 2, can easily take up to 30 mins to run (if implemented well!). If you start late, you will be pressed for time while debugging.

3. If you have any question, please post your question on Piazza. Other students may have encountered the same problem, and it helps to keep communication both efficient and uniform.

4. Write-up: Items to be included in the write-up are mentioned in each question and summarized in the Write-up section. Please note that we DO NOT accept handwritten scans for your write-up in this assignment. Please type your answers to theory questions and discussions for experiments electronically.

5. Late Policy: The assignment will not be accepted past 96 hours beyond the specified due date. You will lose 1 point for each hour you are late beyond the deadline for the first 6 hours. Beyond that, the delay will be counted in day units with a 10 point penalty for each day you are late.

Overview

The goal of this assignment is to introduce you to image classification. Specifically, we will examine the task of object recognition using the Bag of Features approach we studied in class and a selection of the classification methods we reviewed. Bag of words models are a popular technique for image classification inspired by models used in natural language processing. The model ignores or downplays spatial information in the image and classifies based on a histogram of the frequency of visual words. The visual word "vocabulary" is established by clustering a large corpus of local features. See Szeliski chapter 14.4.1 for more details on category recognition with quantized features. In addition, 14.3.2 discusses vocabulary creation.

For this assignment you will be implementing a basic bag of words model with some opportunities for extra credit. You will classify images into one of 8 given categories by training and testing on a subset of the CalTech-101 database. You will also evaluate the performance of your system on the test data provided. You will be required to submit a report, detailing your choice of hyperparameters and related experiments, and your interpretation of the results.

The categories we will be working with are:
airplanes, butterfly, grand-piano, helicopter, motorbike, side-of-car, sunflower, and umbrella.

Assignment Details:

This assignment has 3 major parts. Part 1 involves building a dictionary of visual words from the training data. Part 2 involves building the recognition system using our visual word dictionary and our training images. In Part 3, you will evaluate the recognition system using the test images.

Part 1: (20 points) Building the dictionary of visual words comprises the following:

a) (8 points) Use the corpus of training images and extract interest points and their corresponding SIFT descriptors. You implemented this already in your programming assignment 3. You can use your own code or a function available in a library to obtain the descriptors. You can limit the number of keypoints you extract from each image, in the dictionary building phase. Treat this as a parameter that you can vary to tune your system.

b) (8 points) In the second step, you will use K-means clustering on the assembled keypoint descriptors from the training images. The centroids of the clusters will serve as the visual words in your dictionary. Start with a small value of K (K =100) initially. Once you have all the components of your recognition system, gradually increase the value of K to see what dictionary size works best.

c) (4 points) Be sure to save your dictionary while you build the subsequent components of the system, to save time and computing resources.

This will be the implemented as 'build_vocabulary' function in Bag_of_Features_code.py.

Part 2: (50 points) Building your Image Recognition system involves the following steps:

a) (20 points) The first step will be to compute the frequency histogram of visual words for each training image. You will find the keypoints and their SIFT descriptors for each training image and build the histogram of visual words by determining the closest visual word that corresponds to each keypoint. Again, you can threshold the number of keypoints you consider for each image for this step, but the sampling should be denser than in Part 1.a) as the histogram serves as a quantized representation of the image. Histogram h, the vector representation of an image I, is a of size K, where K is the

dictionary size, the number of clusters K; $h(i)$ should be equal to the number of times visual word i occurs in the SIFT descriptor representation of image I.
Important Note: Since the images are of differing sizes, the total count in h will vary from image to image. To account for this, L_1 normalize the histogram before returning it from your function.

It may be helpful to store these frequency histograms for use in subsequent steps (bear in mind that the frequency histograms change when your dictionary changes).

This will be the implemented as function 'get_bag_of_sifts' in Bag_of_Features_code.py.

b) (15 points) Now that you have a way to get a vector representation for the training images, you are ready to build your image recognition system that classifies any new image presented into one of the 8 categories. You will build two classifiers.

The first classifier will make use of K-nearest neighbor classification. For nearest neighbor classification you need a function to compute the distance between two image feature vectors. Write a function to compute the distance between two images based on their frequency histograms. You can use either the Euclidean or X^2 (Chi-squared) distance measure. Use a few different values of K ranging from 1 to 20.

Implement 'nearest_neighbor_classify' in Bag_of_Features_code.py and adapt it to experiment with differing values of k (number of neighbors used).

c) (15 points) The second classifier you build will use support vector machine (SVM). You do not have to implement the support vector machine. You can avail of one of the many library functions available. However, linear classifiers are inherently binary, and we have an 8-way classification problem. To decide which of 8 categories a test case belongs to, you will train 8 binary, 1-vs-all SVMs. 1-vs-all means that each classifier will be trained to recognize 'airplane' vs 'non-airplane', 'butterfly' vs 'non-butterfly', etc. All 8 classifiers will be evaluated on each test case and the classifier which is most confidently positive "wins". E.g. if the 'airplane' classifier returns a score of -0.2 (where 0 is on the decision boundary), and the 'helicopter' classifier returns a score of -0.3, and all of the other classifiers are even more negative, the test case would be classified as an airplane even though none of the classifiers put the test case on the positive side of the decision boundary.

When learning an SVM, you have a free parameter λ which controls how strongly regularized the model is. Your accuracy will be very sensitive to λ, so be sure to test many values.

Implement 'svm_classify' in Bag_of_Features_code.py

Part 3: (10 points) Evaluating the image recognition systems you built in Part 2.

a) (5 points) Write a function that evaluates your K-nearest neighbor recognition system on the test images. Nearest neighbor classification assigns the test image the same class as the majority among the K "nearest" samples in your training set. "Nearest" is defined by your distance function. Report both the accuracy (#correct/#total), as well as the 8x8 confusion matrix C, where the entry $C(i, j)$ records the number of times an image of actual class i was classified as class j. The confusion matrix can help you identify which classes work better than others and quantify your results on a per-class basis. In a perfect classifier, you would only have entries in the diagonal of C (implying, for example, that an `airplane' always got correctly classified as an `airplane').

b) (5 points) Exercise your Linear-SVM based recognition system on the test data sets and report on the accuracy and compute the confusion matrix as before.

You can utilize the 'show_results' function provided in PA4_utils.py.

Part 4: (20 points) Results and Observations

Your summary write-up should be in a PDF document and include the following:

a) (2 points) Results of your interest point detector on one image from any 3 categories. You can reuse the function provided to you as part of Assignment 3 or use a library function.

b) (10 points) Detailed report on the design choices, including dictionary size, similarity and distance metrics used, hyperparameter selection and the experiments that support your choices.

c) (3 points) Include the output of K-Nearest Neighbor recognition system for 2 values of K (K=1, and the best performing among the other values of K you experimented with) and the two suggested distance metrics (4 confusion matrices and accuracies, in all). How do the distance metrics compare? How did you choose to resolve ties?

d) (2 points) Include the output of your linear-SVM based recognition system for 3 values of λ, including the one that yielded the best results in your experiments (3 confusion matrices and accuracies, in all).

e) (3 points) Which of the 2 recognition systems performed better? What might be the reason for this?

Extra Credit: (15 points) TF-IDF

With the bag-of-word model, image recognition is similar to classifying a document with words. In document classification, inverse document frequency (IDF) factor is incorporated which diminishes the weight of terms that occur very frequently in the document set. For example, because the term "the" is so common, this will tend to

incorrectly emphasize documents which happen to use the word "the" more frequently, without giving enough weight to the more meaningful terms. In this project, the histogram we computed only considers the term frequency (TF),i.e. the number of times a visual word occurs in the image (its representation as a set of keypoints encoded as SIFT descriptors).
Now we want to weight the visual word by its inverse document (image) frequency.
The IDF of a word is defined as:

$$IDF_w = log \frac{T}{|\{d : w \in d\}|}$$

Here, T is number of all training images, and $|\{d : w \in d\}|$ is the number of images d such that visual word w occurs in that image.

a) Compute a vector IDF of size 1xK containing IDF for all visual words, where K is the dictionary size.

b) Use TF * IDF in place of TF in computing the histograms of visual words.

c) With this new way of representing the content of an image, modify one of the recognition systems you built in Part 2 (either K-NN for the best performing value of K) or Linear-SVM for the best performing value of λ.

d) Compute and include the accuracy and confusion matrix in your written report. Is there a significant improvement in accuracy from using TF*IDF in place of TF?

What is provided:

Zip file of assignment directory which includes:

- Assignment Description (CV_Prog_Assignment4.pdf)
- code subdirectory containing:
    - PA4_utils.py (no need to edit this)
    - Bag_of_Features_code.py (shells provided for a number of functions you need to code; include any additional functions (K-NN, TF-IDF) with proper documentation here)
- data subdirectory containing:
    - 'train' folder which includes 65 training images for each category
    - 'test' folder with 10 test images for each category
- Jupyter notebook 'ProgAssignment4.ipynb' which provides a shell for exercising your code. It is not complete (K-NN for K > 1, and extra credit are not exercised here), so add cells as you need to fulfill all of the assignment requirements.

What to submit:

The folder you hand in must contain the following:

- code/ - directory containing all your code for this assignment;
    - Any additional functions implemented should be well-documented and included.

    - Your ProgAssignment4.ipynb notebook showing your results. Include additional cells as needed.

- A report in PDF format detailing your design choices, implementation and discussion of results.
- README - text file containing how to run your code, including extra credit implementation.

Hand in your project as a zip file on Moodle.

Credits:

This assignment draws from similar programming projects on scene recognition used James Hays at Georgia Institute of Technology and Ioannis Gkioulekas at Carnegie Mellon University.