Computer Vision Coursework 3
Scene Recognition

Lucas Noyau - ln3g14 - 26921936
Jean-Luc Mourey - jm32g15 - 28231589
December 15, 2017

1 Introduction

This coursework required the development of three different classifiers. The first uses a simple k-nearest-neighbour algorithm. The second uses a set of linear classifiers. The third classifier was given no specifications, and it was left to us to create the best possible classifier.

2 Classifiers

All classifiers used in this project extend from the abstract class `MyClassifier`. It was created in order to limit the amount of repeated code in the other classifiers, as well as give a framework for us to work from. The content of the class is as follows:

- Class variables are `trainingData` and `testingData`. These hold the datasets that are used for training and testing respectively.
- Constructors, one with no parameters, the other uses a String of the path to get the datasets from.
- `go()` method is used to train and test the classifier on the class variable datasets, with the result being saved by `printResults(ArrayList)`.
- `printResults(ArrayList)` takes a list of the predicted classes in String form, and both prints it to `System.out` and to a file `output.txt`.
- `classify(groupedDataset)` takes a dataset and returns the ArrayList of String that is used by `printResults`. It does this by iterating over all the images in the dataset and calling `classify(FImage)` on each of them.
- `train(GroupedDataset)` takes a dataset but doesn't return anything. Each classifier has a different method for training, therefore this method is abstract.

- `classifyFImage` is another abstract method, as image classification depends on the classifier. This method return a String of the predicted class name.

Therefore the only variations in the classifiers listed below are in the `train(GroupedDataset)` and `classify(FImage)` methods, although other methods are used in order to remove duplicate code and make the code easier to read.

2.1 Run 1: A Simple k-nearest-neighbour Classifier

- `Run1(String, String)`, the constructor. Calls the constructor of its parent class by passing `trainingDataPath` and `testingDataPath`.
- `void train(GroupedDataset<String,ListDataset<FImage>,FImage>)` generates the feature vectors of each image in the training dataset and saves them in a `DoubleNearestNeighboursExact` object. Calls the method below.
- `extractFeature(GroupedDataset<String, ListDataset<FImage>, FImage>)` iterates over every image in every class, calling `extractFeature(FImage)` on each. Returns the list of all the extracted feature vectors.
- `extractFeature(FImage)`, which crops the image to a square and resizes it to 16*16 pixels. Returns the new image's feature vector values.
- `@Override String classify(FImage)` which takes an image, gets its nearest neighbours, and returns the predicted class, which is generated by counting the classes of the neighbours to find the most likely class.

2.2 Run 2: A Set of Linear Classifiers

- `Run2(String, String)`, the constructor. Calls the constructor of its parent class like in run 1.
- `train(GroupedDataset<String,ListDataset<FImage>,FImage>)` trains an assigner with a vocabulary created from the set of training images. It then generates the linear classifier from a generated feature extractor which is trained with the data.
- `trainQuantiser(Dataset<FImage>)` iterates through every image in the dataset and adds all the values from the feature vectors of each image's rows to a float array list (List¡float[]¿, not ArrayList¡float¿)

- `extractFeature(FImage)` scans across the image, generates feature vectors, and returns the feature list.
- `Extractor` a class that implements `FeatureExtractor`. uses bag of words to extract features from the given image. is called by the `train()` method.
- `classify(FImage)` is used to pass the image against the `LiblinearAnnotator.classify()` method, returning the predicted class as a String.

We left run2 working for over an hour (2013 MacBook Pro) on a subset of the data. It was getting stuck at `FloatKMeans.cluster(float[][])`. Reducing the number of clusters from 500 to 100, and images per class to 1, allowed us to get past this.

We then tried increasing the number of images per class, at 5 the program took much longer to run, but we got results. We attempted to increase this number again, but the run was unsuccessful.

2.3 Run 3: Developing The Best Possible Classifier

For run 3, we wanted to experiment with Neural Networks. We decided to attempt to use TensorFlow to generate a classifier. To do this, we followed an online tutorial [1]. This lead us to use a TenserFlow Github project's program [2].

Although we had planned to then attempt to create our own Neural Net classifier. Due to time constraints, however, we were unable to get this system working.

To be clear: we did not write the TensorFlow implementation that we used to generate the results for this run. We did write the supporting Python code that allowed us to iterate over the testing data and print it out in the required format. We therefore cannot take credit for the accuracy (or lack thereof) of this run.

3 Individual Contributions

This coursework was done nearly in its entirety during working meetings, in a pseudo-buddy-coding style. Lucas was writing and running the code while Jean-Luc did research, looking up the appropriate methods in the vast OpenImaj library, particular approaches to some problems, and helping with debugging.

References

[1] Lin JungHsuan: Create a simple image classifier using Tensorflow, medium.com/@linjunghsuan/create-a-simple-image-classifier-using-tensorflow-a7061635984a

[2] This is some text github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/image_retraining/retrain.py