COMP3204 Computer Vision

Electronics and Computer Science

Faculty of Physical Sciences and Engineering

University of Southampton

Scene Recognition

William Davies and Matt Andrew

Wednesday, 14 December 2016

Contents

[Classifier 1 3](#_Toc469326307)

[Classifier 2 4](#_Toc469326308)

[Classifier 3 5](#_Toc469326309)

[Contribution 6](#_Toc469326310)

# Training Set Expansion

# In an attempt to improve the quality of our predictions, we decided to expand the size of the training set to get the classifiers more images to learn from. To do this, we added copies of each image that had been rotated a few degrees in each direction to the set. This both increased the size of the training data, and helped to improve rotation invariance of the classifiers. Performance of all three classifiers was noticably improved after implementing this feature.

# Classifier 1

Classifier 1 is a simple TinyImage based classifier. To train the classifier, a TinyImage feature extractor is created, with a Tiny Image size of 16x16. Each image is then resized down to this 16x16 square, and this smaller version image is extracted as the feature for this image.

It is much quicker to compare the pixel values of these 16x16 images to classify them than to compare the whole full size image.

Once all these features are created, they are used to construct a k-nearest neighbour search, which will take in an image, and then give the Euclidean distances from the given image vector to all the image vectors used to create the knn search.

To classify an image, we pass it into this knn search, and get the closest neighbour to it calculated by the search method. As we already know the class of every image within the search, we can predict the target image’s class to be the same as its nearest neighbour’s class.

This is a very simple classifier, which gives it very quick run time over under 15 seconds (classification is multithreaded), but this does result in a rather low prediction accuracy, averaging around 28% correct guesses. This is with a training set of 1200 and a validation set of 225.

# Classifier 2

Classifier 2 is a linear classifier using a BagOfVisualWords feature extractor and a liblinear annotator.

A hard assigner is created to use in the feature extractor, to construct this assigner, we train a quantiser. 10 images from each group of the training dataset are randomly selected to provide the images for the quantiser vocabulary. The quantiser constructs this vocabulary by taking the selected images, extract local features from them using a rectangle sampler with a step of 8 pixels and a patch size of 12x12 pixels, taking a random 50 of these samples, and clustering them using a K means tree ensemble.

The assigner containing this vocabulary is used to create the BoVW extractor. This allows the local features of an image to be matched against the assigner, and a BoVW representation of the image to be created, which is then used to classify the image against the given possible classes. A BlockSpatialAggregator is used to represent this as a aggregate vector containing both feature and spatial information.

A liblinear annotator is created using this extractor, which will create a series of one vs all classifiers using all the structures just built. This annotator will be trained on the full trainingdataset, and then using the knowledge learned from this, is used to annotate and classify the testing set of images.

This classifier takes around 5-10 minutes on my laptop, and gives an accuracy of around 60%, an improvement over classifier 1.

# Classifier 3

Classifier 3 uses a feature extraction technique known as Pyramid Histogram of Words (PHOW) to retrieve suitable features from the images, along with a Bayes annotator to assign the images to a class.

We create an assigner that uses dense SIFT with a step size of 4 and 8 bins to extract SIFT features on a uniform grid across the image to extract features from the images in the training data. These are then used by an ensemble of KD-trees to perform an approximate k means clustering into 25 clusters, with the produced centroids being used as the assigner when classifying images. We use an approximate k means method as opposed to an exact one to drastically cut down operation time and improve execution performance. While we would ideally prefer to use a larger number of clusters in the range of 500, we found that this had an extremely large negative effect on run time to the point that we have not been able to verify the improvement to prediction accuracy it may bring

When an image is being classified, we use a PHOW extractor that uses dense SIFT with the same parameters as the assigner. The image is broken up into four blocks, and a bag of visual words uses the assigner to assign each feature to a visual word. These assignments are then used to create a histogram of each quadrant of the image which are appended together and normalised. This normalised histogram is then used by a naive Bayes annotator to predict the class of the image, returning the class with the maximum likelihood.

While we would’ve like to use pyramid dense SIFT to extract features and improve scale invariance, the fact that we are using multithreading to improve the execution time requires creating a new instance in each thread due to dense SIFT not being thread safe. Doing this with pyramid dense SIFT therefore has an even greater memory cost, and led to running out of memory during execution.

This classifier takes an extended period of time to run due to the slow nature of extracting the image features using dense SIFT: performing SIFT across the whole image instead of only on points of image increases the processing cost significantly. Doing this over every image in the training and test sets therefore takes a long time, which unfortunately cannot be solved with multithreading due to dense SIFT not being thread safe and therefore requiring a new instance in each thread which would in turn introduce a huge memory cost.

This classifier performs very well, producing a prediction accuracy of around 78% correct classifications.

# Contribution

Contribution to the project was an even 50:50 split between both members of the team.