Writeup for CMSC 733
Problem set 5

Jason Filippou
jasonfil@cs.umd.edu

December 9, 2014

Contents

1 Overview - Important notes

In this write-up, we strive to analyze all the different factors that affect the performance of a multi-class classification algorithm and report our results across a variety of experiments. Over the past few years the main focus of Pattern Recognition approaces in Computer Vision has been - perhaps rightfully - on the general domain of finding better feature representations, as opposed to optimizing the learning algorithms. In this project, we follow the latter approach. We examine two very basic - and by now anything but state-of-the-art - feature representations, SIFT and FilterBank, and examine various different parameters of both the dictionary learning and the SVM training phase to see which one provides for better performance. We believe that the conclusions made by this writeup are important for Vision researchers who wish to attain an "edge" in the performance of their Pattern Recognition submodules.

Some important notes about the code:

1. We have transformed the top-level script `ps5` from a MATLAB script to a MATLAB function, since this allowed us to run parallel executions of our code on a

UMD computer cluster to which we had access.[1] Running the function without any arguments (like a script) uses the initial arguments provided by the instructors. More information can be attained by looking at the comments in `ps5.m`.

2. The `getVocab` method runs `vl_kmeans` with a K-means++ initialization.

3. With `LIBLINEAR`, cross-validation is hard. The `train` method only returns an accuracy metric on the training data, neglecting to provide the user with the MATLAB struct it would've otherwise provided her with if cross validation had not been selected. For this reason, we do not do parameter selection for the underlying linear SVM classifier and leave the parameter C to the default value of 1.

2 Correctness of filters

As requested by the project description, we make sure our feature extraction works correctly for both methods. The MATLAB script `evalFeatures` takes care of this. Figure 1 visualizes the SIFT feature extraction for the famous built-in MATLAB image "peppers" whereas figure 2 visualizes the 17 different images produced by applying the 17 different FilterBank filters on the same image.

Figure 1: SIFT descriptors visualized on the built-in "peppers" image.

[1] We used an allocation we had on the Deepthought 1 computer cluster (`http://www.glue.umd.edu/hpcc/dt.html`) to run MATLAB jobs in parallel.

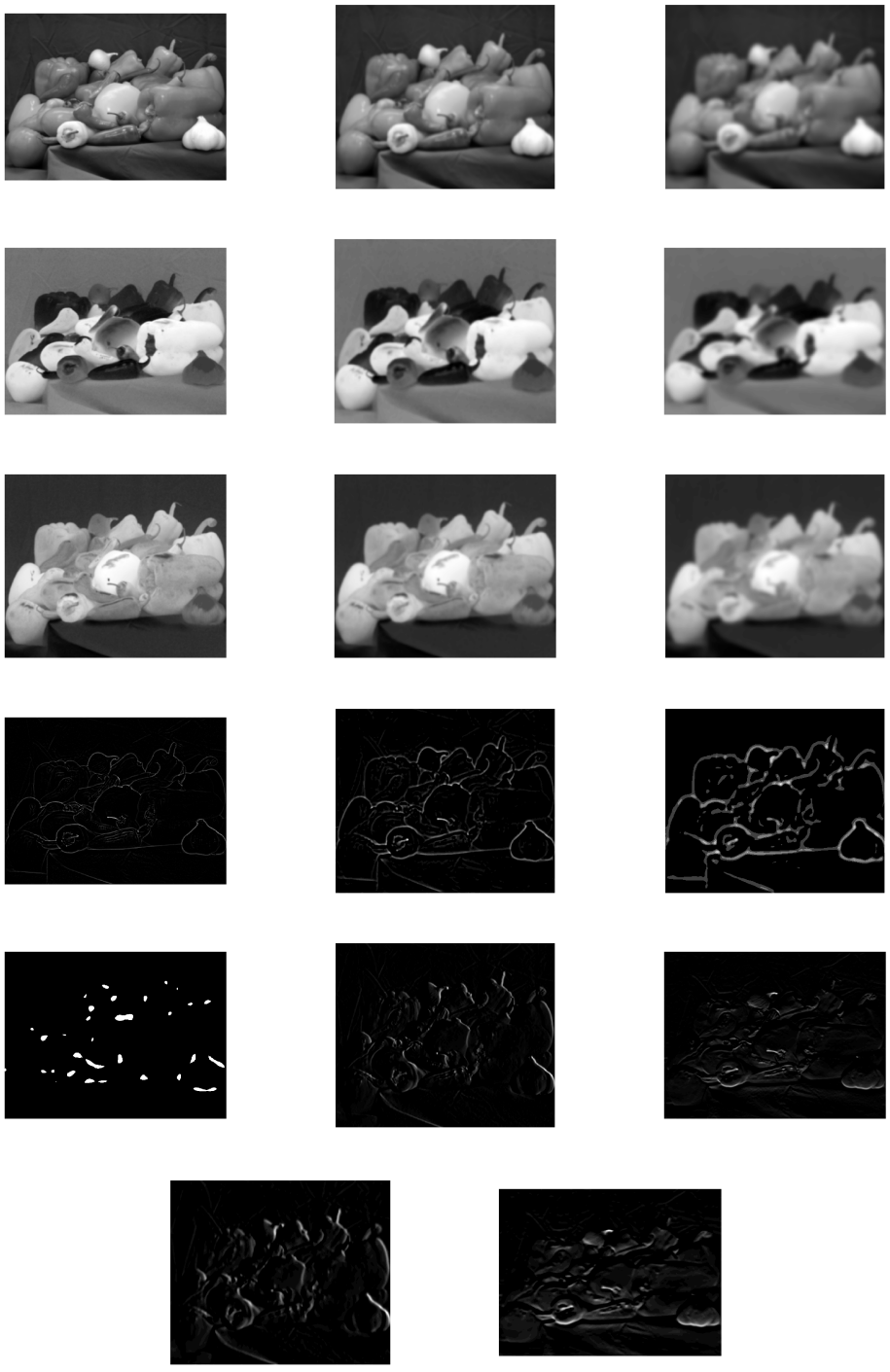


Figure 2: Applying the filterBank filters on the built-in "peppers" image. Numbering of filters begins from top-left and proceeds in row-major order. Consult `filterBank.m` for details.

3 Results with initial set-up and minor modifications

3.1 SIFT features

By "initial setup", we refer to the set-up for `ps5` provided to us by the course instructors. This set-up concerns itself with a subset of the Caltech data featuring the class set {`pizza, platypus, pyramind, revolver, rhino`}, a training ratio and dictionary building ratio of 0.5, 600 visual words generated for the dictionary, SIFT feature extraction and nearest-neighbor encoding. Figure 3 shows the confusion matrices attained for this set-up.

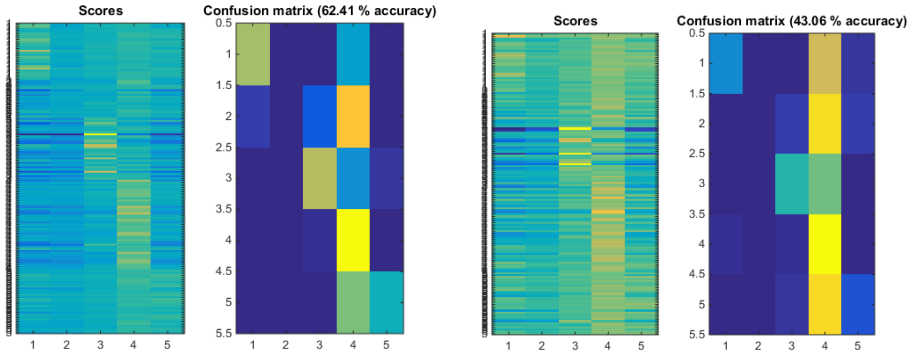


Figure 3: Training and testing confusion matrices for the default problem set configuration.

With the aforementioned initial setup we had an overall training set accuracy of 62.41%, whereas in the testing set we had 43.06% accuracy. The exact per-class accuracies are shown in table 1 Note that, ideally, since the train and test sets are extracted randomly with every run of the code, we should have presented the same experiment a number of times and taken the mean of the accuracies for both data splits, yet even those results in isolation provide for a good feel of what we can get with the default set-up.

We are already in position to present our first important optimization. We note that `LIBLINEAR` performs One-Versus-All (OVA) classification. This means that, for k classes, it trains k binary SVM classifiers. Some (or all) of those k binary classification problems might be imbalanced, because the positive data portions can be quite smaller than the negative data portions, or vice versa. By weighting the positive data portions by the number of negatives over the number of positives, we note a substantial improvement, shown in figure 4 and table 1. The reader may consult the function `trainSVM` to have an idea of how this is attained.

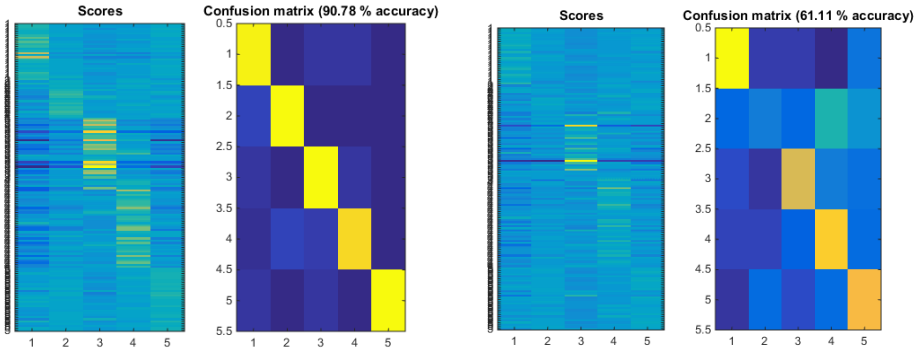


Figure 4: Training and testing confusion matrices for the default problem set set-up with weighting of classes.

Furthermore, the course instructors mention in the write-up that when local encoding is used instead of nearest neighbor encoding, certain classes are expected to show better classification performance, in particular the class `rhino`. Table 1 proves this claim, yet as can be seen in figure 5, local encoding does not necessarily lead to **overall** better classification performance for this initial set-up.

Class Name	**No weights, Nearest Neighbor**	**Weights, Nearest Neighbor**	**No weights, Local Encoding**	**Weights, Local Encoding**
pizza	25.93%	81.48%	11.11%	29.63%
platypus	0%	29.41%	17.65%	35.29%
pyramid	48.83%	75.86%	65.52%	68.97%
revolver	95.12%	75.61%	31.70%	31.71%
rhino	10%	66.67%	**33.33%**	23.33%
Overall	43.06%	**69.44%**	33.33%	**37.5%**

Table 1: Per-class and overall accuracies for the default set-up of the problem set with minor modifications. Note the improvement of results for class `rhino` when local encoding is used, as well as the overall improvement of the classification accuracy when we weight the classification problem (all those points of emphasis are **boldfaced**). The "Overall" accuracy for the 3^{rd} column just happens to be the same as for the class `rhino` (33.33%), i.e this is not a typo.

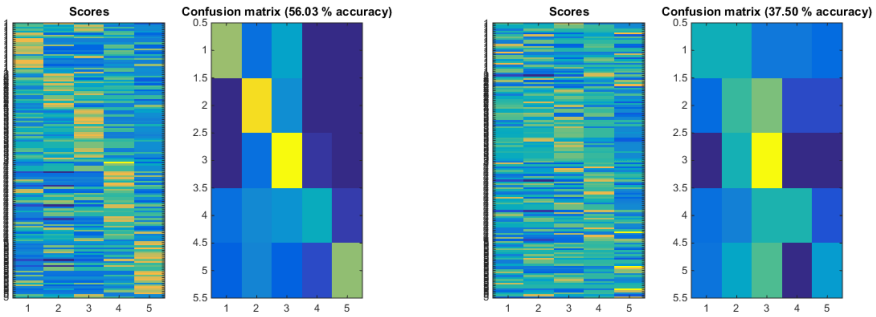


Figure 5: Training and testing confusion matrices for the default problem set configuration with weighting of classes and local instead of nearest neighbor encoding. Despite the fact that some classes can do better than others individually, the overall accuracy for this default set-up has dropped by approx. 30% on the test set.

3.2 FilterBank features

The filterBank features present a computational challenge at the dictionary learning stage. Specifically, since every $h \times w$ image ends up being represented by a vector of length $17 \times w \times h$, the computation of the squared two-norm between examples can become very costly and memory-intensive, particularly when `doubles` are in play. If we feed the original data into the function `vl_kmeans`, MATLAB eventually runs out of memory. In order to deal with this problem, we denote that, when compared to the number of 17-dimensional features involved in the learning of the clusters (which, for k images of mean size $m \times n$, is $k \times m \times n$), the number of visual words to be learnt is small. For this reason, we make sure that we feed K-means a maximum of 10^4 examples. More information is available in the comments of MATLAB function `getVocab`.

The confusion matrices of the original setup with filterBank features can be seen in figure 6. Note that since weighting positives has been deemed beneficial by previous results (see table 1 for a refresher), we perform this optimization for all further experiments, including the one discussed here. Comparing figures 6 and 4 we can see that FilterBank performs better when nearest neighbor encoding is used, while comparing figures 6 and 5 shows that it hurts a little in the case of local encoding.

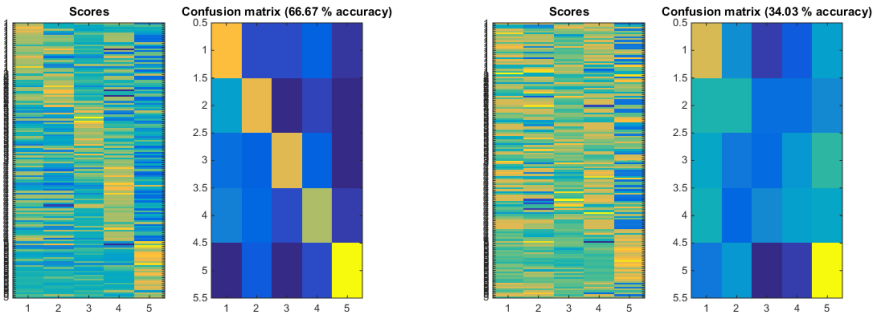
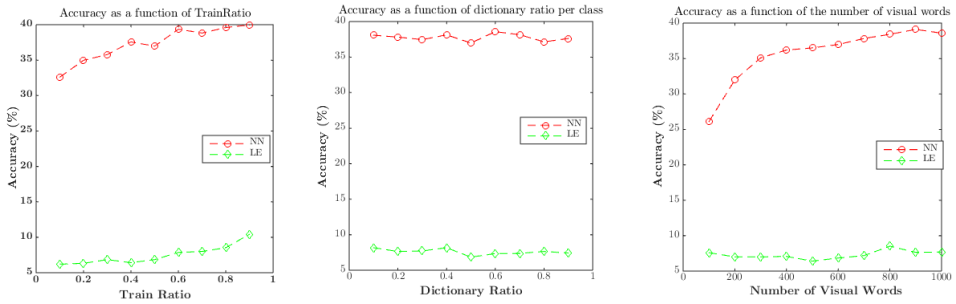


Figure 6: Confusion matrices for initial problem set configuration modified for extraction of FilterBank instead of SIFT features. Left: Testing accuracy when nearest neighbor encoding is used. Right: Testing accuracy when local encoding is used.

4 Results on entirety of Caltech dataset

In this section, we analyze performance of BoVW classification across all 101 classes of Caltech. We extract SIFT features and vary the encoding type, as well as the various parameters (`dictRatioPerClass`, `numWords`, `TrainRatio`).The reason for not considering FilterBank is purely computational; the feature extraction method simply does not scale on the entire dataset, even when we used our aforementioned DeepThought allocation.



(a) (b) (c)

Figure 7: The effect of (a) `TrainRatio`, (b) `DictRatioPerClass` and (c) `numWords` on classification accuracy over the entire Caltech 101 dataset. To produce every plot, the other parameters were kept to their default values, e.g for (a) we kept `dictRatioPerClass` equal to 0.5 and `numWords` to 600. Best viewed in color.

Figure 7 shows the effect of tuning various parameters of the dataset. There are a couple of conclusions to be made after observing the figures:

- Nearest neighbor encoding appears to globally outperform local encoding, by a large margin of ≈30%.

- Unsurprisingly, the learning algorithm tends to benefit from more training data over testing data. Care must be exerted, however, to avoid overfitting.
- It is not clear whether using a relatively large or small dictionary ratio parameter helps with classification accuracy.
- A larger number of visual words to build our vocabulary with leads to a more representative vocabulary and henceforth to increased classification accuracy.

All of those results are the product of jobs submitted to the DeepThought cluster. The full output of these jobs. in the form of .out files created by the SLURM scheduler, are available in the directory `sift_results`, which we include in our submission.