Jim Little

Professor, University of British Columbia

Assignment 5: Scene Recognition with Bag of Words

(assignment is adopted and heavily inspired by a course project given out by James Hays and Sam Birch, with the code being re-written in Python by TAs)

Due: At the end of day**11:59pm**, Wednesday, March 31, 2021.

The purpose of this assignment is to introduce you to image recognition. Specifically, you will be asked to implement a classification approach for scene recognition starting with relatively simple methods. Specifically, you will be implementing bags of quantized local features with nearest neighbor classifiers and linear classifiers, learned by support vector machines.

The assignment

1. Bag of words models are a popular technique for image classification inspired by models used in natural language processing. The model ignores or downplays word arrangement (spatial information in the image) and classifies images 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 Forsyth chapter 16.1.3 for more details on visual word representation with quantized features. In addition, chapter 15.2.3 and 15.2.4 discusses classification techniques covered by this assignment. Note that this assignment will make use of the SIFT features that are described in the paper Distinctive Image Features from Scale-Invariant Keypoints, by David Lowe. SIFT descriptors will form the bases for a visual word ``vocabulary" and the resulting bag of words representation.

2. You are asked to classify scenes into one of 15 categories by training and testing on the 15 scene database (introduced in Lazebnik et al. 2006, although built on top of previously published datasets). Lazebnik et al. 2006 is a great paper to read, although we will be implementing the baseline method the paper discusses (equivalent to the zero level pyramid) and not the more sophisticated spatial pyramid (you are welcome to try the spatial pyramid for your own amuzement).

3. The zip file containing data, per-computed SIFT features and starter code is hw5.zip. Note the file is nearly 500MB and when unzipped will require 3.3GB of disk space. The unzipped directory, contains a skeleton Python program `main.py` (which you can run by typing

```
python main.py
```

at the command prompt). It also contains image files and their associated SIFT features that have been precomputed for you. The images and the SIFT descriptors are also broken out into the `train` and `test` sets. In adition to the `main.py` file, starter code also contains two helper files: `util.py` and `classifiers.py` which you will also need to modify. You can easily run the program on Colab by just copy pasting the

code in main.py into a notebook and uploading the remaining files. Alternatively, you can paste code from all starter files into the notebook if this proves easier for you.

4. `(10 points)`

   Before we can represent our training and testing images as bag of feature histograms, we first need to establish a vocabulary of visual words. We will form this vocabulary by sampling many local features from our training set and then clustering them with kmeans. To do so, you will need to fill in missing bits of `build_vocabulary` function in `util.py`. You WILL want to use KMeans functions available in sci-kit learn library. The number of kmeans clusters is the size of our vocabulary and the size of our features. For example, you might start by clustering many SIFT descriptors into k=50 clusters. This partitions the continuous, 128 dimensional SIFT feature space into 50 regions. For any new SIFT feature we observe, we can figure out which region it belongs to as long as we save the centroids of our original clusters. Those centroids are our visual word vocabulary. Note that for good performance you would likely want to sample many local features for clustering, which may result in KMeans being slow.

   Now we are ready to represent our training and testing images as histograms of visual words. We simply count how many SIFT descriptors fall into each cluster in our visual word vocabulary. This is done by finding the nearest neighbor kmeans centroid for every SIFT feature. Thus, if we have a vocabulary of 50 visual words, and we detect 220 SIFT features in an image, our bag of SIFT representation will be a histogram of 50 dimensions where each bin counts how many times a SIFT 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. To do this, you will need to fill in the details of `get_bags_of_sifts` function in the `util.py`; be sure to follow suggestions in the starter code.

   Before moving to implementing classifiers on this representation, it is useful to see the raw data. Plot average historam for every scene category. Average histograms can be obtained by simply averaging histograms for each training image. You should end up visualizing 15 average histograms which you should also submit as part of the writeup. Write a few sentences to describe how different are the histograms from different classes. Which classes you may believe to be hardest to separate (i.e., which you would be expect to be most confused) looking at these histograms.

5. `(5 points)`

   You should now measure how well your bag of SIFT representation works when paired with a nearest neighbor classifier. There are many design decisions and free parameters for the bag of SIFT representation so accuracy might vary from 30% to 40%. To implement this part follow the instructions in `classifiers.py`. Again, you will want to use sci-kit learn library instead of coding this from scratch yourself.

   To measure performance of nearest neighbor classifier report clssification accuracy and plot confusion matrix (which should be of size 15x15). You will be required to hand in both of these measures as part of your PDF writeup. Experiment with the size of k, how does this effect the performance of the classifier?

6. `(10 points)`

   The last task is to train 1-vs-all linear SVMS to operate in the bag of SIFT feature space. Linear classifiers are one of the simplest possible learning models. The feature space is partitioned by a learned hyperplane and test cases are categorized based on which side of that hyperplane they fall on. Despite this model being far less expressive than the nearest neighbor classifier, it will often perform better. 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'. Perhaps they represent smooth patches, gradients, or step edges which occur in all types of scenes. The prediction from a nearest neighbor classifier will still be heavily influenced by these frequent visual words, whereas a linear classifier can learn that those dimensions of the feature vector are less relevant and thus downweight them

when making a decision. There are numerous methods to learn linear classifiers but we will find linear decision boundaries with a support vector machine. You do not have to implement the support vector machine. However, linear classifiers are inherently binary and we have a 15-way classification problem. To decide which of 15 categories a test case belongs to, you will train 15 binary, 1-vs-all SVMs. 1-vs-all means that each classifier will be trained to recognize 'forest' vs 'non-forest', 'kitchen' vs 'non-kitchen', etc. All 15 classifiers will be evaluated on each test case and the classifier which is most confidently positive "wins". E.g. if the 'kitchen' classifier returns a score of -0.2 (where 0 is on the decision boundary), and the 'forest' classifier returns a score of -0.3, and all of the other classifiers are even more negative, the test case would be classified as a kitchen 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 'C' which controls how strongly regularized the model is. Your accuracy will be very sensitive to C, so be sure to test differnt values. See `classifiers.py` for more details.

Now you can evaluate the bag of SIFT representation paired with 1-vs-all linear SVMs. Accuracy should be from 40% to 50% depending on the parameters. To measure performance of the SVM classifier report clssification accuracy and plot confusion matrix (which should be of size 15x15) again. You will be required to hand in both of these measures as part of your PDF writeup. Experiment with the parameter 'C' and report your experience.

Deliverables

Hand in your functions (i.e., `*.py` files) or a Jupyter Notebook and PDF file showing results and answering questions. These must have sufficient comments for others to easily understand the code. Note you will have to hand in,

1. Average BoW histogram plots for 15 classes
2. Table showing accuracy of nearest neighbor and SVM classifiers. If you run multople nearest neighbor and/or SVM clssifiers, make sure to note which parameters a particular accuracy corresponds to.
3. Confusion matrix showing performance of nearest neighbor classifer
4. Confusion matrix showing performance of SVM classifer
5. Any relevant discussion of experiments

as part of the PDF. Both Python files and PDF should be submitted through Canvas