|  |  |
| --- | --- |
| COMP6223 Computer Vision (MSc) Prof. Mark Nixon, Dr. Jonathon Hare  **Coursework 3:**  **Scene Recognition** | 15/12/2017  Ganiyu Ibraheem, gai1u17  Philipp Seybold, ps1c17 |

This report investigates three different classifiers and their associated feature extractor for scene recognition. It is structured into three sections for each run of the classifiers on given test data. While the first two runs are briefly described in their approach and implementation, the third run will additionally include more detailed explanations on its used methods in comparisons with the first two classifiers. To run each script, it must be in the same directory as the folders “training” and “testing” holding the data.

1. K-Nearest-Neighbour Classifier with “Tiny Image” Feature

Approach

A k-nearest-neighbours (kNN) algorithm identifies the *k* nearest neighbours in a *n*-dimensional space of a given vector *v*. The neighbours consist of labelled training data each transformed into a vector and arranged in some data structure that fits the space dimension best. The class that *v* has the most number of nearest neighbours to is assigned to *v*. An effective way to facilitate multidimensional search is a *n*-dimensional tree that partitions the space in a number of partitions so subtrees can be left out if the current *k*th distance is smaller than the subtree itself. This allows *O(log(n))* complex searches in best and *O(n)* in worst cases. Drawbacks are its complexity depending on the chosen data structure for large datasets, possibly dominating classes when they are represented more often in the training data and performance is dependent on the number of dimensions. Advantageous on the other hand are the zero cost for the learning process and no model concepts need to be respected.

As for the tiny-image feature, one resizes the image to a fixed resolution, suggested is 16x16, and transforms them into a vector by concatenating each image row. It is also suggested to change the vectors to have zero mean and unit length for improved results. When classifying test data, the tiny-image feature is simply compared to the training data through the kNN algorithm. As biggest drawbacks of a plain feature like this one is that it discards the high frequency image content and is not shift invariant.

Implementation

Firstly, each image was resized to 16x16, then concatenated into a feature vector (1-by-256) which was then transformed into a normalised histogram with 16 dimensions. Thereafter, a kNN was trained using the image histograms as inputs and their labels as targets. For the learning process, an optimal was calculated by cross-validation which often lied around and a *n*-d tree represented the data structure. Predictions were performed as usual: the feature vector was extracted from the test image and then classified by the trained kNN. More details can be found in the code comments.

1. Linear Classifiers with Bag-of-Visual-Words Feature

Approach

The classifier will be a set of linear support vector machines (SVM), acting as one-vs-all classifiers. The decisions made by these classifiers can be expressed as “is v in class A or non-A”. For a number of *n* (here: 15) different classes the classification is repeated *n* times after what *v* is assigned to the class with the highest probability of *v* belonging to it. During the training-phase each of the SVMs learns for one label its local feature composition by a set of labelled images from which firstly, its features are extracted and then mapped to a learned cluster (see second paragraph). Afterwards, it is able to identify the chance that a given image belongs to its class or not. Their advantage over kNN classifiers is their ability to detect dimensions of features that are less relevant and would otherwise down weight a decision. They can also be used for non-linear classification utilising the kernel trick implicitly mapping their inputs into high-dimensional feature spaces.

A Bag-of-visual words (BoVW) feature is a bag data structure containing clustered visual words (features) of a “vocabulary” for their image. The vocabulary contains all possible local descriptors that a training image can contain. The clusters are computed by K-Means clustering over a number of sampled features (“visual words”) from the vocabulary. These samples consist of fixed size densely-sampled pixel patches that were be either transformed into SIFT features (which will be done in section 3 of this report) or any other local feature that might fit well for the given context. After an image is decomposed into local descriptors, it is counted how many of these features fall into each cluster in the visual word vocabulary, called vector quantisation, and noted in the BoVW. From that distribution, the likelihood for a category is computed by the classifier. Advantages and drawbacks depend on the chosen local feature, but by a BoVW feature container, the accuracy already increases considerably as well as the computation time through the K-Means clustering.

Implementation

Following the suggestions in the assignment, these methods and parameters were used for the second run: The patches’ size was set to 8x8, they were sampled every 4 pixels in the x and y direction and transformed into normalised histograms (like in section 1) for each image. Together with the images’ labels form they the visual words dictionary. Afterwards, the clusters were computed using K-Means with 10 random features per training image and the number of clusters set to . Then, for each image, the Bag-of-Words is created by mapping the image features to the closest cluster mean and counting for each cluster how many features were mapped onto it. These BoVWs were subsequently used as input together with their images’ labels as target for training the support vector machines (SVM). As for the training, for each classification problem did its SVM receive all BoVWs for that label as positive examples and the rest of the input as negatives. The SVMs used a standard regularisation factor of 1. When it came to the prediction, the image’s inputted Bag-of-Words was given to every label classifier which then computed the probability of that image belonging to its class. The highest probable label was returned as result. Further details can be found in the code comments.

1. CNNs with SVMs and Dense-SIFT Features

Approach

### Classifier

Linear SVMs represented through the convolutional layers and the of the CNN.

During the learning phase a Stochastic gradient descent optimizer is used, that stochastically approximates the gradient descent method to minimise the error function of the classifier. It updates incrementally and frequently to give an immediate insight about the performance of the model. The noisy update process can allow the model to avoid local minima although updating the model so frequently is more computationally expensive so it is taking significantly longer to train models on large datasets. In this use case with a rather small dataset it provided a considerable optimisation.

### Feature Extractor

Is done in the first densely-connected layer with 128 units and a rectifier activation function in the CNN

SIFT (Scale-invariant feature transform ) Features -> detects and describes local features.

DEF: Now an orientation is assigned to each keypoint to achieve invariance to image rotation. A neigbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. (It is weighted by gradient magnitude and gaussian-weighted circular window with \sigma equal to 1.5 times the scale of keypoint. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with same location and scale, but different directions. It contribute to stability of matching. Now keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is devided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

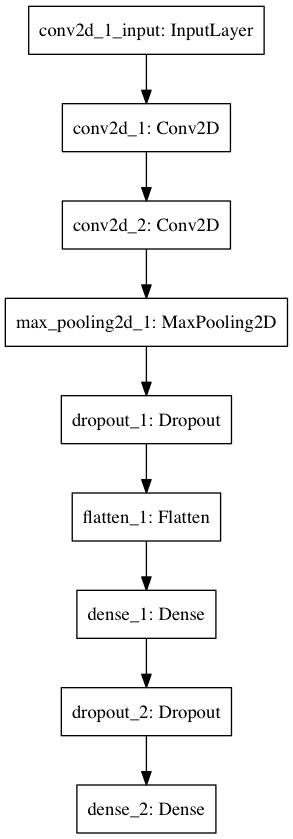
* Detection for normal SIFT by David Lowe's algorithm
* We used Dense SIFT which computes a SIFT descriptor at every location of the image (based on histograms of gradients) like in section 2 (a patch every 4 pixels)

Besides the advantage of well representing an image, SIFT features might bring along complexity problems when facing large resolution images and thus making it necessary to use k-means to learn a million clusters in 128 dimensions (!) from 10’s millions of features.

Compare to the features of the previous two runs, this feature is to an extend invariant to transformations (translation, rotation and scaling) and also robust against brightness variations, noise and smaller geometrical deformations of higher-order. They therefore deliver a great representation of images features, especially for comparing them.

### Convolutional Neural Network (CNN)

CNNs are deep, feed-forward artificial neural networks with focus on analysing images. They come handy for Classification problems with their ability to incorporate feature extraction, classification by SMVs and their training as well as other mappings into one system. Each weight adjusting layer acts as linear classifier, they are able to do non-linear classification. Due to a large offer of frameworks for the major programming languages, they are easy to implement.

Implementation

Python OpenCV, chainer, keras, sklearn

1. Convolutional Layer 3x3 kernel receiving a 256x256x1 image as input and outputs a 32 dimensional descriptor using a rectifier as activation function
2. A second 3x3 kernel with a rectifier as activation function that outputs a 64 dimensional descriptor
3. Pooling Layer downsizing the input by 2 / 50%
4. A Dropout to prevent overfitting randomly set 50% of the input units to 0
5. Flattens the inputs to vectors
6. Densely-connected layer with 128 units and a rectifier activation function
7. Another Dropout with 50% dropout-rate
8. Final Densely-connected output layer with 15 units (the number of different classes/labels) and a softmax-activation function that computes for each class the probability that the Input belongs to it

CNN Parameters:

1. Loss function= Categorical Crossentropy (For multi-class classification problems)
2. Optimiser = Stochastic gradient descent optimizer, with learning rate=0.01, decay=0.00001 momentum=0.9 and enabled Nesterov momentum
3. Metrics / Performance indicator = Accuracy
4. Instructions in order to run the code
5. Appendix A: Misclassification Error for k-neighbours of kNN

