Indoor Scene Recognition

CS229 Autumn 2011 Final Project Writeup

Lu Li (lululi@stanford.edu), Siripat Sumanaphan (siripat@stanford.edu)

December 16, 2011

1 Introduction

While scene recognition problem is not new, it is still a challenging, open-ended problem on which a lot of further work can be done to improve feature selection and applicable learning algorithms. Several computer vision research groups at MIT have worked on scene recognition, with many classes of scenery images ranging from outdoor landscapes to indoor rooms or areas [6, 7]. While the accuracy rate is high for some outdoor landscapes [6], classifying the indoor specific areas still posts a difficult task due to similar image features across categories and huge variation within category.

Our primary goal was to be able to identify the bathroom images given a set of images of all areas in a home. We first selected bathroom simply because of its unique visual features. Later on we extended our work to also include binary classification of other classes as well as multi-class classification. Our dataset was downloaded from [5], which Ariadna Quattoni et al. also used on for their work on indoor scene recognition (with a larger set of scene categories) [6]. For our work, all algorithms were run on SIFT feature space. We chose to work with SIFT feature as it was reported by Jianxiong Xiao et al. to be one of the few top choices yielding high recognition accuracy [7]. In addition, most images of indoor home areas share a lot of common edge-, curve- and corner-related characteristics, which SIFT features can well capture.

2 Methods

We derived the outline for our methods from the paper *Linear spatial pyramid matching using sparse coding for image classification* by Jianchao Yang et al [9]. The general flow of algorithms is shown in figure 1. We will explain our working steps in the subsequent sections.

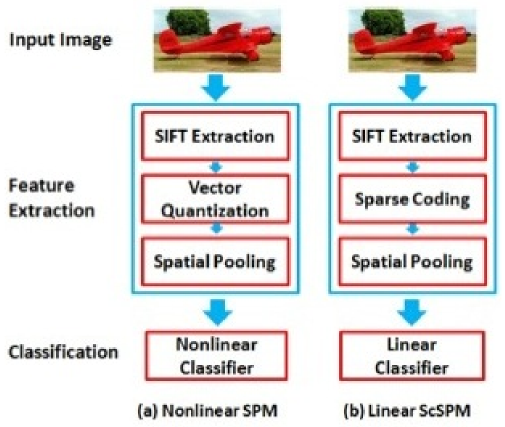


Figure 1: Algorithm flow (figure from [9])

2.1 SIFT feature extraction

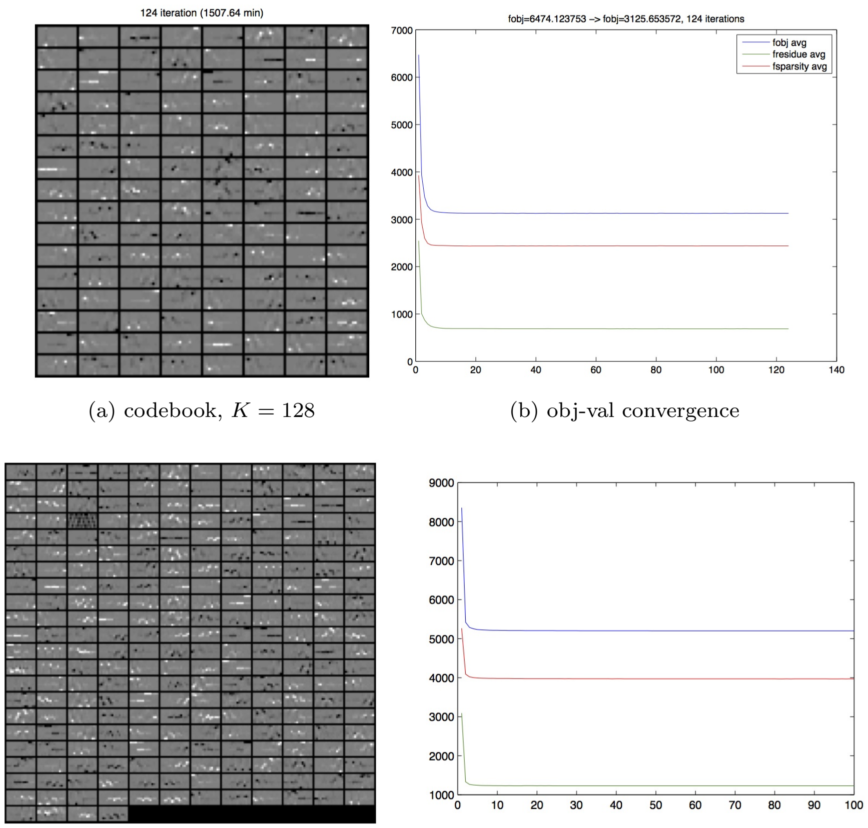
We used an online open C++ SIFT extractor library written by Zhenhui Xu [8]. The library was written based on the work done on SIFT features by David Lowe [4]. The extractor converts an image into gray scale, and doubles the size, when necessary, (for effective local extrema extraction) before extracting the SIFT vectors. For each image, the extractor output a set of keypoints, $X = \{x_j^{(i)} \in \mathbb{R}^N, N = 128, j \in \{1, ..., n^{(i)}\}\}$, where $n^{(i)}$ represents the number of keypoints for image i. It is often the case that $n^{(i)} \neq n^{(k)}$ for $i \neq k$, even though the original image sizes might be the same. From experiments, we found that $n^{(i)}$ ranges from approximately 200 to 6000.

2.2 Vector Quantization using K-means

To begin our classification, we first would like to learn for a *codebook* or a *dictionary*, the basis with which to quantize our SIFT feature vectors. Our simple starting step was to learn for this basis using K-means.

We selected the number of centroids, K, to be 128, below the minimum number of keypoints per image we have ever encountered. The training for the codebook $V \in R^{N \times K}$ was done on 86,229 SIFT feature vectors (or keypoints) taken from 70 random images of seven different areas in a home, i.e., 10 images per category. These areas include bathroom, bedroom, dining room, garage, kitchen, living room, and children room. The codebook then consists of the K centroids we get from K-means.

2.3 Vector Quantization using Sparse Coding



(a) codebook, $K = 128$

(b) obj-val convergence

(c) codebook, $K = 256$

(d) obj-val convergence

Figure 2: Results of sparse coding: codebooks and convergence of objective values

Sparse coding is our alternative to the simple K-means algorithm in finding the codebook. It was reported to be an effective algorithm for this application because image patches are sparse in nature [9]. Here, for a set of random SIFT features $S = \{x^{(i)}, i = 1, ..., M\}$, we solve for a codebook V with K basis vectors by solving [9]

$$\min_{U,V} \sum_{i=1}^{M} \|x^{(i)} - Vu^{(i)}\|_2^2 + \lambda \|u^{(i)}\|_1 \tag{1}$$

$$\text{subject to } \|v_k\|_2 \leq 1, k = 1, 2, ..., K$$

where $u^{(i)}$ is column i of the matrix $U \in \mathbb{R}^{K \times M}$, and v_k is column k of the matrix $V \in \mathbb{R}^{N \times K}$.

To solve for codebook using sparse coding, we used Matlab sparse coding software package provided by Honglak Lee et al., which can be found at [2], the work done based on the paper *Efficient sparse coding algorithms* by the same authors [3]. Due to frequent warnings of internal maximum number of iterations reached, either by Matlab fmincon or |1|s_featuresign.m in the software package, we experimented on several different numbers of basis vectors, tolerances, and λ values, and performed some evaluations on the resulting codebooks (see section 3.1). We can see from figure 2 that the objective values converged only after about 10 iterations, but the values were still large, although we have no cost function from K-means to compare with. By observation, we found that the final objective value was larger for a codebook with larger number of basis vectors.

2.4 Histogram as Feature for K-means-based Codebook

After training for the centroids, we simply used the histogram

$$z^{(i)} = \frac{1}{n^{(i)}} \sum_{j=1}^{n^{(i)}} u_j^{(i)}$$

as our feature for image i. Here $u_j^{(i)} = e_{k_j^{(i)}} \in \mathbb{R}^N$, where e_k is the k-th basis vector of the form $[0, ..., 0, 1, 0, ..., 0]$ with 1 occuring at the k-th index; $k_j^{(i)}$ represents the cluster $x_j^{(i)}$ belongs to.

The training set $\{(z^{(i)}, y^{(i)}), i \in \{1, ..., m\}\}$, m the number of training images, was fed into LIBLINEAR [1]. Here, $y^{(i)} \in \{1, -1\}$ for binary classification, and $y^{(i)} \in \{1, 2, 3, ..., n\}$ for multi-class classification, where n is the number of classes.

2.5 Maximum of Absolute Weights as Feature for SC-based Codebook

For a codebook V derived from sparse coding, we solved for the weight vector $u_j^{(i)}$ associated with a keypoint $x_j^{(i)}$ by solving equation (1), but minimizing only with respect to U instead of U and V. In this step, we again utilized |1|s_featuresign.m from Honglak Lee's matlab sparse coding software package [2].

After we got $u_j^{(i)}$, we calculated the feature $z^{(i)}$, associated with image i, from

$$z_k^{(i)} = \max\{|u^{(i)}|_{1,k}, |u^{(i)}|_{2,k}, ..., |u^{(i)}|_{n^{(i)},k}\}, k = 1, ..., N.$$

The $z^{(i)}$'s were normalized, and fed into LIBLINEAR to train for an SVM model, the same procedure as described in section 2.4.

2.6 Spatial Pyramid Matching

In addition to classification based only on the histogram (or max-pooling) of quantized SIFT features extracted from the entire image, we also extended our feature vectors to capture more spatial information of an image by concatenating with location-specific histograms.

In our case, we partitioned the image into $2^l \times 2^l$ sections using $l = 0$ and 1. For a partition, we extracted SIFT features and quantized them to get a histogram (or weights) specific to that partition and scale. All histograms were concatenated together to form a higher-dimensional feature representation of the image. These longer features vectors were used to train a linear SVM in the same manner as mentioned before. We expected the trained model to be more effective for classification of classes with large enough dataset to match the dimension of the feature vectors (now 128×5), e.g., living room, kitchen, and bedroom, each having more than 500 images in our dataset.

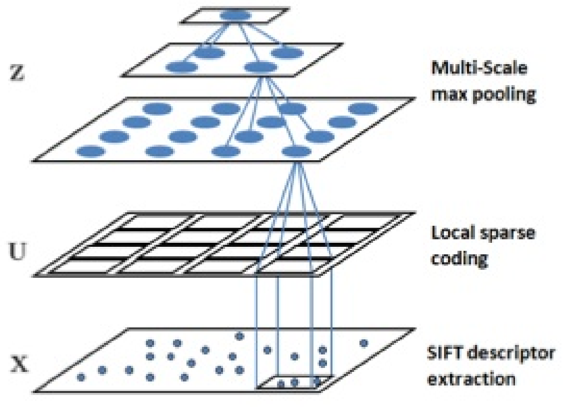


Figure 3: Spatial Pyramid Matching (figure from [9])

3 Results

3.1 Sparse Coding: Codebook Evaluation

The evaluation was done by binary classification of bathroom images vs all other classes. The features used were from $l = 0$ only ($z^{(i)} \in \mathbb{R}^{128}$). The %-train and %-test are based on the number of bathroom images available in the dataset (196). Images from other six classes were randomly selected so that the numbers of positive and negative samples are equal in both the train and test sets.

Table 1: Sparse Coding: Codebook Evaluation

SC parameters	40%-train, 60%-test	50%-train, 50%-test	60%-train, 40%-test	70%-train, 30%-test	77%-train, 23%-test
① $K = 128, \lambda = 5$, tol = 0.005, S_1	70.79±2.36%	71.07±3.57%	70.83±3.6%	70.67±3.96%	73.10±4.32%
② $K = 128, \lambda = 5$, tol = 0.01, S_2	67.02±3.27%	68.59 ±3.15%	70.90±3.41%	69.08±4.67%	68.45±4.87%
③ $K = 256, \lambda = 10$, tol = 0.5, S_2	72.37±2.64%	72.27±3.13%	71.99±2.53%	72.83±3.41%	73.81±4.59%

Here we show the mean accuracy ± one standard deviation. S is the set of SIFT features used. S_1 includes 86,229 SIFT features extracted from the total of 70 images, 10 randomly selected images per class. S_2 includes 70,000 SIFT features; 10,000 randomly extracted features per class. Codebook ③, with $K = 256$, seems to perform best for binary classification of bathroom.

3.2 Binary Classification

One specific class vs all other classes, 70% train and 30% test

Table 2: Binary Classification Results

algorithms	bathroom	bedroom	kitchen	dining room	living room
K-means, $l = 0$	69.75%	62.88%	50.00%	62.20%	63.21%
K-means, $l = 0, 1$	68.07%	66.41%	65.91%	65.09%	62.20%
SC ①, $l = 0$	70.75±4.00%	59.56±4.28%	63.44±5.00%	60.89±3.69%	66.86±4.38%
SC ②, $l = 0$	70.28±4.26%	58.28±3.82%	65.86±4.14%	60.61±4.02%	68.31±4.27%
SC ③, $l = 0$	73.53±3.04%	63.28±4.32%	63.58±4.35%	63.33±5.03%	67.78±3.65%
SC ①, $l = 0, 1$	70.28±3.50%	62.17±4.55%	65.42±4.40%	67.50±2.76%	67.03±4.25%

For results of K-means, the negative examples for each class were selected evenly among the other four classes, and all the data available were used, i.e., 197 images for bathroom, 663 images for bedroom, 275 images for dining room, 535 images for kitchen, and 507 images for living room.

For the results of sparse coding with $l = 0$ the negative examples were selected evenly among the other six classes, and the number of positive images for each class was fixed to be 192 (132 train, 60 test).

For the results of sparse coding with $l = 0, 1$, the negative examples were selected from the other four classes only. The number of positive images was still fixed to be 192. Hence, we actually had the number of training data points much less than the dimension of the feature vectors (128 $\times$ 5).

3.3 Multi-class Classification

We ran multi-class classification on five classes - bathroom, bedroom, kitchen, dining room and living room, with again 70% train and 30% test. The baseline accuracy for random guess is 20%.

For codebook generated by K-means ($l = 0$ and 1, all images used):
Using Crammer and Singer algorithm: Accuracy = 34.67%
Using one vs all algorithm: Accuracy = 36.00%

For codebook generated by sparse coding ($l = 0$, 190 images per class):

Using Crammer and Singer algorithm: Accuracy = ① 32.97±3.68%, ② 34.60± 2.91%, ③ 37.70± 3.84%

Using one vs all algorithm: Accuracy = ① 37.33±1.95%, ② 36.92± 2.29%, ③ 44.58± 2.89%

4 Conclusion

From our results for binary classification for each category, we can see that bathroom has higher accuracy compared to others. It makes sense since bathroom is the most distinguishable among the five classes and has very different objects and layout. Sparse coding slightly outperforms K-means on bathroom and living room classification while the opposite case for the other three classes. The results from sparse coding for $l = 0, 1$ did not improve the accuracy possibly due to small number of data points available for training, much less than the dimension of the feature vectors. We could not generate $z^{(i)}$'s for the entire dataset for $l = 0, 1$ using SC-based codebook due to time limitation. The computation was fairly expensive for the steps involved, i.e., generating all the keypoints, doing l_1 feature-sign, and max-pooling.

From the results for multi-class classification, although the accuracy is not high, our results are better than the baseline random guess. Sparse coding with $l = 0$ and one vs all algorithm did particularly well compared to other combinations of algorithms, especially with codebook ③ where we had $K = 256$. These results seem to suggest the underlying structural difference among the five categories that we explored.

5 Acknowledgement

We would like to thank Will Zhou for providing us with ideas, advices, and recommending us the paper *Linear spatial pyramid matching using sparse coding for image classification* [9], from which this project was originated.

References

[1] Machine Learning Group at National Taiwan University. Liblinear – a library for large linear classification. http://www.csie.ntu.edu.tw/~cjlin/liblinear/.

[2] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. http://ai.stanford.edu/~hllee/softwares/nips06-sparsecoding.htm.

[3] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *In NIPS*, pages 801–808, 2007.

[4] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[5] Ariadna Quattoni and Antoni Torralba. Indoor scene recognition database. http://web.mit.edu/torralba/www/indoor.html.

[6] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *CVPR*, pages 413–420. IEEE, 2009.

[7] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, pages 3485–3492. IEEE, 2010.

[8] Zhenhui Xu. Sift extractor from open pattern recognition project. http://www.openpr.org.cn/index.php/Code-of-Individual-Algorithms/40-SIFTextractor/View-details.html.

[9] Jianchao Yang, Kai Yu, Yihong Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1794 –1801, june 2009.