CS172 Computer Vision I: Image Classification based on BoF , SPM and SVC

Ziqi Gao
2018533193
gaozq@shanghaitech.edu.cn

Abstract

In this paper, Bag of Features (BoF) is implemented for feature extraction and Support Vector Machine (SVM) are used for training in classification part. In addition, Spatial Pyramids Matching (SPM) is implemented for better feature extraction; linear SVC and non-linear SVC are compared for the classification tasks. All different approaches are implemented in the code and experiments are carried on the Caltech 256 Dataset.

1. Problem Formulation

Object recognition is an important and challenging area in computer vision, and one of its tasks is image classification. Traditional recognition pipeline includes hand-designed feature extraction and trainable classifier, which extract information from image pixels and generate object classes finally. In this paper, Bag of Features (BoF) is implemented for feature extraction and Support Vector Machine (SVM) are used for training in classification part. In addition, Spatial Pyramids Matching (SPM) is implemented for better feature extraction; linear SVC and non-linear SVC are compared for the classification tasks.

2. Algorithm

2.1. Bag of Features

Bag of Features, a.k.a. Bag of Visual Words, represents images as orderless collections of local features. It originates from Bag of words, which use orderless document representation: frequencies of words from a dictionary to describe context. Instead of using word frequency, BoF uses visual vocabulary frequency for constructing the dictionary (codebook), whose components are the result of clustering among the images, which are quantized by image descriptors as vector forms.

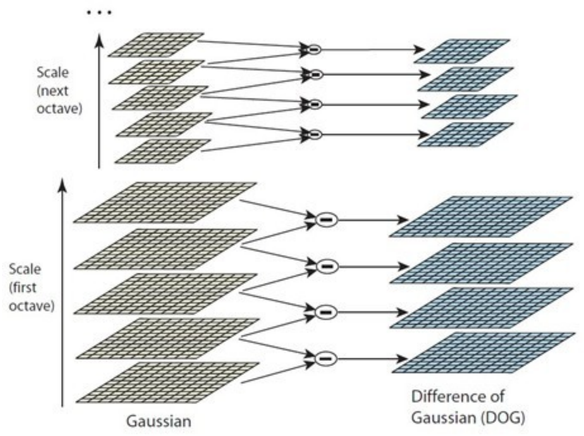

Figure 1. Gaussian paryamid generation in SIFT

2.1.1 Local Feature Detection and Description - SIFT

To be specific, BoF is based on Scale-Invariant Feature Transform (SIFT) for local feature detection and description, which extracts extracting distinctive features invariant to image scale and rotation. Following are the major stages of computation used to generate the set of image features[2]:

- Scale-space extrema detection: The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.
- Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.
- Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location

- for each feature, thereby providing invariance to these transformations.

- Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

In implementation, a SIFT descriptor is a 128-d vector for each keypoint.

2.1.2 Codebook Generation

After extracting descriptors of keypoints of the images, BoF generates codebook of the visual words of size K by clustering those N descriptors. This is implemented using K-Means, a clustering algorithm that partition N vectors into K clusters using random initialization of K-center points, assigning each point to its nearest center, replacing the K centers with the mean of each center iteratively.

To reduce the temporal and spatial cost, MiniBatchK-Means is used in implementation. It uses small random batches of data of a fixed size to improve spatial performance and takes small randomly-chosen batches of the dataset in each iteration to improve temporal performance.

2.1.3 Feature Quantization

In this part, BoF represent each image by the frequency of the appearance of visual words. Specifically, for each descriptor, it can be clustered into one of the K key words. For each image, it can be quantized into the distribution of the K keywords appearances, which form a histogram in implementation.

In this way, BoF finds an effective way to represent images.

2.2. Spatial Pyramids Matching

Spatial Pyramids Matching (Spm) is based on BoF, with improvement on utilizing multi-levels spatial information of a singe image. This technique works by partitioning the image into increasingly fine sub-regions and computing histograms of local features found inside each sub-region.

A toy description of the spacial property of SPM is shown in figure 2. The image has three feature types, indicated by circles, diamonds, and crosses. At the top, we subdivide the image at three different levels of resolution. Next, for each level of resolution and each channel, we count the features that fall in each spatial bin. Finally, we weight each spatial histogram by a pyramid match kernel. [1]

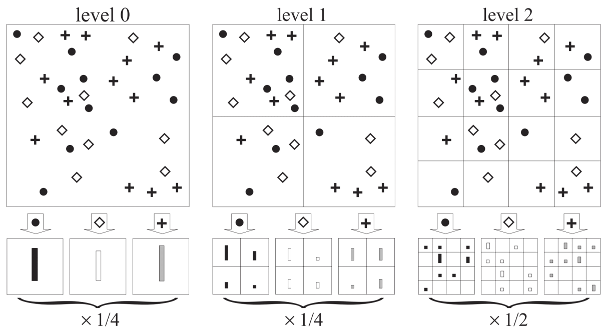

Figure 2. A toy description of SPM

2.3. Support Vector Classifier

Support Vector Classifier is based on Support Vector Machine, a binary classification technique. Support Vector Machine uses a hyperplane that maximizes the margin between the positive and negative examples to classify data. SVC can be achieved in one-vs-one or one-vs-all SVMs, and the latter one has a better performance after doing experiment on a 32-classes sample of Caltech 256 Dataset.

To better deal with non-separable data, we can use non-linear SVM, whose central idea is to map the original input space to some higher-dimensional feature space where the training set is separable. Kernel trick can be used here.

In implementation, LinearSVC is used for linear SVM and SVC with gamma, a hyperparameter of the radial basis function kernel, being adjusted is used for non-linear SVM (kernel SVM).

3. Implementation

3.1. Code Repository

- main.py: a Python file containing implementations of BoF + SVM & SPM + SVM on Caltech 256 with 15 training images per class.

- 256_ObjectCategories: a repository containing the Caltech 256; now empty for homework uploading.

- data_handler.py: generate label.txt, train.txt and test.txt according to CLASS_NUM and TRAIN_SIZE.

- train.txt, test.txt: a set of paths + correspongding labels to images in training and testing dataset.

- label.txt: a set of all labels.

- result.txt: the results found in experiments.

3.2. How To Run

Download caltech 256;
Run python main.py.

3.3. Implementation detail

In main.py, BoF, SPM, SVC are implemented based on the algorithm specified above. Important third-party modules used are numpy(1.13.3), cv2(3.4.2), scipy(0.19.1), and sklearn.

For KMeans, sklearn.cluster.MiniBatchKMeans() is used, with K being set to 2000 for 257 classes of images. This number is set by doing experiment on 32-classes Caltech Dataset and find the proper ratio.

For SVC, sklearn.svm.LINEARSVC() is used for linear SVM and sklearn.svm.SVC() is used for kernel SVM. They both have hyper-parameters C, for which I chose 7 values (0.001 - 1000) and used the best one found. For kernel SVC, gamma is introduced for carrying out non-linear SVM.

For the pooling part, np.bincount() is used with considering the weight of each vote.

The result is shown by GridSearchCV.best_estimator_ for the parameters and sklearn.metrics.classification_report for showing the accuracy on each class.

3.4. Experiments: Problem encountered and Result

3.4.1 BoF + Linear SVM on Caltech 256

Setting codebook size K as 2000, I got results below:

Size	15	30	45	60
Accuracy	0.16	0.20	0.23	0.24
# of SIFT	3947520	7895040	1184256	15790080

The results show a linear growth in memory occupation in linear increase of dataset size. The accuracy is growing with more and more training data, but the rate of its growth is decreasing.

As can be seen in the last line, the original number of SIFT descriptors is too big, not to mention the actual size should be multiply by 128, which causes many memory issues. I tried to solve this issue in many approaches. First, I assembled a high performance computer with 64 AMD Ryzen threadripper 3970x 32-core processorS, 64GB memory, and 1T SSD. But I still found that memory was not enough, so I set swap into 400GB and optimized the code using gc module in python to do garbage collection efficiently, which eventually got me the results. However, the running time was still unacceptably long(like 2 days? orz) eon this computer; and its memory performance was poor too. So I tried to find another way to optimize the code for the following experiments.

It is astonishing but true that the best parameters found is always *LinearSVC(C=0.001, class_weight=None, dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=1000, multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0).* So I used this parameter for 3.4.2 to reduce time.

Also, the SIFT descriptors are sampled in the following experiments to save time and space; new python memory optimization trick is used in hope of improving the results.

3.4.2 SPM + Linear SVM on Caltech 256

To save time and memory, I used level-1 SPM in this experiment.

To illustrate the guess about the parameters is valid, the best parameter found on 15 images Caltech 256 with SPM = 1 is *LinearSVC(C=0.001, class_weight=None, dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=1000, multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0)*

which turned out to be exactly same as 3.4.1's result, illustrating the idea above.

Size	15	30	45	60
Accuracy	0.21	0.25	0.28	0.30

The results are consistent with the BoF results: its accuracy is growing with more and more training data, but the rate of its growth is decreasing.

3.4.3 Linear SVM & Kernel SVM comparison on sampled Caltech 256 (32 classes)

Image representation techniques used in this experiment is a 2-levels SPM. To save time, this experiment is conducted on 32 classes of the Caltech 256 dataset with K = 300. RBF Kernel is chosen because of its adaptability in different scenarios and good performance. Gamma is the hyper-vector of the RBF Kernel.

SVM Type	Linear	RBF Kernel
Accuracy	0.36	0.39

The best parameters found:

LinearSVC(C=0.01, class_weight=None, dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=1000, multi_class='ovr', penalty='l2', random_state=None, tol=0.0001, verbose=0).

SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False).

It is shown that RBF (Radial Basis Function) is the best option of kernnels in 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed', which is accordance with common sense and the LinearSVC result.

4. Some Random Thoughts

Even the result of SPM + SVM is not very good on Caltech 256 Dataset. The number of images of this dataset is large, but many of the images are very similar with blank background. Thus, Nearest Neighbor on resolution-reduced images (which neglect the details so the similar images will look almost or exacly same) may have a better performance than the traditional BoG or SPM + SVC pipeline.

References

[1] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories". In: 2 (2006), pp. 2169–2178.

[2] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110.