Electronics and Computer Science Faculty of Engineering and Physical Sciences University of Southampton
COMP3204 Coursework 3: Scene Recognition

Tang Ke (kt3n17), Manqi Dong (md8g17), Lynn Wang (lw5n15) and Daniel Queiros (damq1g17)

30th Dec 2020

1 Introduction

The predominant aim of this coursework was to experiment with writing classifiers to work on labelled sets of images, using increasingly sophisticated techniques such as nearest neighbour classification, quantized local feature bags and vector machines, while comparing their accuracy and efficacy in making classification predictions on unlabelled sets of images.

2 Run #1: K-Nearest-Neighbours and Tiny Images

2.1 Implementation

For this initial run, we used the tiny image feature to develop a simple k-nearest-neighbour classifier, utilising the KNNAnnotator class. To compartmentalize our code, we wrote a separate class to represent a TinyImage object, using the FeatureExtractor interface to implement the extractFeature method. This class has two responsibilities: initially, an input image is cropped to a square and resized to 16x16 using a ResizeProcessor. Subsequently, the pixel numbers are iteratively normalised, as the image is set to have zero mean and unit length (and each image row is concatenated to pack the pixels into a vector).

2.1.1 Training and Testing

There are two datasets downloaded from the coursework specification: a training and testing set. In the former, all the images are labelled as belonging to one of fifteen categories, while in the latter the images are unlabelled. Therefore in each run, to test our algorithm's accuracy on the unseen data, we split the training set into two parts, so that 75% could be used purely for training purposes, and the remainder used for testing (where the correct labels were known). Ultimately, the three "run.txt" files were created purely using the unlabelled testing dataset.

For this run, in the Run1 class containing the main method, both the training and testing sets were read in. The training set was split using a GroupedRandomSplitter, where 75% was used to train the classifier, and 25% were used for testing its accuracy. Thereafter, a KNNAnnotator was initialized and trained in this class. After the completion of the training, a portion of code ran the classifier to classify the remaining 25% of labeled data, and compare the predictions with the images' actual label to calculate its accuracy. Finally, the images from the unlabeled testing dataset were classified, as the results were written to an output file named "Run1.txt".

2.1.2 Hyperparameters Tuning

In Run #1, the only parameter that required tuning was the K value used for the k-nearest-neighbour algorithm. Through some experimentation, we found that a k value of approximately 16 yielded the best accuracy for our classifier of around 20%, where the accuracy began to tail off afterwards (however the k value used in this region didn't appear to have a significant effect).

3 Run #2: Dense Patch Sampling and Bag of Visual Words

3.1 Implementation

For Run #2, we used the LiblinearAnnotator class to automatically create a set of 15 one-versus-all classifiers, this time using a BagOfVisualWords based on fixed size densely-sampled pixel patches. As not much code was required in making use of the OpenIMAJ API, we wrote a single class fr this run. Firstly an extractor method was implemented for sampling 8x8 patches, with a step size of 4 in both the x and y directions. For every patch, the pixel numbers were concatenated into a vector and then wrapped into an LocalFeatures object along with the patches' locations. For each image passed to the extractor method, a list containing all the LocalFeatures objects were returned.

In order to learn a "vocabulary" of features extracted from all images, a HardAssigner was implemented, which first stored the feature vectors of all patches from all images into a vocabulary list. The assigner then used this vocabulary to execute k-mean clustering, clustering those features into different classes according to the number of clusters. Lastly we created an inner class called BOVWExtractor, making use of the HardAssigner and a BlockSpatialAggregator to aggregate spatial histograms computed from the bag of visual words. The result was normalised, and returned as a feature vector.

For our training method, we used LiblinearAnnotator with the extractor mentioned above. Similarly to Run #1, the training set was split so that 75% of the images were used for training and the remaining 25% for testing. The unlabeled testing set was then classified, and the results thereof stored in an output file named "run2.txt".

3.1.1 Hyperparameters Tuning

In this run, there were primarily three parameters that required tuning: the patch size, the step size, and the number of clusters. To determine the optimal value for each parameter we made it the unique variable (tweaking it independently), tested the accuracy three times to calculate a mean average, and thus selected the values which yielded the highest average accuracy. We found that the cluster size did not have as much effect on the predictive accuracy as the other two parameters, though noticeably peaked at around 550.

Our final output file, for which we were able to achieve an accuracy of 50%, was created via a run with the optimal values found for all three parameters, given in the table below:

Variable	Optimal Value
patch size	8
step size	6
cluster size	550

4 Run #3: Pyramid Dense SIFT

4.1 Implementation

For Run #3, we took some inspiration from OpenIMAJ tutorial 12, using dense SIFT features, a homogeneous kernel map, and a Baynesian annotation model. According to this academic paper[1], SIFT is a robust mechanism for image matching, retaining discontinuities; furthermore, the spatial model accommodates matching objects located in different sections of a scene, akin to non-local means methods.

Similarly to Run #1, to better compartmentalise our code, we create a separate Record class in addition to our main Run3 class. This Record class implements the Identifiable and ImageProvider classes, wrapping an input image into a Record object containing the image id and the corresponding FImage object; this made it easier to keep images tied to their respective ids. In the main Run3 class, both of the input datasets were read in via loadData() and, as with Run #1 and #2, stored in VFSGroupDataset objects. The convertDataset method was then called to read in the portion of data to train on, wrapping each FImage as a Record object, and replacing the

ListDataset<Record> and its corresponding category name into a new MapBackedDataset (as this data structure supports more functions than the more primitive VFSGroupDataset).

Due to using the PyramidDenseSIFT class, unlike in Run #2, we didn't need to write our own custom implementation for image processing; using the OpenIMAJ class, we could easily analyse FImages to obtain Byte key points directly. However, similarly to Run #2, a HardAssigner was still implemented in order to learn the "vocabulary" of features from sampled images. We then applied a trainQuantiser method, where all images were retrieved from Record objects and analysed by a PyramidDenseSIFT object. The analysis result for each image was obtained via the getByteKeypoints method, using a threshold of 0.005, and these results were stored in a LocalFeatureList called 'allKeys'. Using data in allKeys, we computed the k-means clustering. As with Run #2, we also implemented an inner extractor class using the FeatureExtractor interface (though this time passing Records as second parameters rather than FImages). Using the HardAssigner passed into the constructor, the extractor could aggregate histograms from the bag of visual words, and then apply normalization before returning a resultant feature vector.

In our training method used to train the input data, a PyramidDenseSIFT object called pdsift was initialised to extract features from images, before the training dataset was converted to a MapBackedDataset. As previously, the training set was split with 75% used for training purposes and the remaining 25% for testing. Thereafter, a HardAssigner object assigner was initialised, passing in a sample set containing 30% of training data and the pdsift object for clustering proposes. Finally, a feature extractor was initialised, passing in the assigner and pdsift; unlike in the previous runs, the extractor was wrapped in a HomogeneousKernelMap object in order to transform the data into a compact linear representation apt for the annotator. With all of the aforementioned objects initialised, a NaiveBayesAnnotator could thus be used to train the classifier on the training data. We utilized the same testing method from Run #1 and #2, which simply used the 25% portion of labeled data to demonstrate the accuracy of our algorithm.

4.1.1 Hyperparameters Tuning

The three parameters that required tuning for this run, having a bearing on the predictive accuracy, were the bin size for feature extraction, the step size for the dense SIFT pyramid, and the cluster size for the k-means clustering.

Similarly to Run #2, our final output file, for which we were able to achieve an accuracy of 63%, was created via a run with the optimal values found for all three parameters, given in the table below:

Variable	Optimal Value
patch size	5
step size	7
cluster size	100

5 Conclusion

In conclusion, we found that our run #3 classifier was by far the most effectively in terms of predictive accuracy due to its increased complexity, with a combination of state-of-the-art techniques used. Conversely, our basic KNN classifier in run #1 was computationally fast due to its simplicity, but as expected achieved a much lower predictive accuracy of only 20%. It was clear (as can be seen on our graphs below) that parameters such as patch size and step size had a much greater bearing on predictive accuracy than k value and cluster size.

The most appropriate solution to use ergo depends on the application at hand and size of data set being processed (the effect of a large data set on processing time being particularly noticeable during run #2). We appreciated the effectiveness of cutting-edge techniques such as SIFT feature detection and Naive Bayes classification, while also becoming cognizant of the wide array of techniques and ways they can be chained together.

6 Team Member Contributions

Due to the impact of COVID-19, working together as a team was a unique challenge. As three of us team members (Ke, Manqi and Linying) live in the same student accommodation, the Run #1 and Run #3 tasks were distributed amongst us, whilst Run #2 was primarily assigned to Daniel, and the report was written by all of the team members. We jointly agree that each team member made a fair contribution to this coursework.

7 Appendix

Figure 1 - Average accuracy with increasing k-value

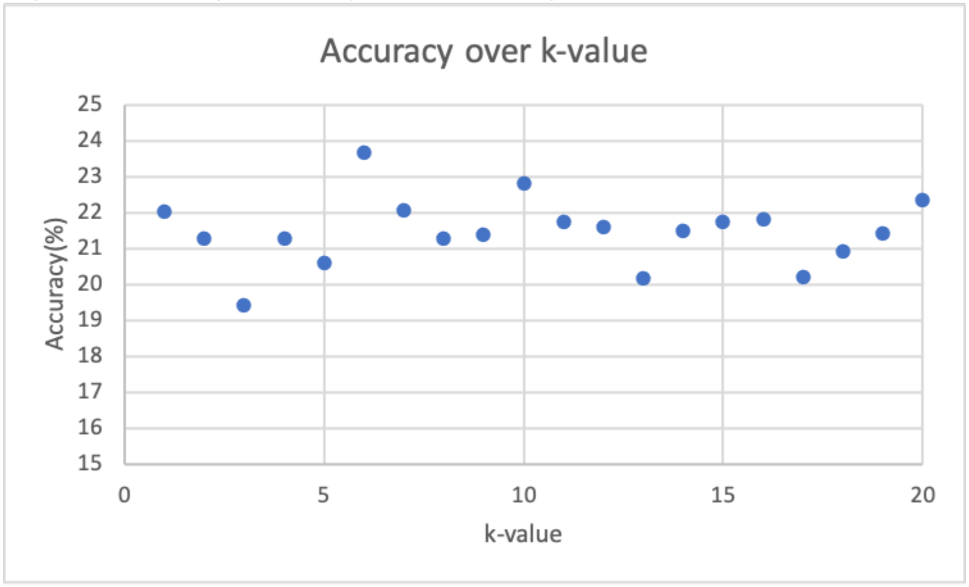


Figure 2 - Average accuracy with increasing patch size

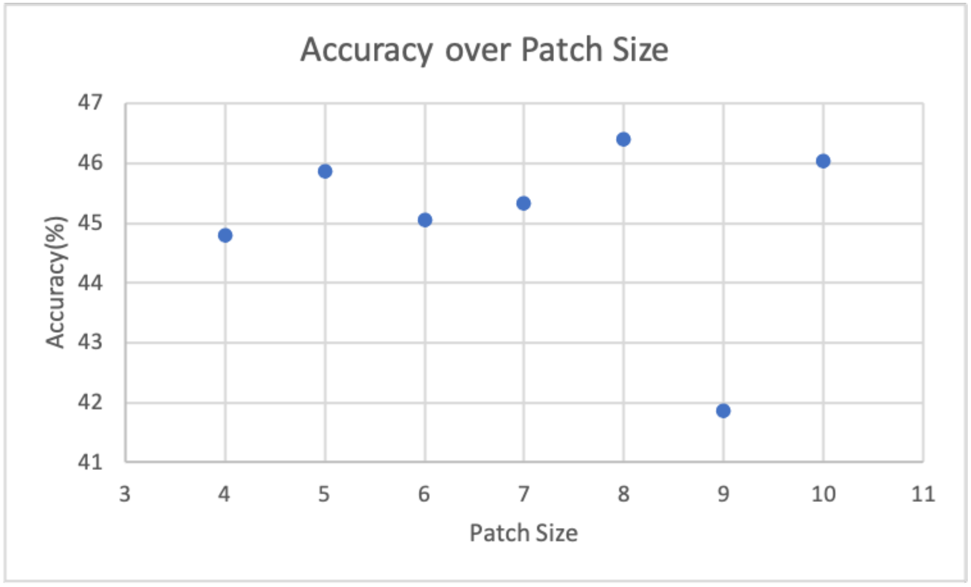


Figure 3 - Average accuracy with increasing step size

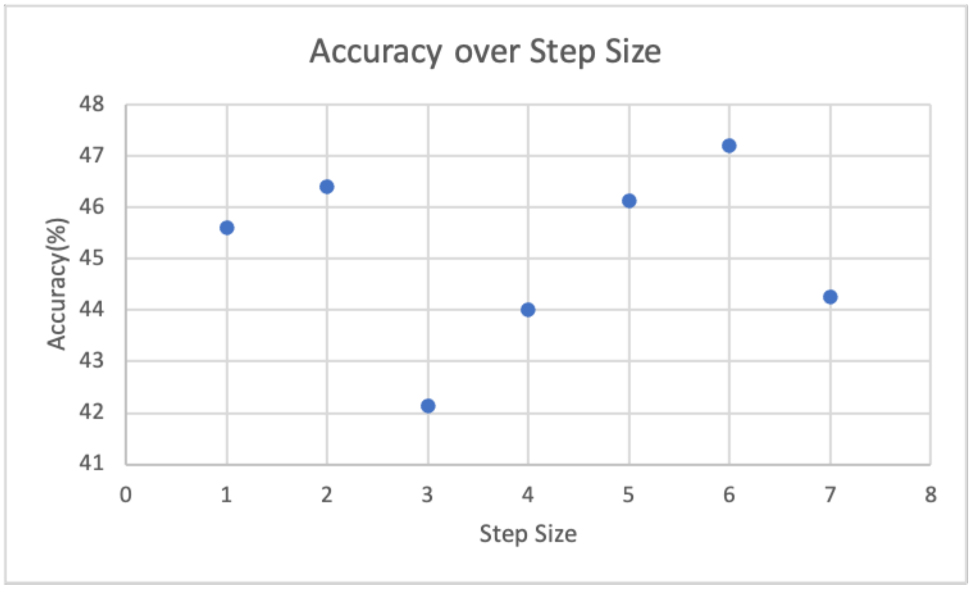
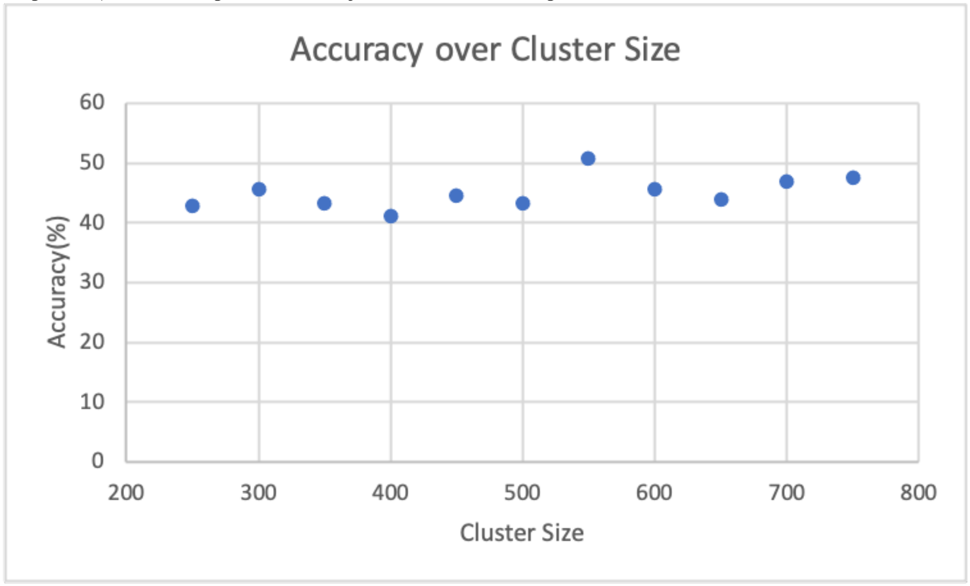


Figure 4 - Average accuracy with increasing cluster size



References

[1] Ce Liu et al. "SIFT Flow: Dense Correspondence Across Different Scenes". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5304 LNCS. PART 3. 2008, pp. 28–42. ISBN: 3540886893. DOI: 10.1007/978-3-540-88690-7-3.