COMP6223 Computer Vision (MSc) 15/12/2017
Prof. Mark Nixon, Dr. Jonathon Hare

**Coursework 3:** Ganiyu Ibraheem, gai1u17
**Scene Recognition** 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.

# 1. K-Nearest-Neighbour Classifier with "Tiny Image" Feature

## 1.1 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 a data structure that fits the space dimension best. The class that $v$ has the highest 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 to the subtree itself. This allows $O(log(n))$ complex searches in the best and $O(n)$ in the worst case. Drawbacks on the one hand are firstly 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 secondly performance is dependent on the number of dimensions. Advantageous on the other hand is 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.

## 1.2 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 $k$ was calculated by cross-validation, which often lies around $k = 23$ 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.

# 2. Set of Linear Classifiers with Bag-of-Visual-Words Feature

## 2.1 Approach

The classifier consists of 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). Secondly, 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 weight a decision down. 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 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. Based on 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.

## 2.2 Implementation

Following the suggestions in the assignment these methods and parameters were used for the second run: The patches' size was set to 16x16, then they were sampled every 16 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 and the number of clusters set to $k = 500$ from a 10th of the number of the images' features. 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, each classification problem's SVM receives 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 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 the result. Further details can be found in the code comments.

# 3. Deep Convolutional Neural Network

## 3.1 Approach

### 3.1.1 Classifier

This approach also uses a form of SVMs that are represented through "neurons" in the convolutional layers in the CNN, but the CNN as a whole *is* the classifier. 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 gives 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.

### 3.1.2 Feature Extraction

As mentioned in the previous section a good approach would have been using a SIFT (Scale-invariant feature transform) feature to describe and compare key points in the images. In the original algorithm by David Lowe, interest points are computed and then transformed while this approach uses Dense-SIF, selecting from an equal grid of patches for each image instead. First an orientation is assigned to each key point around which a neighbourhood is taken depending on a selected scale. This is done to compute the gradient magnitude and direction in that region. Afterwards, a normalised histogram is created containing 36 bins that cover 360 degrees to safe the orientation from values over a threshold of 80%, or the highest peak if none found. The orientation is weighted by gradient magnitude and Gaussian-weighted circular window with a $\sigma$ equal to *1.5* times the scale of the key point. This will result in key points with same location and scale, but with differing directions which contributes to stable matching. Finally, the key point descriptor is created by selecting a 16x16 patch around the key point. It is then divided into 16 sub-blocks of a 4x4 size. For each sub-block, an 8 bin orientation histogram is created. From that follows a total of 128 bin values that represent the feature vector [http://opencv-python-tutroals.readthedocs.io]. Compared to the features extractors of the previous two runs, this one is to an extend invariant to transformations (translation, rotation and scaling) and also robust against illumination changes, noise and smaller geometrical deformations of higher-order. They therefore deliver a great representation of images features, especially for comparing them. But besides these advantages, SIFT features might bring along complexity problems when facing large resolution images and databases, thus making it necessary to learn a million clusters in 128 dimensions (!) from 10's millions of features in the K-Means algorithm.

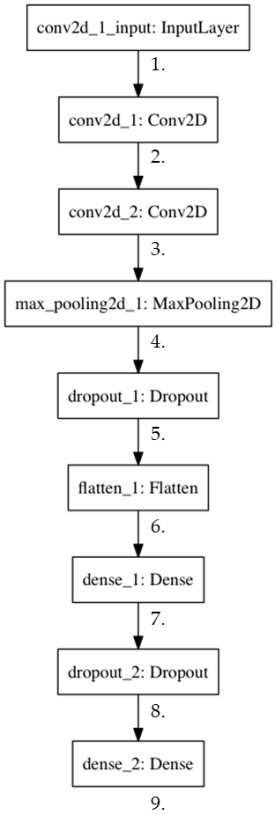
**But,** in order to utilise the functionalities of a CNN, no explicit feature extractor was implemented. This design decision was made during the work on run 3, hence the time spend on implementing Dense-SIFT is represented as the first paragraph of this chapter. The local descriptors are localised in the convolutional layer and transformed in the densely-connected layer into a 128 dimensional vector through a rectifier activation function.

### 3.1.3 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 extractors, classifiers (SMVs) and their training as well as other mappings into one system. Each weight adjusting layer acts as a linear classifier, hence multiple layers of those allow to perform non-linear classification. Due to a large offer of frameworks for the major programming languages, they are easy to implement.

## 3.2 Implementation

Like the first runs, does this implementation also follow the order to first extract, pre-process (resizing to 256x256) and structure the training data to then learn a model making same labelled images recognizable as such. The implementation of the classifiers and feature extractors is hidden in the layers of the CNN and can be found in the Keras package documentation. The layers of the CNN (shown by the illustration on the right) were setup with the following parameters:




| #L | Function |
|---|---|
| 1. | The Input Layer holding a 256x256x1 sized image (grey-scale) |
| 2. | Convolutional Layer 3x3 kernel outputting a 32 dimensional vector using a rectifier as activation function |
| 3. | A second 3x3 kernel with a rectifier as activation function that outputs a 64 dimensional vector |
| 4. | Pooling Layer downsizing the input by 50% |
| 5. | A Dropout to prevent overfitting randomly set 50% of the input units to 0 |
| 6. | Flattens the inputs to vectors |
| 7. | Densely-Connected Layer with 128 units and a rectifier activation function |
| 8. | Another Dropout with 50% dropout-rate |
| 9. | 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 |

The parameters for training the CNN layers:

- Loss function: Categorical Cross Entropy (For multi-class classification problems)
- Optimiser: Stochastic Gradient Descent optimiser with
  - » *Learning rate = 0.01*
  - » *Decay = 0.00001*
  - » *Momentum = 0.9*
  - » *Nesterov momentum = enabled*
- Metrics / Performance indicator: Accuracy

# 4. Instructions in order to run the code

The following packages need to be installed in the Python environment to execute the Python scripts for all three runs: OpenCV, chainer, Keras, sklearn. To run each script, it must be in the same directory as the folders "training" and "testing" that hold the data.

# 5. Appendix A: Misclassification Error for k-neighbours of kNN

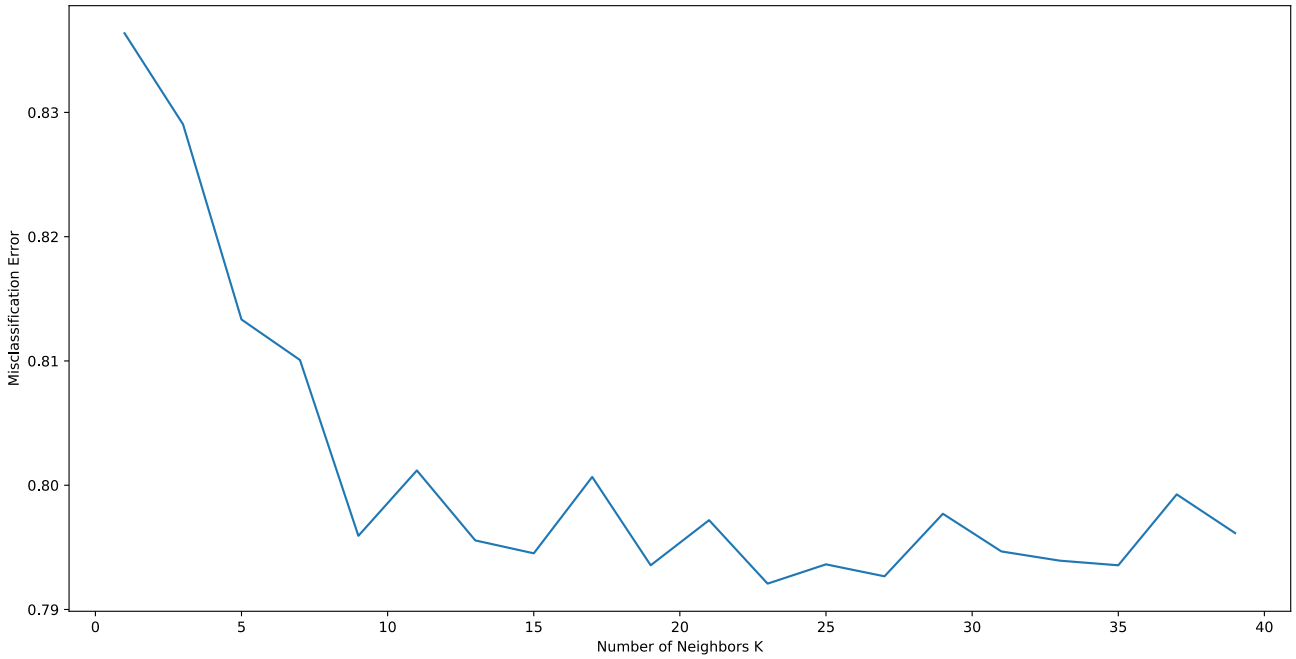



***Figure 1:*** *The line plot shows the misclassification error for a selected number of neighbours k for the kNN-algorithm used in the first run. The accuracy for* $k = 23$ *seems to be the highest.*