COMP6223 Computer Vision
Coursework 3 - Scene Recognition (Team 1)

Cian Raychaudhuri chr1g17 | Aran Mcconnell ajm2g17 | Yue Shuen, Kew ysk2a15 | Nur Afiqah, Binti Kamarudin nabk1g17

1. Run 1

Run 1 uses a simple approach using k-nearest neighbor to classify an unknown input to a trained model using a 'tiny image' representation of images. This is essentially an image that is resized to a fixed number of pixels. As suggested in the instructions, a size of 16 x 16 was used as the tiny image. This was then packed into a vector and given to the model. These vectors were used to 'train' the model and can be thought of as each image being plotted on a multidimensional graph. When confronted with an unknown input, the model compares the input against every past input and calculates the euclidean distance which is a measure of how different the input is to each known point. It then finds a number of the closest points to the input by ordering the list from closest to furthest and uses k of these to vote on what the unknown input is. In this way, we can classify images.

1.1. Training

For training this model, the data was split into 80% training and 20% test-validation sets. This allows us to train the model on 80 percent of the data and then validate the model using the remaining 20% to indicate the accuracy of the model on unknown data to the model. The data was split using a function from the sklearn library that randomly selects images from the labelled data-set [4].

1.2. Tuning the model parameters

To tune the parameter k of the model, a range of values for k were tested. This was achieved by creating multiple models with each value of k in a range defined at the top of the code. Every odd value was then selected from this list and used to create models trained with the same randomly selected data and add them to a list. This list was then used to run the validation set through the models to get accuracy values for the models with varying k. This gave accuracies for each value of k, however, due to the fact that the data was chosen randomly, the accuracy on any given run did not represent the true average accuracy. To overcome this, the process was run 1000 times, and the final average accuracy was calculated as the mean of accuracies over all the runs.

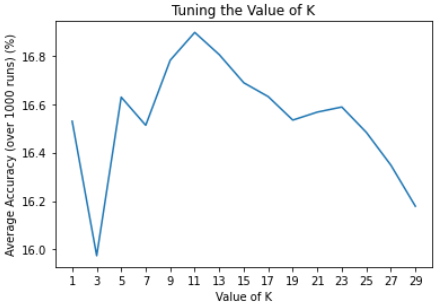


Figure 1: Average accuracy against value of k over 1000 runs.

The output of the script can be seen in Figure 1. It is clear that 11 is the most optimised the model can be, giving an accuracy of approximately 16.9%. This is a very low accuracy, however this was expected as there was no feature extraction involved, so the model relied on the intensities of the same area of pixels being similar across all images in that class. This is unlikely to be true across all of the images in the class. But this is still an improvement on an algorithm that 'guesses' as this would yield an accuracy of approximately 6.7% as there are 15 classes so random choices can be expected to guess correctly 1 in 15 times. Hence, this algorithm provides an improvement on recognition opposed to guessing with no information. Changing the dimensions of the tiny image were also investigated but with limited change to the accuracy so will not be discussed here.

2. Run 2

In run 2, dense sampling and Bag-of-visual-words (BoVW) was used as the feature extraction method. Using the BovW as input to k-Means clustering, the vocabulary–a set of vectors that represented the dataset–was constructed. Each feature vector in the BoVW was then mapped to the closest vocabulary vector, and the frequency of feature vec-

tors across the vocabulary was computed to generate a histogram per image. The histograms were then used as inputs to train an ensemble of 15 one-vs-all linear classifiers for each class.

2.1. Training

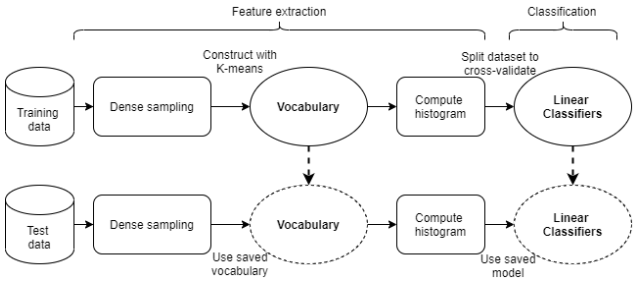


Figure 2: Workflow of experiment for run 2

The experiment, designed as in Figure 2, was split into four stages: dense sampling, constructing vocabulary, computing histogram, and classification. In dense sampling, the image was sampled using overlapping grids across the image in both directions. The vectors in each grid was taken as a feature vector, flattened, and mean-centred.

Using k-Means clustering, the vocabulary representing the dataset was constructed. The vectors in the vocabulary are the centroids of each cluster. Following the methodology in [1], centroids containing less than a minimum amount of samples were removed from the vocabulary to avoid obtaining sparse features. This threshold was set as a minimum percentage of the total number of feature vectors. Using these centroids, each image was then mapped to a histogram using Euclidean distance. The histogram represents the frequency of each vocabulary vector appearing in an image.

For training, the dataset was split into train and validation test set. A stratified 80:20 split was used to return a randomised split that preserved the class distribution. The original training data consisted of 1500 samples with uniform distribution across classes, hence the validation set consisted of 300 samples, with 20 per class. A grid search was also used to get the most fitting regularisation parameter, C, for the LinearSVC model [4] used.

2.2. Tuning the model parameters

The most important hyperparameter for this model was the vocabulary size, k i.e. the number of clusters of k-Means algorithm. The vectors in the vocabulary describe the most important features in the images.

To tune k, the classifier was trained and evaluated on features extracted using different k ranging between $[100, 2000]$. The results of tuning k is shown in Figure 3.

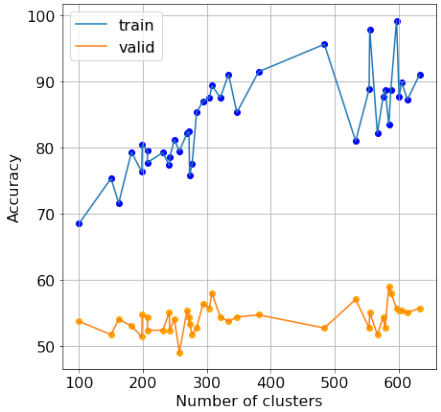


Figure 3: Plot of training and validation loss with varying values of k. The k values are computed after removing clusters with less samples than the minimum.

Due to the minimum threshold set on the cluster population, the actual vocabulary size was equal to or smaller than the parameter set for the k-Means model. The training accuracy improved as the vocabulary size increased. The validation accuracy showed a similar upward trend, though at a much smaller scale. The results proved that smaller vocabulary size led to underfitting, as evidenced by the low training accuracy. The model did not display overfitting due to the threshold set on the minimum cluster population–as the k-Means algorithm parameter was increased, the resulting vocabulary size stopped increasing after a point. Finally, $k = 1000$ was chosen, as its validation accuracy was the highest at 59%. The training accuracy was 83.5% and the actual vocabulary size was 585.

Other hyperparameters that could be tuned were the size of the sampled patches and the interval between patches. Some preliminary investigations were made in this respect, however there was no observable difference after using varying sizes and intervals: $[(8, 4), (16, 4), (16, 8), (32, 8)]$. Finally due to time constraints, the parameters $(11, 10)$ were used, using [1] as guidance.

2.3. Results

The performance of the classifier on the validation dataset was evaluated by calculating the precision and recall per class (Table 1), and also qualitatively by manually looking at the images and their predicted labels. This model performed best on the 'Forest' class, with both recall and precision being the highest overall. This was likely due to the class being more distinctive compared to the rest. In contrast, classes with similar features showed underwhelming performance. Most notably the indoor scenes, such as Bedroom and LivingRoom, had poor precision and recall. The results in [1] had also shown that the vocabulary vectors representing these classes were remarkably similar.

Class	Precision	Recall
bedroom	0.35	0.4667
OpenCountry	0.55	0.6111
kitchen	0.55	0.55
Street	0.7	0.5833
Suburb	0.8	0.6154
TallBuilding	0.7	0.6364
Highway	0.8	0.6667
industrial	0.35	0.6364
livingroom	0.4	0.5
store	0.45	0.5625
Insidecity	0.4	0.5714
Mountain	0.5	0.5
Office	0.65	0.5652
Coast	0.75	0.5556
Forest	0.9	0.75

Table 1: Classifier precision and recall per class, on validation set.

3. Run 3

In the third run, image representations based on BoVW using 7 different feature descriptors were experimented with. It follows 3 steps: feature detection, feature description and codebook generation. Generated histograms from codebook generation is the input to train 4 different classifiers. Apart from that, Transfer Learning as feature extractor too was explored.

3.1. Training

In the third run, a number of different feature extraction techniques were investigated, in order to compare their performance. The workflow for Run 3 is similar to that of Run 2, as shown in Figure 2, except instead of dense sampling, different approaches to extract image descriptors were used. As a starting point a pipeline was setup to take a set of descriptors produced by any feature extraction method, and generate a bag of visual code-words from it using K-Means (with 200 clusters). This would then be used to create histograms for each image and these histograms would be used as the input to train a number of different classifiers. Classifiers used were; NuSVC (Support Vector Classifier), GaussianNB (Naive Bayes), Random Forest, and Gradient Boosting. To validate each feature an 80:20 train test split was carried out the data, and the model was trained. Then the model was used to predict the remaining 20%.

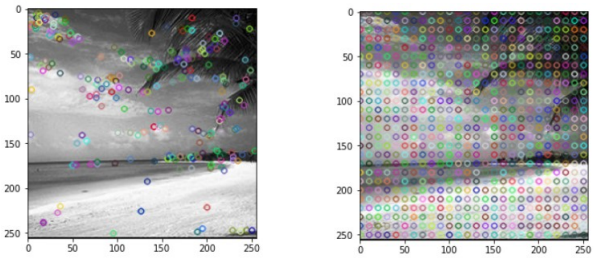
Apart from mentioned approaches, Transfer Learning is also experimented with in two ways – Transfer Learning as feature extractor model and Transfer Learning with final fully-connected layer modified to the dataset having 15 classes as output.

Techniques investigated were, SIFT, DenseSIFT with Gaussian Pyramid, DenseSIFT with Spatial Pooling (PHOW), KAZE and ORB. These follows the general pipeline mentioned before, feature detection, feature description and codebook generation using KMeans with 200 clusters.

Scale-invariant Feature Transform (SIFT) [3] is a feature detection algorithm for detecting and describing local features in an image. DenseSIFT generates a SIFT descriptor at each location of an image controlled by a step size. Figure 4 shows the difference between both algorithms. Gaussian Pyramid is a multi-scale signal (image) representation. An image is subsampled with Gaussian blur and scaled down for a number of levels. DenseSIFT with Gaussian Pyramid is simply DenseSIFT on each level for an image. Spatial Pooling, otherwise known as Pyramid of Histogram Words (PHOW) allows spatial information in addition to BoVW. For PHOW, an image is subsampled to multiple levels and pooled for each level (Figure 5) and PHOW is the concatenated histograms of visual words of all subregions at different levels. For each region, DenseSIFT is applied to each subregion.

ORB local features is a feature extraction technique designed as a more efficient (and unpatented) alternative to SURF and SIFT. It uses a fusion of the FAST and BRIEF methods for keypoint detection and descriptor generation, but then modifies these methods greatly to make the ORB much more rotation invariant than BRIEF. KAZE local features is another feature extraction technique that lies somewhere between SURF AND SIFT in terms of computational expense. KAZE uses multiscale processing techniques, and claims to have performance on par with SIFT for some applications.

In a deep neural network, the first few layers work as feature extractor for the image to be classified by the final few fully connected layers for a classification task. For that reason, Transfer Learning in this section is explored as a feature extractor model with extracted inputs as to linear classifiers and also fully connected layer for classification. The pretrained model chosen for this approach was ResNet50 [2].



(a) SIFT (b) DenseSIFT

Figure 4: SIFT and DenseSIFT Descriptors for an Image.

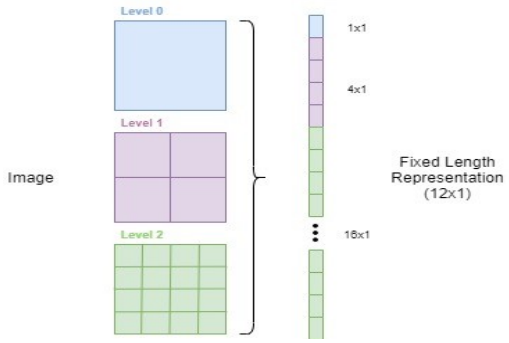


Figure 5: Pyramid Histogram of Words

Feature Description	Classifiers			
	SVC	GaussianNB	Random Forest	GradientBoosting
SIFT	21.81	44.01	26.36	45.99
DenseSIFT	66.46	59.60	47.47	30.70
DSIFT + Gaussian Pooling	65.85	55.60	50.70	31.71
DSIFT + PHOW	71.92	50.30	49.49	30.30
KAZE1	5.72	43.33	23.33	42.23
KAZE2	14.03	39.41	23.88	39.98
ORB1	7.62	22.76	16.38	19.48
ORB2	7.63	23.94	16.47	19.66
ORB3	7.62	23.37	15.39	18.55
ResNet50	**82.63**	79.00	71.31	56.76

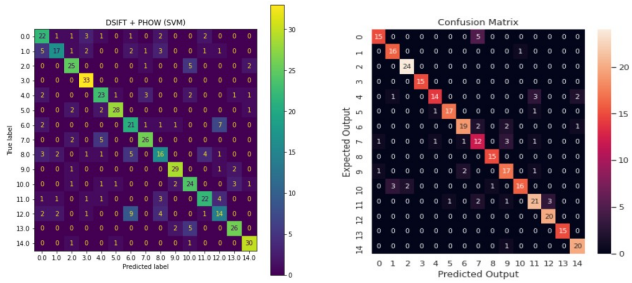
Table 2: Performance of SIFT, KAZE, ORB, ResNet50 feature descriptors on SVM, GNB, RF and GB classifiers.

3.2. Tuning Model Parameters

Parameters for various feature descriptors is set as such: DenseSIFT – step size 10, Gaussian Pyramid – 4 levels, PHOW – 3 levels. For KAZE, the threshold parameter was varied between 0.0001, and 0.0005. Any higher that this would lead to 0 features found in some images. For ORB the edge threshold parameter was varied between 5 and 10, and the fast threshold between 10 and 20. The results of these different ORB and KAZE matchers is detailed in Table 2. With Transfer Learning, the pretrained Resnet50 model is retrained using Cross Entropy Loss, Adam optimiser with a learning rate of 0.001 and a Exponential Learning Rate Scheduler which decays the learning rate by gamma of 0.1 every 7 epochs for 100 epochs.

For all parameter variations tested, ORB is by far the worst technique, only reaching a maximum of around 24% accuracy. The KAZE local features method then is significantly better than ORB, at least when using certain classifiers. KAZE using a threshold of 0.0001 managed to reach 43.3% accuracy using the GaussianNB classifier. In terms of classifiers using KAZE and ORB, SVC and Random Forest give consistently terrible results, with GaussianNB consistently being quite good, and GradientBoostingClassifier beating it in some cases only.

In contrast, all DenseSIFT related approaches obtained best model performance with SVC as evidenced in Table 2. ResNet50 as feature extractor achieved 82.63% of overall accuracy on validation set. Finally when training ResNet50 model with two fully connected layer, the model obtained the best model performance, achieving 86.32% in accuracy as shown in Figure 6b.



(a) PHOW+SVM (b) ResNet50

Figure 6: Model Performance on feature descriptors using DSIFT+PHOW and ResNet50 on SVC.

4. Team Contributions

Table 3 details the contributions of each member towards this coursework.

Contributions	Scripts	Report
Cian Raychaudhuri	Graphs/Run1/, Run1.py, Run1_TUNE.py, utils.py	Section 1
Aran Mcconnell	run3-kaze-orb_files/, Run3-kaze-orb.py	SIFT, KAZE and ORB related parts in Section 3
Yue Shuen, Kew	run3-esther/, Run3-DSIFT.py, Run3-DL.py	DenseSIFT and Transfer Learning related parts in Section 3
Nur Afiqah, Binti Kamarudin	run2/, run2.py, run2_count.py,	Section 2

Table 3: Breakdown of contributions within the team

References

[1] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 524–531 vol. 2, 2005.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[3] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.

[4] 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.