Coursework 3 Scene Recognition

Feng Xie – fx1n18, Wei Deng – wd1u18

1 Introduction

In this coursework, we were given 1500 images for training which consist of 100 images for each of the 15 scene classes, and 2985 images for testing. The measuring criterion is based on the proportion of number of correct classifications to the total number of predictions.

The programming language we used for this coursework is Python. The main related frameworks we used were sciki-learn and opencv-python.

The second part of the report will focus on describing how we implemented the classifier and the third part will demonstrate the results of our experiments.

2 Methodology

2.1 Run #1

'Tiny image' feature refers to the one when we crop the image to a square about the centre and resize it to a small fixed resolution. In run one we resized it to 16 by 16. And we flattened the tiny image to a vector and used it as a feature for the classifier. Also, zero mean and unit length for one image were achieved by subtracting the mean value from each feature vector and dividing the zero-mean vector by its Euclidean distance.

In run #1, we used k-nearest-neighbour classifier to train the tiny image features. To find the optimal K value, we used the method of GridSearchCV in sklearn with which we can integrate together the processes of using different K values for training and using 10-fold cross validation for estimating the performance of each model. Since we were using tiny image features in run#1, the time cost for using the grid search to tune parameters is quite acceptable.

2.2 Run #2

In run #2, we extracted fixed-size patches from images and flattened every patch into a vector. All patches were mean-centered and normalized separately before being applied in KMeans clustering. It was found that the training process of traditional KMeans for run#2 is very time-consuming and it requires too much memory so that we chose to use MiniBatchKmeans which is an alternative version of KMeans but more effective. MiniBatchKmeans uses a fixed batch size for training in every iteration and although the result would be slightly different than KMeans, it is still acceptable.

With KMeans clustering, we built a vocabulary for applying Bag of Visual Words. Every patch would then be mapped to a visual word. After converting every image to the presentation of Bag of Visual Words, we used Logistic Regression to train a model and evaluated the model by apply 10-fold cross validation on the training set.

To start, we tried 500 clusters and 8 by 8 patches which were sampled every 4 pixels in the x and y directions. By doing experiments, we found that the score in 10-fold cross validation can be raised when using a bigger number of clusters and a smaller size of patches.

The logistic regression classifier was initialized as below:

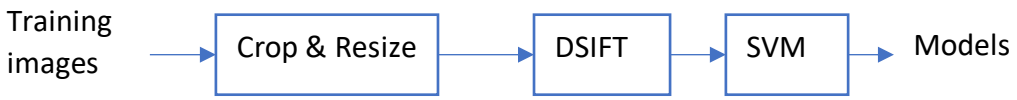
```
clf = LogisticRegression(multi_class='ovr', solver='  sag', n_jobs=-1)
```

We explicitly set “multi_class=ovr” so it would use the one-vs-all strategy. This strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. “n_jobs=-1” means it will use all the processors available. The solver of LR was set to “sag” rather than the default “liblinear” because it is more suitable for large dataset.

2.3 Run #3

In run #3, we extracted dense sift descriptors from images and explored two different ways to use them. Different from the norm sift features, the location of each key point is from a predesigned location rather than from Lowe's algorithm.

2.3.1 Method 1

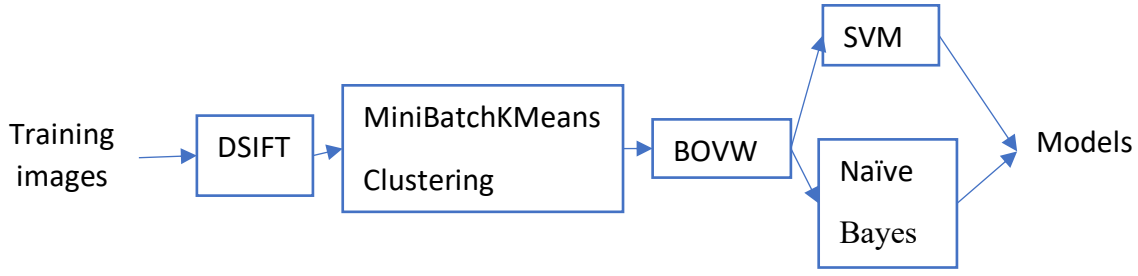


The first method we explored with dense sift descriptors was inspired from run #1. Instead of extracting “tiny image” features, we extracted all the dense sift descriptors from every image and flatten them into a vector and use it as a feature for later use in the classifier. We used Support Vector Machine as the classifier with the polynomial kernel which is non-linear. Because it is time-consuming to train models with svm, we split 33% of the training set for testing the performance instead of using cross validation. The following code shows how we trained and evaluated the SVM model.

The parameters that can be tuned are the step size when extracting dense sift features and the resized resolution of the image. The result will be shown in the next part of the report.

The theory supporting why we used this method is that the dense sift descriptors are able to describe the local interest points much better than tiny image features. As long as we set an appropriate step size for extracting dense sift features, preventing the dense sift descriptors from being too dense, a flattened vector of all dense sift features would be a good representation of an image.

2.3.2 Method 2



The second method we explored with dense sift was inspired from run #2. Each sift descriptor would be treated as a visual word. Then we applied MiniBatchKMeans to learn a vocabulary. All the sift descriptors were scaled before applying the clustering. After building a model of Bag of Visual Words, we used Naïve Bayes and Support Vector Machine to train classifiers. The metric for the Naïve Bayes was 10-fold cross validation while the metric for SVM was splitting 33% of the training set for testing.

The parameters that can be tuned are the number of clusters and the batch size when doing MiniBatchKMeans and the step size when extracting dense sift features. After getting the best classifier we can find by doing experiments, we predicted the labels of the testing set. When doing the scaling process, we used the scaler that had fitted the training set to transform the testing set.

The reason why the performance of this method is better than the first one might be that apart from extracting appropriate number of dense sift descriptors, we clustered them into different groups and applied Bag of Visual Words, which made the feature vectors more representative.

3 Results

3.1 Run #1

For KNN classifiers, we did experiments with the range of K values set to 1 ~ 30. The optimal K value was 9 while the accuracy in 10-fold cross validation was 0.223.

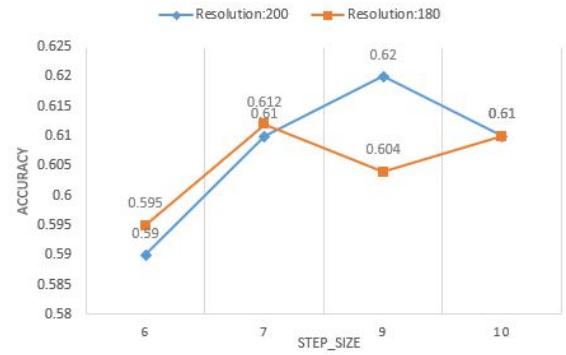
3.2 Run #2

We tried different parameters with the classifier of Logistic Regression. The result is shown as below.

Run#	Classifier	Solver	#clusters	Stride	Patch size	Accuracy
Run#2	LR	liblinear	600	4	8	0.65
	LR	liblinear	600	4	6	0.63
	LR	liblinear	600	4	6	0.65
	LR	liblinear	700	3	6	0.67
	LR	sag	700	3	6	0.69

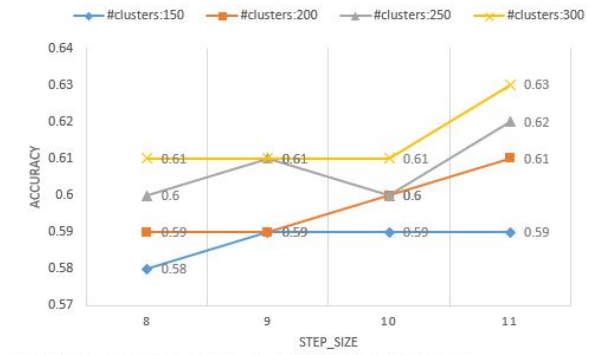
3.3 Run #3

In the first method of run #3, we used SVM classifier and tuned the resolution of the resized image and the step size. In the second method of run#3, we used SVM and Naïve Bayes as classifiers and tuned the parameters of the number of clusters and the step size.



RUN #3 METHOD 1 WITH SVM

RUN #3 METHOD 2 WITH SVM



RUN #3 METHOD 2 WITH NAIVE BAYES

4 Conclusion

In run #1, we used the tiny image features which are quite simple and the accuracy is just as low as 22% in 10-fold cross validation. In run #2, it seems that by tuning appropriate parameters, the accuracy can reach as high as 69% with a logistic regression classifier. In run #3, with two different methods, the accuracy cannot still reach a higher score than run #2. In future experiments, a more sophisticated feature should be considered to build a better classifier.

5 Individual Contributions

Feng Xie is responsible for writing well-structured and commented code after we discussed the methods for each implementation and finished our own version of code. Wei Deng wrote the first version of the report and Feng Xie wrote the final report based on Deng's version.

6 References

[1] https://scikit-learn.org/stable/

[2] https://docs.opencv.org/3.4.3/d6/d00/tutorial_py_root.html