Codebooks for classification using Random Forests

Martin Ferianc, Divyansh Manocha
{mf2915, dm2515}@ic.ac.org, CID:{00984924, 01053537}

1. Introduction

This work demonstrates the application of Bag of Words (BoW) on a classification task on the Caltech 101 dataset [1]. It uses SIFT [2] descriptors, which are subsequently quantised into a codebook by a k-means or random forest (RF) algorithms. Classification is performed by a RF.

The original Caltech 101 dataset contains small, centered RGB pictures of objects belonging to 101 categories. For the ten classes, 15 images were chosen for training and testing respectively - eliminating the possibilities of a bias towards a certain class.

2. Bag of Visual Words

BoW corresponds to a histogram of a number of occurrences of particular patterns in a given image. Using BoW is advantageous because BoW enables to detect objects of varying sizes and affine transformations, while obtaining low-feature dimensions which are more compute and memory efficient.

We created a BoW representation using k-means and RF algorithms. Both codebooks were created by first extracting $d = 128$, d dimensional SIFT descriptors from all images. Second, randomly selected $100k$ of these features were sampled out of the training images to form a training set for creating the codebook by k-means clustering.

2.1. k-means

2.1.1 Vector Quantisation Process

The training set was used to create a codebook, consisting of k codewords. The codewords corresponded to cluster centres in the k-means algorithm. They also represent similar original features.

We initialise the centres with the *k-means++* [3] algorithm. Then, we associate all features in the training set with the nearest cluster. Subsequently the cluster centres are recomputed with respect to their corresponding points based on Euclidean distance. The *k-means++* technique has the compute complexity $\mathcal{O}(\log k)$ and a guaranteed convergence. After the convergence, each descriptor for each image is mapped to the closest cluster centre to give a histogram of k bins per each image in training and test sets. The mean histograms corresponding to individual classes are shown in Figure 10.

Despite the simplicity, it should be noted that k-means only converges to a local minima and is computationally demanding, both during optimisation and testing. The procedure is described in Listing 1.

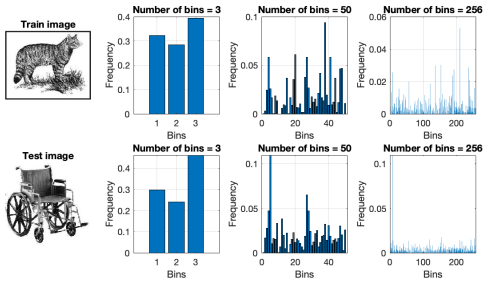

Figure 1: Sample normalised quantisations for varying number of bins with k-means.

2.1.2 Vocabulary Size

As seen in Figure 1, if a small number of codewords, such as $k = 3$, is used it can result in generalising and finally underfitting to the data, where histograms might be similar for images from different categories. A large value of k, for example $k \geq 256$ as seen in Figure 1, can however result in overfitting to the train data. The new feature representation will be fitted towards noise and dissimilarities. Although, not conducted in this project, one method of estimating k would be cross-validation.

2.2. Random Forest Classifier

The BoW created features allow a RF classifier to be fit to the training images to classify objects into classes. The summation, instead of the product, of the tree outputs was used as the fusion rule. This section provides an outline of the performance optimisations conducted on the classifier with a k-means codebook. This involved tuning the number of trees, tree depth, randomness, vocabulary size and weak learners for splitting. Whilst the selection of the hyperparameters was largely based on the testing accuracy, the confusion matrices were also taken into account.

The order of tuning was important, as conducting a full grid search would be infeasible in the time frame of the project. Tuning was therefore conducted individually, setting all but one hyperparameter to be constant.

2.2.1 Number of Trees

Varying the number of trees, as shown in Figure 2, has a large effect on the accuracy at lower values - thereby a decreasing rate of change. It is well known that a RF is able to mitigate individual tendencies of trees to overfit. More trees tend to provide better performance as the averaging effect is observed. The example of Figure 9 also clearly demonstrates

this.

The computational times have been shown to increase approximately linearly. This is expected as theoretically to be $O(n_{trees} \times nlog(n))$, where n is the number of nodes. It should be noted that it is possible to parallelise the training. The optimal number of trees which was observed was **120**.

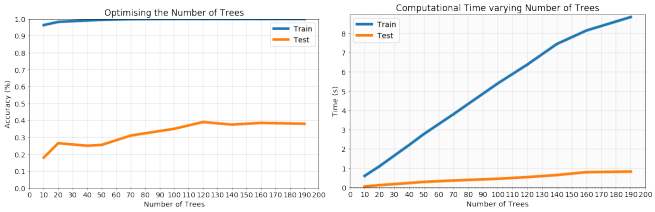

Figure 2: Variation in the accuracy with respect to the number of the trees (left). Variation in the computational time (right).

2.2.2 Maximum Tree Depth

The train and test errors with a varying number of maximum tree depth are illustrated in Figure 3. It is noted that accuracy peaks in the middle of the graph, and is lower at either end. This is because the RF classifier underfits the data by over-generalisation with a shallow tree, and overfits as leaf nodes have fewer points for a deeper tree.

The computational times have also been shown to increase almost exponentially. This would be because the number of nodes in a tree grow exponentially with the depth. Theoretically it should be $O(n_{trees} \times n \times d)$, as the depth of nodes is given by $log(n)$. The beset value was at a depth of **13**.

The confusion matrices allow us to make particular inferences, that an aggregate metric such as classification error cannot. It can be observed that the mis-classifications of a particular class are more prevalent in forests with lower number of trees, as expected, but also a larger depth.

2.2.3 Randomness

At each node that is not a leaf, a left and right child node retain the remaining data. The split itself is determined using the weak learners explored in Section 2.2.4. Bagging often leads to correlated trees if the dataset is not large, as is the case here. The *randomness* of the forest is determined by the number of split functions used. The information gain is maximised by increasing the randomness, increased by the number of random splits.

There was no significant gain in the performance metric by increasing the number of splits, albeit there was a small increase. However, the performance seemed to deteriorate after

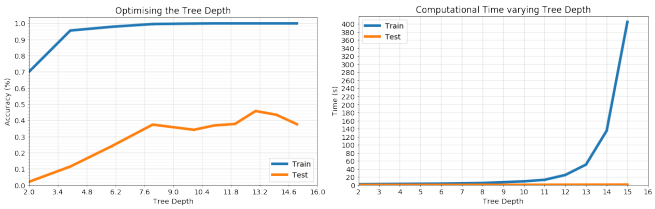

Figure 3: Variation in the accuracy with respect to the depth of the tree (left). Variation in the computational time (right).

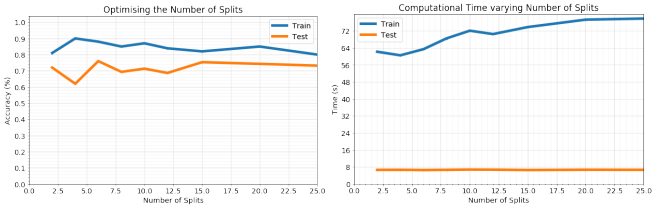

Figure 4: Variation in the accuracy with respect to the randomness between the trees (left). Variation in the computational time (right).

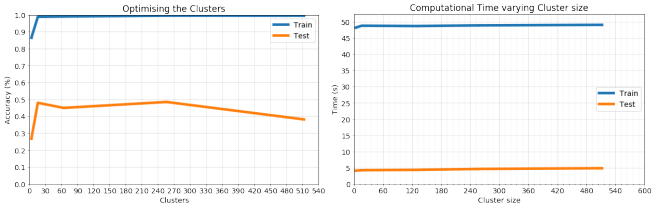

Figure 5: Variation in the accuracy with respect to the k-means cluster size (left). Variation in the computational time (right).

more than 20 splits. As would be expected, the computational time increased linearly. The optimal number of split functions was observed to be **6**.

2.2.4 Weak Learners

Two weak learners were explored, to split the dataset at each node. The functions use a geometric primitive, ψ, to separate the data. Axis-aligned weak learners splits the data with a line ψ aligned with one of the axes of the feature space: $h(\mathbf{v}, \theta) = [\tau_1 > \psi(\mathbf{v})\theta > \tau_2]$. The two-pixel test uses more than one dimension as compared to the former mentioned function. It also approximates the gradient, $h(\mathbf{v}, \theta) = [x_i - x_j > \tau]$.

Both learners are relatively simple and have a small difference to the computational time, as observed. Axis aligned was found to provide better results for the classifier with **47 %** test accuracy over 43 % for the axis-aligned.

2.2.5 Vocabulary Size

The size of the BoW, k clusters, has a significant impact on the performance metric.

There is observed to be a trade off between the training time and the generalisations of each class inferred. Figure 5 shows that increasing the number of bins results in improved classification accuracy, albeit a very small increase. It was observed that a vocabulary size from **256** onwards did not improve the performance by any significant amount, whilst still increasing the computational time significantly. Overfitting causes the accuracy to worsen thereafter.

The optimal hyperparameters as outlined above provide a test acccuracy of **76%**. The corresponding confusion matrix is seen in Figure 7. A tendency to misclassify a particular class as another is not observed, however classes 5 and 7 were more often misclassified than the others as depicted in Figure 6.

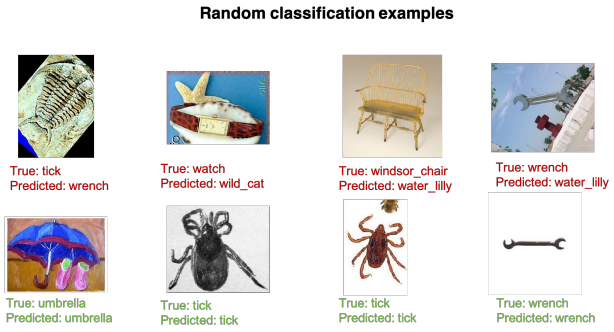

Figure 6: Example cases of correct and incorrect classification with k-means and a RF classifier.

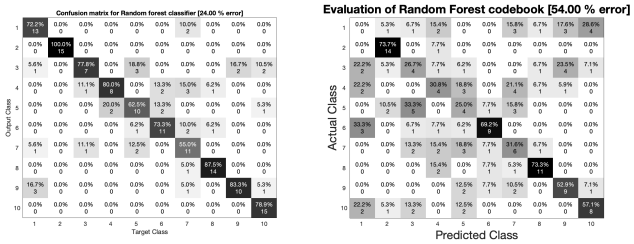

Figure 7: Confusion matrices of the best performing tuned RF classifier with k-means codebook (left) and tuned RF codebook and classifier (right).

3. Random Forest Codebook & Random Forest Classifier

3.1. Vocabulary Creation

A codebook may be constructed using RF instead of k-means. The information from the SIFT descriptors (more specifically a variant of them), returned from $\boldsymbol{vl}_{phow}$ [4], are used to construct the trees. Leaves of the RFs form as clusters, equivalent to that in k-means, in the codebook. The vocabulary size has the upper estimate, $k \leq n \times 2^{D-1}$, where n is the number of trees and D their maximum depth. For the detailed procedure refer to Listing 2.

For smaller trees, it is expected that k-means has superior discriminating power. This method is not expected to outperform k-means in accuracy which makes use of all dimensions simultaneously. It is also expected that the training time will be lower. The training time complexity of k-means is $O(DN'k)$, whilst it is $O(\sqrt{D}N'log(k))$ for the RF. N' is the number of patches, k the number of clusters and D is the dimensionality of the descriptor. Therefore, the codebook forest is expected, and is observed, to be more efficient than its alternative k-means construction.

3.2. Optimal Hyperparameters

The found optimal configurations can be seen in Table 1. Corresponding accuracy and training graphs, similar to that shown for the classifier, can be found in Figures 11 (number of trees) 12 (tree depth), 13 (vocabulary size), 14 (randomness) in the Appendix.

The corresponding confusion matrices can be seen in Figure 7. Classes 1, 3, 5 were often mispredicted as 10, 9 and 3 respectively. In fact Class 1 was never predicted correctly.

Table 1: Summary of the hyperparameter tuning using both codebook generation methods with a RF classifier.

Hyperparameter	k-means Codebook	RF Codebook	RF Classifier
Number of Trees	N/A	50	120
Max Depth of Tree	N/A	2	13
Number of splits	N/A	4	6
Vocabulary size	256	64	120
Weak Learner	N/A	Two-pixel	Axis-aligned
Test accuracy	76.0%	56.0%	Optimal
Train time	71.7s	58.5s	Optimal

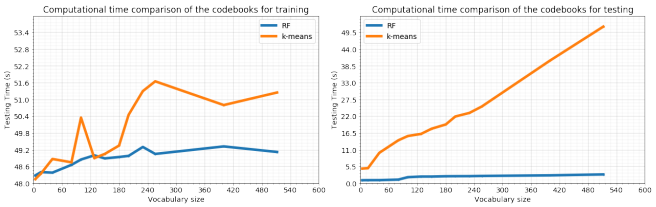

Figure 8: Training times (left) and test times (right) affected by the vocabulary size to compare k-means with a RF codebook. The measurement times are of growing the forest with the same classifier, which includes generating the codebook.

3.3. Comparison

Through the analysis conducted above, a brief comparison between k-means and a RF codebook was made. Overall accuracy was significantly better (by almost 20%) for the best k-means codebook and its classifier than the alternative, as shown in Table 1. Whilst such a large difference could be due to not reaching the optimal, it is certain that the k-means achieved better performance.

Consider the effect of varying the vocabulary size on both methods, as seen in Figure 8. From the discussion in section 3.1, it is expected that k-means has a linear computational complexity, whilst the RF will have a logarithmic complexity with respect to the vocabulary size. This is corroborated by our findings shown in Figure 8.

4. Conclusion

A brief comparison of the acccuracy and training time of the two methods of codebook generation has been articulated in Section 3.3. Considering the small forest construction required and its metrics in Figure 8, it was found that random forests scale better than k-means. However, the complexities associated with achieving a richer codebook are drawbacks that one must consider. Further work may involve cross validation to pick the optimal hyperparameters. Perhaps with more computational power and time, a more thorough analysis can be produced. Alternatively a minibatch approach to k-means may be used as an alternative to reduce the computational costs.

References

[1] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer Vision and Image Understanding*, vol. 106, no. 1, pp. 59–70, 2007.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[3] D. Arthur and S. Vassilvitskii, "K-means++: the advantages of careful seeding," in *In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[4] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms." `http://www.vlfeat.org/`, 2008.

1. Randomly initialise k Clusters;
2. Split features randomly to k groups;

while *!Stopping criterion* **do**
  for *Point in Data* **do**
    Calculate distance to cluster centres;
    Sort distances;
    Reclassify features w.r.t. to minimum distance to cluster;
  end
  Recalculate new clusters k based on point means;
end
histograms = None
for *image in set* **do**
  for *features in image* **do**
    n = Find nearest cluster from; k
    histograms[image][n]+=1; *//Increment bin in image histogram corresponding to bin*
  end
end

Algorithm 1: Bag of Words k-means Algorithm.

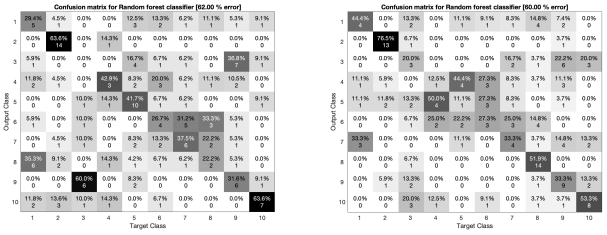

Figure 9: An example of the confusion matrices produced by two models with the same unoptimised hyperparameters, except for the number of trees (70 left, 140 right).

1. Set the parameters of the tree (depth, weak learner, optimisation function);

while *!Stopping criterion* **do**
  if *Leaf node* **then**
    Store labels and features; Compute statistics;
  end
  else
    Choose a feature for split;
    Compute gain by splitting;
    Split and create nodes;
    Continue for each child node;
  end
end
histograms = None
for *image in set* **do**
  for *features in image* **do**
    leaf index = Traverse tree with features
    histograms[image][leaf index]+=1;
    // Increment bin in image histogram corresponding to the leaf node
  end
end

Algorithm 2: Bag of Words Random Forest Algorithm.

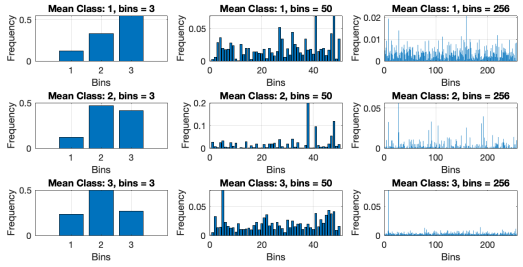

Figure 10: Normalised class means for different classes using k-means.

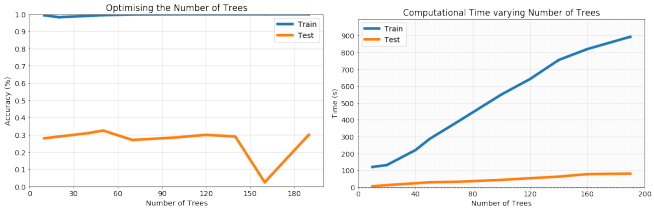

Figure 11: Variation in the accuracy with respect to the number of the trees (left). Variation in the computational time (right).

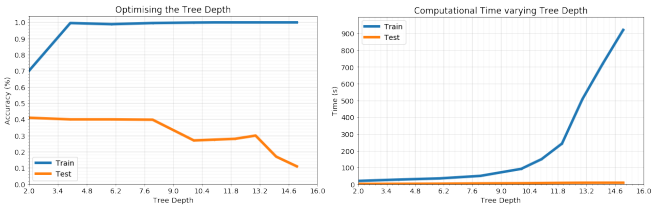

Figure 12: Variation in the accuracy with respect to the depth of the tree (left). Variation in the computational time (right).

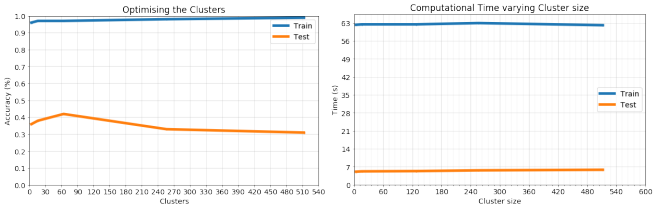

Figure 13: Variation in the accuracy with respect to the k-means cluster size (left). Variation in the computational time (right).

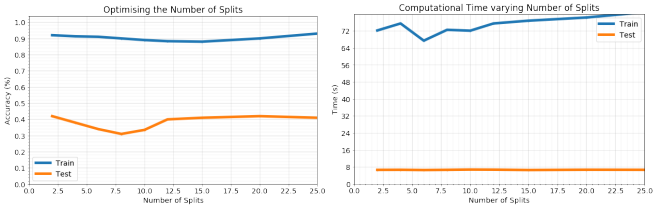

Figure 14: Variation in the accuracy with respect to the randomness between the trees (left). Variation in the computational time (right).