COMP3204 Coursework 3
Scene Recognition

Joshua Thorne (jtt1g18)
Reece Buckle (rtb1c17)
Turgut Torun (tt6u17)
Max-Morby Roy (mmr1u16)

University of Southampton

January 2021
Department of Electronics and Computer Science

1 Run #1

1.1 Description

The k-nearest-neighbour classifier decomposes an image into a "tiny image" feature and then clusters the features.

1.2 Process

1. Instantiate a **TinyImageKNNClassifier(size: 16, k: 18)** object:
   (a) **Size: 16** represents the size of the "tiny image" that will be extracted from each image.
   (b) **K: 18** represents how many clusters are used in the nearest neighbour classifier.
2. Instantiate a **TinyExtractor()** object:
   (a) This contains an **extractFeature()** method which crops an image to a square and resizes to a predetermined size. The rows of pixels are then concatenated together to return a feature vector.
3. Instantiate a **KNNAnnotator<>** with the "tiny image" feature extractor, Euclidean distance metric and specified number of clusters to train the model.
   (a) The **KNNAnnotator** assigns each image to the closest cluster and that cluster is given the label of the majority of the images.
   (b) To produce a label for a test image, the closest cluster to this image is determined and the label of the cluster is assigned to the image.

1.3 Results

Mean centring and normalisation of the feature vectors were initially implemented, however this caused a negligible impact and occasionally reduced the prediction accuracy of the model. **testTinyImageKNNClassifier()** in App.java showcases how the Tiny Image KNN Classifier was extensively tested. Because this approach is very quick, it was run on a range of values of K (number of clusters) (between 1 and 50) 10 times such that the average accuracy and error rate could be recorded. This was found to be **K: 18, Accuracy: 21.91%, Error: 78.09%.** Though that being said, results varied very little between 19-21% across all values of K. See Appendix 5.1 for results.

2 Run #2

2.1 Description

The Linear Classifier makes use of a **PatchExtractor** class, BOVW **FeatureExtractor** class, **LiblinearAnnotator** (OpenImaj library) and a **HardAssigner** method.

2.2 Process

1. Instantiate a **LinearClassifier(step: 4, size: 8, clusters: 500)** object:
   (a) **Step: 4** represents the number of steps between each patch in an image i.e. to sample 4 pixels in each x and y direction.
   (b) **Size: 8** represents a patch size of 8x8 pixels which are extracted from each image.
   (c) **Clusters: 500** represents the amount of clustering (typically between 50 and 500).
2. Instantiate a **PatchExtractor()** object:
   (a) This contains an **extract()** method which returns a list of local features in the form of a patch and its location within the image.
   (b) A **RectangleSampler** instance is utilised to extract the region of interest for the patches and to obtain the x,y coordinates.
   (c) Each patch is mean-centred and normalized and then concatenated into a single dimension.
3. Utilise the **HardAssigner** interface within the **trainQuantiser()** method:
   (a) **trainQuantiser()** utilises the **PatchExtractor** to retrieve the patch features from each image as it iterates through the training dataset.
   (b) A sample of 10 features are taken from each image.
   (c) **trainQuantiser()** then performs K-Means clustering on the extracted features in order to learn the vocabulary.
4. Instantiates the Bag of Visual Words (BOVW) extractor which implements the **FeatureExtractor** class:
   (a) This extracts hard-assigned BOVW representations of each image which are then matched and placed within the same identifier/cluster.
5. Use the **LiblinearAnnotator<>** to construct and train the Linear Classifier (utilising the BOWV Extractor):
   (a) This is then trained with the loaded images (and in the case of accuracy testing, a split training set).

2.3 Results

In order to test the accuracy of our Linear Classifier without bias, we split the training input set 90/10 and 80/20 and trained the classifier on a variety of different clusters size (5-500). The range of accuracy returned between 44-53%. It was observed that 50 clusters produced a greater accuracy than 500 clusters. We tried decreasing the step size, however at 2 or below, we ran out of heap space (over 8gb); a step size of 3 showed no significant improvement.

A best-case accuracy 53.3% was recorded with the following parameters **Step size: 4, bin size: 8, clusters: 50** with a 90/10 split. Using the Caltech101 approach was much faster as less images were used, but returned an overall accuracy of 33%. As an alternative to this approach, we extracted 10 random samples from each image which significantly reduced the running time for the cost of 3%. Finally, to investigate the importance of mean centring and normalising each patch before returning the feature, the absence of this returned a very poor accuracy of 25.3%. See Appendix 5.2 for results.

3 Run #3

For task 3, we implemented 3 different strategies: a Naive Bayes Classifier, a Dense SIFT Classifier (both utilising OpenImaj) and a Convolutional Neural Network strategy utilising the 'Deeplearning4j' library. The Dense SIFT Classifier resulted in the best prediction accuracy of 80%. As such this is used for the prediction of task 3.

3.1 Naive Bayes Classifier

3.1.1 Description

The Naive Bayes Classifier makes use of the PHOW Feature Extractor, NaiveBayesAnnotator (OpenImaj library) and a HardAssigner method.

3.1.2 Process

1. Initiate a **NaiveBayesClassifier(Step size: 4, Bin size: 8, E Threshold: 0.015f, Clusters: 500)**.
   (a) **Step size: 4** the number of pixels the patch is moved in the x and y direction.
   (b) **Bin size: 8** the bin size which will be extracted from the image.
   (c) **E Threshold: 0.015f** the threshold value in the form of Dense SIFT with byte vectors.
   (d) **Clusters: 500** represents the amount of clustering.
2. Utilise the HardAssigner interface within the **trainQuantiser()** method:
   (a) Use the denseSIFT extractor to retrieve patch features from the training dataset.
   (b) Perform K-Means clustering on the extracted features in order to learn the vocabulary.
3. A **PHOWExtractor**<> object is instantiated:
   (a) This class extracts the hard-assigned PHOW representations of each image which are then matched and placed within the same identifier/cluster. Bag of Visual Words assigner is used in the implementation.
4. **NaiveBayesAnnotator**<> is used to train and construct the classifier.

3.1.3 Results

Opposed to the Linear Classifier, the Naive Bayes Classifier did return a higher average accuracy. A best-case accuracy of 61.3% was achieved using the following parameters: **Step size: 2, Bin size: 8, E Threshold: 0.015f, Clusters: 25** with a 90/10 split. Increasing the cluster size beyond 25 had diminishing results. However, lowering the step size from 4 to 2 did improve accuracy but below 2, we ran out of heap size.

3.2 Pyramid Histogram of Words (PHOW) with Dense SIFT Features

3.2.1 Description

The PHOW method extracts Dense SIFT features from images to be quantised into visual words that can be used to build a spatial histogram of the word counts.

3.2.2 Process

1. Instantiate a **DenseSIFTClassifier(clusters: 600, siftStep: 5, siftFeatures: 2000)** object:
   (a) **Clusters: 600** the vocabulary of the Bag of Visual Words.
   (b) **SiftStep: 5** the step size of the sampling window for the Dense SIFT.
   (c) **SiftFeatures: 2000** the number of SIFT features used to produce the Bag of Visual Words through clustering.
2. Instantiate a **PyramidDenseSIFT**<> object:
   (a) The extractor is fed a **DenseSIFT** object with bin sizes of 7.
3. Obtain a **HardAssigner**<> object using the **trainQuantiser()** method:
   (a) This method takes a sample of the images and the Pyramid SIFT extractor.

(b) The SIFT extractor obtains all the key-points for every image.
(c) 2000 random key-points are then chosen.
(d) The key-points are then clustered to form the vocabulary for the Bag of Visual Words.
(e) A **HardAssigner**<> that assigns data to a visual word is returned.

4. A **PHOWExtractor**<> object is then instantiated which produces the spatial histograms:
   (a) The object takes the hard assigner and SIFT extractor.
   (b) The SIFT features are extracted from the image and transformed into words using the hard assigner.
   (c) The histogram is then created using the Bag of Visual Words and a **PyramidSpatialAggregator**.
5. A **HomogeneousKernalMap** object is instantiated and wraps the **PHOWExtractor**<> to transform the data into a compact linear representation.
6. A **LiblinearAnnotator**<> object is instantiated to create 15 one-vs-all classifiers.

3.2.3 Results

Training and testing images were obtained from a 90/10 split on the dataset. The **ClassificationEvaluator**<> class was utilised to obtain a detailed report of the model's performance.

At first a **BlockSpatialAggregator** was implemented and the **HomogeneousKernelMap** was not used. This resulted in a classifier with a prediction accuracy of 57%. After changing the aggregator to a **PyramidSpatialAggregator** and implementing a **HomogeneousKernelMap**, the accuracy increased to 76.7%.

The clusters, step size for the dense SIFT and the number of SIFT features were then fine-tuned to create a model with an accuracy of 80% (see Appendix 5.4).

3.3 Convolutional Neural Network

To create the **CNN**, the **DeepLearning4j** library was utilised. The **CNN** was comprised of 34 layers made up of:

- **Convolution Layers** for feature extraction
- **Batch Normalization Layers** for standardisation
- **Activation Layers** using **Relu** to set negative values to 0
- **Pooling Layers** using average pooling to extract more important features
- **Drop-out Layers** to drop out nodes in each layer during training to prevent over-fitting
- **An Output Layer** for classifying the data using a **SoftMax** activation function

The model was trained using an epoch of 50. **Stochastic Gradient Descent** was utilised for back propagation and **Adam** was utilised for learning (with a learning rate of 0.01).

3.3.1 Process

1. Initially the train-test split of the data was created from the images cropped to (224×224), vectorised and random samples were evenly distributed and separated into batches of size 10. This data was then normalised to allow the network to infer its weights more easily.
2. Following this, the CNN was created with input shape $(channels \times height \times width) = (1 \times 224 \times 224)$ and the output layer were set to have nodes representing each class where the number of output nodes was 15.
3. The network was then trained using 50 epochs in order to tune the network weights on the training split.
4. The train and test splits were then evaluated.

3.3.2 Results

This approach was disappointing as we only managed to achieve a best accuracy of 39%. In particular the network was able to evaluate scenes such as Coasts, Forests, Streets, Kitchens and Stores quite well but fell short on other classes. These results could have been improved if the input data had been augmented by rotating and transforming some of the images, however given the length of time needed to train a neural network this was not possible due to hardware constraints. Furthermore, a better result could have been achieved by attaching this network to the end of a pre-trained frozen network such as RESNet.

4 Contributions

Max:

- Worked on Task 1, Task 2 and Task 3 (Convolutional Neural Network)
- Fine tuned the parameters of Tiny Image KNN Classifier for finding the best value of K (0 to 50)
- Ran analysis tests for the CNN
- Wrote helper methods for splitting, testing, classifying and outputting data
- Writing report (section 3)

Josh:

- Worked on Task 1, Task 2 and Task 3 (Dense Sift Classifier)
- Helped with debugging errors for Task 1 and Task 2 to ensure correct functionality
- Refactored code for readability and usability
- Writing report (all sections)

Reece:

- Worked on Task 2
- Ran majority of analysis tests for Task 1, Task 2 and Task 3 (all approaches)
- Refactored code for readability and usability
- Coordinating workload & handling GIT source control
- Writing report (all sections)

Turqut:

- Worked on Task 3 (Naive Bayes Classifier)
- Writing report (section 3)

5 Appendix

5.1 Appendix A: Results for Tiny Image KNN Classifier

K	Accuracy	Error
1	0.20405	0.79595
2	0.19185	0.80815
3	0.1885	0.8115
4	0.19725	0.80275
5	0.2052	0.7948
6	0.20975	0.79025
7	0.20975	0.79025
8	0.2143	0.7857
9	0.21265	0.78735
10	0.21655	0.78345
11	0.2189	0.7811
12	0.2125	0.7875
13	0.2069	0.7931
14	0.21255	0.78745
15	0.2099	0.7901
16	0.21215	0.78785
17	0.21425	0.78575
18	0.2191	0.7809
19	0.2084	0.7916
20	0.20885	0.79115
21	0.208	0.792
22	0.20365	0.79635
23	0.2094	0.7906
24	0.2123	0.7877
25	0.20455	0.79545
26	0.21075	0.78925
27	0.21215	0.78785
28	0.20005	0.79995
29	0.20275	0.79725
30	0.2046	0.7954
31	0.2114	0.7886
32	0.20875	0.79125
33	0.2132	0.7868
34	0.21525	0.78475
35	0.20965	0.79035
36	0.21085	0.78915
37	0.20195	0.79805
38	0.20545	0.79455
39	0.2091	0.7909
40	0.2154	0.7846
41	0.20565	0.79435
42	0.2081	0.7919
43	0.2056	0.7944
44	0.20345	0.79655
45	0.20855	0.79145
46	0.20685	0.79315
47	0.20495	0.79505
48	0.1972	0.8028
49	0.2091	0.7909

5.2 Appendix B: Results for Linear Classifier

Training Split	Cluster Size	Accuracy	Error Rate	Duration (Minutes)
90/10	5	34.7	75.3	0.63
90/10	50	53.3	46.7	2.62
90/10	500	44.7	55.3	29.78
80/20	20	46.7	53.3	1.17
80/20	50	49.3	50.7	2.40
80/20	100	45.3	54.7	5.32
80/20	500	45.0	55.0	27.72
70/30	500	44.0	56.0	24.50
80/20[1]	500	42.3	57.7	7.32
80/20[2]	500	25.3	74.7	29.65
15/15[3]	500	33.3	66.7	6.45
80/20[4]	50	45.7	54.3	4.40

Key

[1] Extracting 10 random samples from each image (reduced running time for the cost of 3%)

[2] Excluding mean centring or normalisation of each extracted patch

[3] Caltech 101 approach: sampling a random 15 images for training and testing from each set

[4] Decreasing step size to 3 shows no change

5.3 Appendix C: Results for Naive Bayes Classifier

Training Split	Step Size	Bin Size	Cluster Size	Accuracy	Error Rate	Duration (Minutes)
80/20	4	8	5	44.0	56.0	7.5
80/20	4	8	25	58.7	41.3	8.5
80/20	4	8	100	59.3	40.7	13.2
80/20	2	4	25	61.0	39.0	14.3
90/10	2	8	25	61.3	38.7	17.3

5.4 Appendix D: Results for Dense SIFT Classifier

Training Split	Cluster Size	Sift Step	Sift Features	Accuracy	Error Rate	Duration (Minutes)
90/10	300	5	1000	76.7	23.3	10.5
90/10	300	3	1000	74.7	25.3	20.8
90/10	600	5	2000	80.0	20.0	19.1
90/10	300	5	2000	78.0	22.0	10.8

5.5 Appendix E: Confusion Matrix from CNN

Coast	Forest	Highway	Inseidecity	Mountain	Office	OpenCountry	Street	Suburb	TallBuilding	Bedroom	Industrial	Kitchen	LivingRoom	Store	
19	-	1	-	-	-	-	-	-	-	-	-	-	-	-	Coast
-	18	-	-	-	-	-	1	-	-	-	-	-	-	1	Forest
14	-	2	-	-	-	-	1	-	-	-	2	-	-	1	Highway
-	-	-	1	-	1	-	4	3	2	-	2	6	-	1	Inseidecity
9	2	1	-	6	-	2	-	-	-	-	-	-	-	-	Mountain
-	-	-	1	-	1	-	-	-	-	-	2	16	-	-	Office
17	-	1	-	-	-	1	1	-	-	-	-	-	-	-	OpenCountry
-	-	-	-	-	-	-	17	2	-	-	1	-	-	-	Street
-	2	1	4	-	-	1	9	1	-	-	2	-	-	-	Suburb
-	-	-	1	-	2	-	2	-	-	1	3	11	-	-	TallBuilding
-	1	1	-	-	2	-	-	-	-	1	7	8	-	-	Bedroom
-	-	-	-	-	1	-	2	-	-	1	10	-	-	6	Industrial
-	-	-	-	-	1	-	-	-	-	1	-	17	-	1	Kitchen
-	-	-	-	-	-	-	-	-	-	3	1	16	-	-	LivingRoom
-	2	-	-	-	-	-	4	-	-	-	-	-	-	14	Store