Selected Topics for Computer Vision Coursework 1

Ashish Sundar
CID: 01058844
as13215@ic.ac.uk

Harshil Sumaria
CID: 01053573
hs2715@ic.ac.uk

1. Introduction

The subset of the Caltech101 dataset comprised of 10 classes from which 15 images were selected randomly for training and 15 for testing. SIFT (Scale-Invariant Feature Transform) was computed using the OpenCV toolbox[1], on the images in order to obtain 128 dimension feature descriptors. The images in Figure 1 show examples of SIFT features.

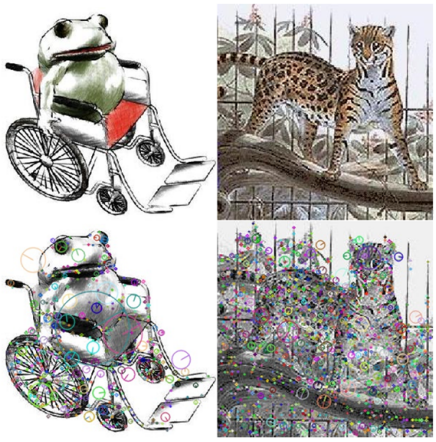

Figure 1. Two images from the 'Wheelchair' and 'Wild Cat' classes, with the SIFT patches circled below

Once these features were obtained, a codebook was computed (as detailed in sections 2 and 3 below) and histograms were generated of the training and test images. These Bag of Words representations were passed to the Random Forest Classifer for classification (section 4). This report outlines the theory behind the methods used, followed by the results in section 5.

2. K-Means Codebook

This was done by obtaining all the patches for all the training images in the classes concerned and applying a K-Means clustering algorithm to them. This was done to cluster similar patches, and thereby obtain patches that occur in a lot of images, or at least patches similar enough to patches that occur in other images. From each cluster, the mean image is obtained, and these form the visual codebook. The visual codebook is formed of visual 'words' which are the most commonly occurring patches that are unique enough to distinguish between different patches.

Initially, we used 16 patches to test and debug the code, but it was quickly found that this was too small of a size for a visual word set, so after experimenting with many bin sizes, we decided 256 bins was the optimum trade off between time efficiency and performance.

To get the histograms each patch in the picture is compared with one of the 256 average patches with respect to euclidean distance. A vector containing 256 values is created per image, and the closest set of the average patches per patch of the image is incremented proportionally to obtain a histogram. This histogram shows which bins are used most in the image and thus quantise the high number of initial patches (roughly 4000) to a very low number (256).

This vector quantisation process is implemented with nearest neighbours with a euclidean error function, and the class probabilities are used to add to the coding vector. This is done so that more than one average patch can be accounted for per patch in the actual picture. This is to make the algorithm more robust and invariant to minor disturbances. Figure 2 shows sample histograms obtained for the images in Figure 1, and displays the effect of changing vocabulary size.

The algorithm used was batched K-Means, which, instead of using the entire dataset per iteration of K-Means to fit the clusters, only uses a portion of the data. This means that it is much more computationally efficient and therefore time efficient, however it means that the performance does tend to go down as the sample size is much smaller.

3. Random Forest Codebook

As [2] states, K-Means clustering is slow due to it's reliance on nearest neighbour search in order to quantise the training and test images. The data is of a high dimension (128), and it has proven hard to accelerate search in high dimensions.

Due to the high dimensionality of the data, it is clear

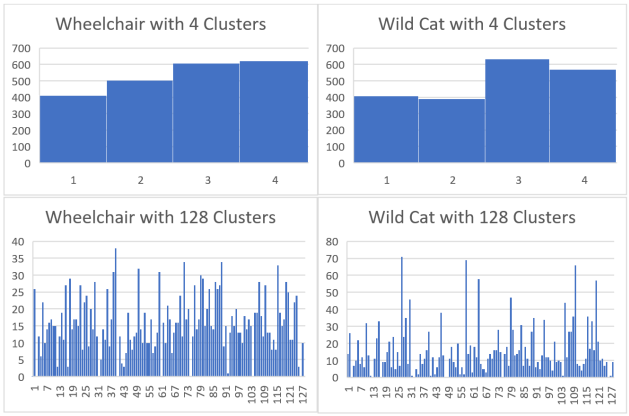

Figure 2. Histograms of two images obtained using the K-Means codebook for varying numbers of clusters (vocabulary size)

an ensemble method will be required. An ensemble of random trees offers logarithmic time coding as well as a similar level of performance. In this case, clustering trees were used. This process works by utilising the trees as a method of splitting the data such that each leaf becomes a visual word. Once the codebook is trained using a random selection of feature descriptors from the training set, it is then used to transform the training and test images into histograms, as before.

Using the 'Random Trees Embedding' algorithm from the 'Sci-Kit Learn' library [3] allowed us to train an unsupervised forest and transform the training and testing data efficiently. The algorithm proved to be much quicker than the K-Means codebook. The outputs of vector quantisation are much more complicated due to the tree structure. The vocabulary size is undetermined until runtime as it is less than or equal to the product of the number of trees and the number of leaves. It is determined by how the data is fit to the codebook, whereas the K-Means vocabulary size can be explicitly determined by choosing the number of clusters.

4. RF Classifier

The classifier used with the codebook representations of the data is done with a classification based random forest. This is, as mentioned before, because random forests are very quick at running through data, if requiring slightly more computational power to test. This is because of their logarithmic scaling with the size of the data. The nature of forests also mean that they do not need to be regularised, as the forest itself is an ensemble method which acts to regularise the individual decision trees, which naturally end up overfitting when used separately.

This ensemble method is based on bagging, where each tree, instead of being given all the data with all the dimensions, will only be given some of the dimensions and some samples of data. If enough trees are used, then almost all of the data will end up being represented. We chose to use 60% of the data and 70% of the data. This combines to form 42% of the data per iteration, and with 8 trees, 98.7% of the data is used. Therefore with 100 trees we are confident all of the data is used.

5. Results

The overall trend seen by our results, is that of the K-Means codebook being much slower to train but having equal if not better performance than the Random Forest codebook. We expected the RF Codebook to be quicker as well as having a better performance, due to the high dimensionality of the data and the benefits of the tree structure however this potentially may not have been the case due to the libraries used.

The use of the sci-kit learn library did not enable testing of different weak-learners. Our chosen algorithm uses axis-aligned splits due to this being the most common splitting strategy, with a compromise between performance and computational complexity.

5.1. Classification

The confusion matrices in figures 3 and 4 show the classification results for both codebooks. In these cases, the K-Means codebook had marginally better performance, however other simulations, which will be discussed in section 5.2 showed much better performance.

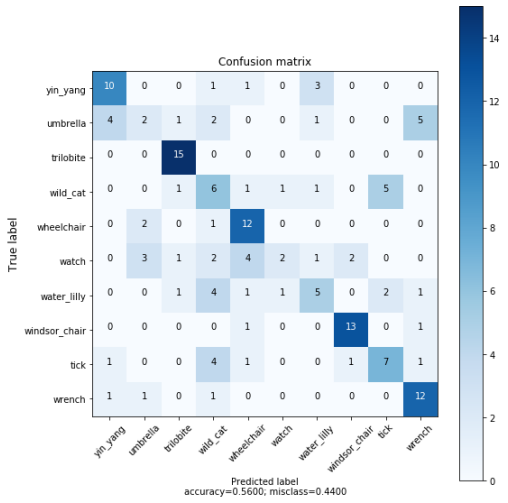

Figure 3. Confusion Matrix with the K-Means Codebook

In both cases, the classifier performed extremely well with the 'Trilobite' class. This is due to the similarity of all of the images in the set. K-Means was weakest with the watch and umbrella sets, only classifying 13% of images correctly in these classes. 33% of umbrellas were mis-

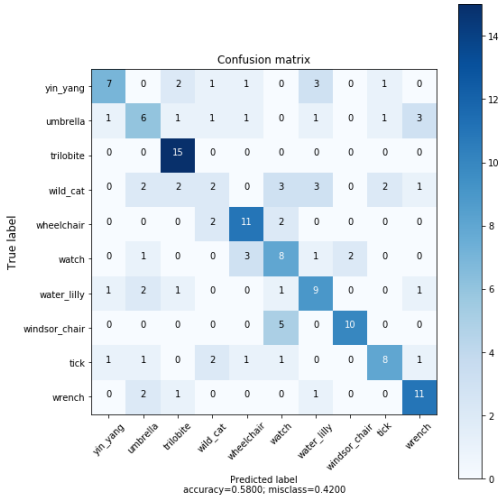

Figure 4. Confusion Matrix with the Random Forest Codebook

classified as wrenches. For the Random Forest Codebook, the least correctly classified class was the Wild Cat, only achieving a 13% success rate. The greatest misclassification was Windsor Chairs as watches, with 33% misclassified.

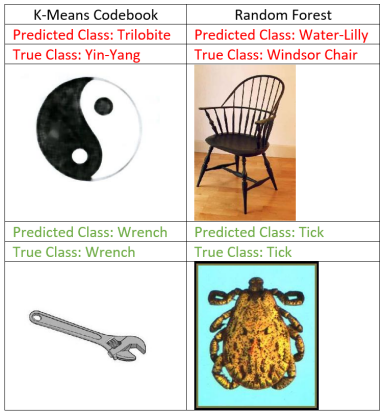

Figure 5. Example success and failure images

The optimum parameters for the classifier for each codebook were determined using the 'GridSearchCV' function from the 'Sci-Kit Learn' library[3]. The number of trees, the maximum depth of each tree, and the minimum number of samples required to split a node were varied from 10-1000 trees, depths of 5-50 and minimum samples from 2-5. The optimum classifier was found to be 100 trees, with a minimum numbers of samples to split at each node of 3 and a maximum tree depth of 15. Too low a depth risks underfitting the model, whereas a high depth leads to a risk of overfitting due to a reduced number of points at the leaf nodes. Therefore a compromise depth is chosen to avoid under or overfitting, as well as the longer computation times for an increased depth. Increasing the number of trees reduces overfitting. The degree of randomness was explored but had little effect on the classification rate.

5.2. Vocabulary Size

The choice of this parameter is key for performance. A low vocabulary size will lead to poor generalistion and underfitting, whereas a larger vocabulary size will improve performance but increase computation time and memory overheads. Therefore a compromise must be struck. Varying the number of clusters for the K-Means codebook showed the best point to be at 256 clusters, where a classification accuracy of 66% was achieved. Above this, the computation time was taking minutes longer for negligible gain in performance. A similar trend was seen for the random forest codebook.

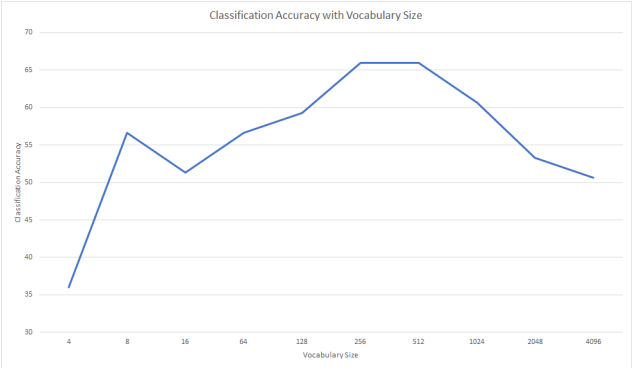

Figure 6. The relationship between Vocabulary Size and Classification Accuracy for the K-Means Codebook

5.3. Time Performance

Whilst the random forest codebook had a slightly worse performance relative to the K-Means codebook, it was markedly quicker to train and transform the data. For a vocabulary of 256, the K-Means codebook took 59 seconds whereas the random forest codebook took 42 seconds, quicker by 17 seconds. The performance difference is much clearer for a larger vocabulary size. With a vocabulary size of 2048, K-Means took 4 minutes 31 seconds but the random forest took 56.8 seconds for a vocabulary size of 2000. This is due to the time overhead of nearest neighbors.

5.4. Comparison

Based on our results, it is clear to see both codebooks have their own strengths. The K-Means codebook leads to a greater classification accuracy however takes a long time to train. The tree structure of the random forest codebook is simpler and therefore quicker to train. The training time for the Random Forest codebook also scales better than the K-Means codebook.

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. pages 985–992. MIT Press, 2007.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.