Coursework 3: Scene Recognition

COMP6223 Computer Vision (MSc)

Team Elephant
Vasin Srisupavanich ID:31300162 vs2n19@soton.ac.uk
Thanakorn Panyapiang ID:31446612 tp2n19@soton.ac.uk
Zixuan Cai ID:31303538 zc1g19@soton.ac.uk
Mingjun Xie ID:30971586 mx4n19@soton.ac.uk

Statement of Contributions

Run 1 was completed by Vasin Srisupavanich.
Run 2 was completed by Thanakorn Panyapiang.
Run 3 was completed by Zixuan Cai, Mingjun Xie and Vasin Srisupavanich.

Run 1

Run 1 was implemented using **k-nearest neighbour classifier** and "**tiny image**" representation. All the implementation was done using Python, OpenCV, NumPy and scikit-learn library. Tiny image is one of the simplest feature to do the image classification. To create tiny image vector, we implemented in four steps. First, if the image is not in a square shape, the image is cropped to be a square size about the center. The size of the image is chosen by selecting the length that is shorter between the width and the height of the original image. This is done in order to keep the aspect ratio of the original image when resizing. Second, the image is then resized to 16x16 square image. Third, the 2d matrix of the image is flatten into 1d feature vector. Finally, the feature vector is normalized to be zero mean and unit length (feature scaling). This is necessary and help increase the accuracy when using k-nearest neighbor or other machine learning algorithms when the distance between data point is important.

K-nearest neighbor algorithm is one of the simplest supervised machine learning algorithm. It requires no training process, and classify new data based purely on the proximity of the data using the distance between data points. During the prediction process, the algorithm will selected k points that have the shortest distance between the testing and training data. If k is 1, the label of that training data is returned. However, if k is more than 1, the resulting label is returned base on the majority vote. The advantage of k-nearest neighbor is that it can have complex decision boundary, and can be used easily with multi-class problem. However, the disadvantage is that the algorithm can be very slow if there are a lot of training example or the data has a lot of dimensions.

In this run, we used scikit-learn's implementation to create the model. L2 euclidean distance is used as the distance metric. To select the optimal number of k, we implemented 10 fold cross validation on the training data, and use the one (k=7) with the highest average score. The accuracy from doing cross validation on the training data is very similar for all the range of k (1-20), which is roughly around 20% accuracy. We also tried with different sizes (8x8, 32x32), however the results from running cross validation are roughly the same. Lastly, we tried to compare the classification results from running k-fold cross validation on training data between normalizing the feature vector to be zero mean and unit length and without normalizing, the result is that without normalization the accuracy drop to around 16%, which is 4% less than classification with vector normalization. This is because k-nearest neighbor classify data based on the distance between data points as mentioned in the first paragraph.

Run 2

In this section, we implement scene classifiers using the bag-of-visual-words technique. The implementation can be divided into 4 steps as follow.

Feature Extraction

The feature we use for this classifier is densely-sampled pixel patches. Each image is divided into several 8x8 patches. Each patch is flatten into a single vector and down-sampled with 25 percent rate(i.e. take one pixel from every four pixels). Then, each pixel is normalized to have a unit length and zero mean.

Building Visual Vocabularies

After feature vectors have been extracted, the next thing is to build visual vocabularies. All feature vectors are grouped using K-Means algorithm. In choosing the value of K, we run 5-fold cross validation on 3 values of K: 500, 600, and 700. The best value of K we get is 700 with an accuracy of 36 percent on test set(300 randomly sampled images from all training images).

Building Image Histograms

To build train images histograms, each feature vector for each image from the first step are assigned into a cluster by finding the nearest centroid obtained from the previous step. Then, we compute the frequency of each visual vocabulary and normalize the histogram.

Training Classifiers

We use Linear Support Vector Machine to classify test images. To train the classifier, normalized histogram of training images are used as a training data. For test images, feature vectors are extracted as described in the first step. Then, we compute test images normalized histograms before classify them using linear support vector classifier.

Run 3

In this section, our implementation can be divided into two parts, feature extraction by Dense-SIFT and Classification by linear SVM [1].

Feature Extraction

Dense-Sift Feature Extraction of Every Image

We extract the features from every image by Dense-SIFT. Dense-SIFT extractor use a patch with a given size and move from the starting position with a given step distance until the patch cover the whole image. The patch contains $4 * 4$ bins, in which we calculate gradients of 8 directions started from the center of the bins. So the features extracted by Dense-SIFT has high toleration against rotation as well as scaling [2]. In order to make the features fully represent a whole image we set step distance as 3 and patch size as 3.

Visual Words Building

We use Dense-SIFT to extract the features which are used to build the visual words from all images in training set. In this part, we decide to set step distance as 15 which is looser than that when extracting the features above and patch size as 3 after testing for many times since the visual words extraction don't need so many patches in each image when doing the Dense-SIFT. We can't use the features from Dense-SIFT directly, because the shape of the Dense-SIFT features of one image is a matrix like 280 by 128 (this is what We got), not a vector, which means that we can't use normal classifier to classify this kind of features. Therefore, we need to build visual words at first. Then we cluster these features which are 128 dimensions vectors by K-Means Clustering and get the visual words. After testing the value of K for many times, we found that when $K = 230$, the performance is relatively better. Each visual word is a vector with 128 dimensions.

Histogram Building for Every Image

In this part, we calculate the frequency of each visual word appears in one image and build the histogram. The x axis of the histogram is 230 visual words and y axis is the number of each visual word in one image. Then we change this histogram into a vector with a shape of 230 dimensions which is much easier to classify by SVM.

Parameter Adjustment

To reach a higher accuracy rate, we try to change parameters in the functions. As we know, if we have more vocabulary, we can describe a article in more detail. So, more visual words is always better. However, if we set the value of K too large when we clustering the features, the performance won't be good which means that we need to get a suitable K rather than a larger K since if we cluster too much centers, we may get the letters but not words. After testing, we think 230 is almost the most suitable number of visual words.

Step distance is a key parameter of the algorithm. In the process of getting visual words, we set a relatively large step distance when extracting Dense-SIFT features from images, it is just like sparse SIFT feature extraction. After testing, we find that changing the step distance when we get the visual words just has slight effect on the prediction result. Therefore, we set the step distance of getting visual words as 15 to save processing time while the accuracy is relatively good. In the process of histogram building for every image, the step distance should be tight enough to fully cover the whole image. So we set the step distance as 3.

Patch size is also a very essential parameter for the performance. We assume that a smaller patch size may be better because if we choose a large patch size, it is like we use visual phrase or sentence but not visual words. And our testing result does support our assumption. So we set the smallest patch size, 3, in both visual words building and histogram building for every image.

Classification by SVM

At this stage, we use ten fold cross validation to estimate the generalization performance of our model since the accuracy rate will fluctuate when we use different parts of data set to train and test. Before we training the classifier, we need to normalize the feature vectors of all the images as some values in the vector are so large that they will have some bias while model training. At first we tried the non-linear SVM, but the performance is not so good, so, we choose linear SVM which has about 70% accuracy rate.

References

[1] Olgun M, Onarcan A O, Özkan K, et al. Wheat grain classification by using dense SIFT features with SVM classifier[J]. Computers and Electronics in Agriculture, 2016, 122: 185-190.
[2] Mark S. Nixon, Alberto S. Aguado. Feature Extraction Image Processing for Computer Vision, 2012.