The Neural-SIFT Feature Descriptor for Visual Vocabulary Object Recognition

Sybren Jansen

December 21, 2014

Master Thesis

Artificial Intelligence

University of Groningen, The Netherlands

First supervisor:
Dr. Marco Wiering (Artificial Intelligence, University of Groningen)

Second supervisor:
MSc. Amirhosein Shantia (Artificial Intelligence, University of Groningen)

university of groningen / faculty of mathematics and natural sciences / artificial intelligence

Abstract

In computer vision, one area of research which receives a lot of attention is recognizing the semantic content of an image. It's a challenging problem where varying pose, occlusion, scale and differing light conditions affect the ease of recognition. A common approach is to extract local feature descriptors from images and attach object class labels to them, but choosing the best type of feature to use is still an open problem. Some use deep learning methods to learn to create features during training. Others apply local image descriptors to extract features from an image. In most cases these algorithms show good performance, however, the downside of these type of algorithms is that they are not trainable by design. After training there is no feedback loop to update the type of features to extract, while there possibly could be room for improvement.

In this thesis, a continuous deep neural network feedback system is proposed, which consists of an adaptive neural network feature descriptor, the bag of visual words approach, and a neural classifier. Two initialization methods for the neural network feature descriptor were compared, one where it was trained on the popular Scale Invariant Feature Transform (SIFT) descriptor output, and one where it was randomly initialized. After initial training, the system propagates the classification error from the neural network classifier through the entire pipeline, updating not only the classifier itself, but also the type of features to extract. The feature descriptor, before and after additional training, was also applied using a support vector machine (SVM) classifier to test for generalizability.

Results show that for both initialization methods the feedback system increased accuracy substantially when regular training was not able to increase it any further. The proposed neural-SIFT feature descriptor performs better than the SIFT descriptor itself even with limited number of training instances. Initializing on an existing feature descriptor is beneficial when not a lot of training samples are available. However, when there are a lot of training samples available the system is able to construct a well-performing feature descriptor when starting in a random state, solely based on classifier feedback. The improved feature descriptor did not only show improved performance in the setting in which it was trained, but also while using an SVM classifier. However, the improvements were small and were only demonstrated with one other classifier. Therefore, more experiments are needed to get a better grip on the generalizability of the improved descriptor.

Contents

1 Introduction

One of the most challenging problems in computer vision is to recognize the semantic content of an image. This is especially the case in situations where objects vary in pose, where there is occlusion, and where differing light conditions are present. Detecting the contents of an image (e.g., visible objects or scene category) is an important task in image retrieval and robotics, among others.

In image retrieval tasks, a search query is given and images containing this query should be reported back. Such systems could be very useful, for example, in medical diagnosis. Apart from aiding doctors you could take a picture at home of some skin disease you have and run it through a database. Similar photos will show up which tells you what the disease might be and if you should call a doctor or go to the nearest hospital as soon as possible.

For robotics, localization is one of the most fundamental problems. Without this, a robot does not know how to go to a certain place when ordered to do so. This is identical to when we humans do not know how to get home when we're in a deserted place which we do not recognize. Scene recognition can help a robot localize itself [12, 8]. Additionally, a lot of tasks for a robot involve manipulating certain objects (e.g., bringing coffee or finding emergency buttons in residential care homes). Without knowing which object is which, the robot has a hard time performing any of these tasks.

A common approach to object recognition in complex and changing environments is to extract local feature descriptors from images and attach object class labels to them [44]. Given the extracted features from a test image, these are then matched against features from each class. When there are enough matching features for an object class in a test image, that specific class is detected.

Finding the best features is still an open problem in computer vision. Some have used deep learning architectures to learn to create features during training [25, 28]. Other methods include extracting features using fixed algorithms. Such algorithms have shown to give good performance in many applications [2, 10, 17, 26, 36, 40], however, the downside of these type of algorithms is that they are not trainable by design. After training there is no feedback loop from the classifier to the feature extraction stage to update the type of features to extract, while there possibly could be room for improvement.

This thesis proposes a continuous feedback system to improve existing feature descriptors. To create a trainable feature descriptor a fixed algorithm has to be made trainable. For this purpose, trainable feature descriptors like artificial neural networks can be applied and initially trained on such descriptors. The popular Scale Invariant Feature Transform (SIFT) algorithm [30] will be used for this purpose. Based on this trainable network (termed 'neural-SIFT'), the bag of visual words approach and a neural network classifier, a system is proposed which allows for the classification error to be propagated all the way back to the feature extraction network (termed 'full backpropagation'), which in turn tries to improve its feature extraction capabilities.

1.1 Related work

1.1.1 Deep learning

Deep learning architectures (e.g., employed in [20, 28, 46]) have recently become one of to the most common used systems in image classification tasks. One of the most classic architectures for deep learning are artificial neural networks. Generally, deep learning architectures involve modeling high-level abstractions in data by using multiple non-linear transformations. The most common abstraction in an image is the object to be recognized. Deep learning architectures can also model intermediate levels such as the edges, corners or shape of an object. This abstraction level is usually referred to as feature extraction. Based on these extracted features the final abstraction can be made towards the object class.

1.1.2 Feature extraction

A simple approach for object recognition is to use global features like color histograms [42]. However, global features lack the power to distinguish between foreground and background objects. Color histograms also suffer from differing light conditions which causes the performance to drop significantly. Nowadays, a more common approach is the use of local descriptive features. Such a feature could represent a certain shape or curve in the image. The actual content of the feature, however, is not really relevant. According to Tuytelaars and Mikolajczyk, the ideal local feature should be distinctive, invariant, and robust [44]. These properties are closely related, if a feature is more invariant it generally leads to a reduced distinctiveness. If a feature must be more robust, typically some information is disregarded and therefore the feature can become less distinctive.

There has been extensive research proposing different local descriptors. One of the most popular descriptors is the Scale Invariant Feature Transform (SIFT) [30] (see Section 2.2). A comprehensive study comparing multiple local features on differing image transformations showed that the SIFT descriptor performed as one of the best [33]. Typically, SIFT detects salient keypoint regions, which correspond to parts in the image containing relevant information, and extracts a feature vector from each of these regions. Another method is to replace keypoints with a fixed partitioning scheme so that the whole image content is represented. Features can then be extracted from each image patch (e.g., used in [1, 3, 7]). Abdullah et al. compared both schemes and found that they perform similarly well [3].

Some have tried to use machine learning techniques to learn and extract local features from training images. One of these techniques involves using artificial neural networks (ANNs), which have been successfully applied to detect edges [41], corners [43], and other features. These ANNs can also be used as deep learning architectures by themselves when composed of multiple layers, for example, in character recognition [14]. One advantage of using neural networks with several hidden layers is that they can perform any non-linear feature extraction. As a downside, though, neural networks suffer from having no statistical basis, the network will behave like a black box after training [15].

For classification, one approach is to extract features from the test image and compare these with every feature from every known object class. One can imagine that the computation time needed for this approach increases very

rapidly as the number of classes and number of extracted features increases. As with SIFT, a single keypoint results in a 128-dimensional feature vector and depending on the number of scales used, a typical image can contain between 1000-3000 keypoints [30]. Detecting the keypoints at different scales and matching the keypoints consumes a lot of computation time. There are alternatives that show similar performance and are faster in terms of computation. One of them is the Speeded Up Robust Features (SURF) algorithm [6]. This algorithm is more suitable for real time image processing, but it performs less well than the SIFT algorithm regarding accuracy [24].

1.1.3 Bag-of-words

Another type of matching is inspired by the bag-of-words method frequently used in text classification [23, 29]. In text classification, word frequency information is gathered and stored in a histogram. Based on this histogram, a classifier can determine the semantic context of the text. Apparently, there are specific words that have high indicative power for certain contexts. Sivic and Zisserman proposed to use this for the visual domain as well, which has been shown to work surprisingly well in image classification and categorization [2, 10, 17, 26, 34, 36, 37, 40, 47].

Terms like bag-of-keypatches [10] and bag-of-visterms [34] have been used to make the distinction for computer vision applications. The idea is to cluster the extracted features from training images to obtain visual codebooks or visual keywords. As a result, these visual keywords represent similar features. The extracted features from a given test image are then matched to the visual keywords and the frequency of matches per cluster is stored in a histogram. This histogram is then used as input for a classifier.

The counting of matches per cluster is called the hard bag-of-features approach [40], as it is employing a hard assignment scheme: a single feature has only one closest cluster. Other methods using a soft assignment approach to construct a histogram have been introduced. These approaches give weights to multiple clusters which are close to a feature. Weights can be given by ranking nearest neighbors by distance [22] (i.e., the lower the distance, the higher the rank and the higher the weight), or using the distances itself [38, 45].

Histograms have the advantage of simplicity and computational efficiency, but ignore any spatial information in the image, information which could be of potential use. It is then also highly surprising that this method shows such great results, even under challenging real-world conditions including intra-class variations and background clutter [10, 47]. Sivic and Zisserman even showed the strength of this method in object and scene retrieval in videos [40]. Zhang et al. researched the effect of possible 'hints' in background features as some scenario background features are not entirely uncorrelated from the foreground (e.g., cars are usually on a road and a boat is usually found on water) [47]. Despite these correlations, using additional background information does not improve performance, the features extracted from the object itself play the key part in recognition.

The lack of spatial information has been addressed by using so-called spatial pyramids, first introduced by Grauman and Darrell [18] and later adopted to use with the bag of visual words approach by Lazebnik et al. [26]. The idea behind this is to divide the image into multiple regions and create a histogram for each of them. Spatial information can be captured by combining these histograms to form a set of histograms (e.g., by concatenation). This approach can

be applied at different resolutions to create an even richer representation of an image. It is shown that this method can often outperform the single histogram approach [7, 18, 19, 26].

Multiple approaches to clustering the visual keywords have been proposed, some have used k-means clustering [10, 40], some have used Gaussian mixture models (GMMs) [17, 37]. k-means clustering [31] provides a hard assignment scheme, while with GMMs a soft assignment scheme is used, where a single feature can belong to multiple clusters. Abdullah et al. compared multiple soft assignment schemes with the traditional hard assignment scheme [2], their soft assignment approaches perform significantly better than the hard method.

1.1.4 Classification approaches

For the last stage, classification, numerous methods are available. One very popular method in classifying histograms of visual keywords is the Support Vector Machine (SVM) [9] used by [2, 10, 17, 26, 34, 36, 37, 47]. Abdullah et al. also tried k-nearest neighbors (k-nn) as classifier, but showed that SVMs significantly outperform the k-nn method [2]. Another approach is to use neural networks. Egmont-Petersen et al. provides an extensive overview of different types of neural networks used in this scope [15].

1.2 Research questions

The main challenge of this thesis is to create a modular object recognition system which can learn from its mistakes. Based on neural networks and an adaptation of the bag-of-words approach to the visual domain, the system should learn, based on the classification error, to extract 'better' features than the initial local image descriptor. Better features in the sense that these features should be more distinctive, in a way to achieve higher recognition accuracy. To sum up the goal of this thesis, in the conclusion of their review of more than 250 research papers on image processing with neural networks, Egmont-Petersen et al. state:

> *A true challenge is to use ANNs [artificial neural networks] as building blocks in large, adaptive systems consisting of collaborating modules. Such an adaptive system should be able to control each module and propagate feedback from the highest level (e.g., object detection) to the lowest level (e.g., preprocessing). (Egmont-Petersen et al., 2293)*

The focus of this thesis is not on achieving the highest recognition accuracy possible, but rather to improve recognition results by training the feature extraction system based on the classification error. The main question is: Will this learning process be able to improve the feature descriptor in a way that the whole system achieves a higher recognition accuracy? An important follow up question is whether or not this improved feature descriptor will be generalizable to other classification systems (e.g., by using a support vector machine as classifier), or if it is optimized specifically for the system at hand. If this would not be the case the entire error propagation pipeline would need to be applied to each new setting, which would be far from ideal.

Another goal of this thesis is to investigate the role of initializing the feature extraction system on the output given by an existing feature descriptor. As mentioned above, artificial neural networks will be used in this system. Initializing a neural network to an underlying feature extraction function can take

quite some learning time. If the system would be able to learn itself to extract good features when starting in a random state, this would save training time and time investigated in choosing an appropriate feature descriptor to train on.

These objectives lead to the following research questions:

1. Can training the feature descriptor based on the classification error improve recognition results?
   (a) What is the recognition accuracy before and after applying the additional training step?
   (b) Will retraining the vocabulary and classifier help improve results even further?
2. Can the system come up with a good feature descriptor without initializing on an existing one?
   (a) What is the recognition accuracy after applying full backpropagation when starting with a random feature descriptor compared to the accuracy achieved when initializing on an existing feature descriptor?
3. Is the improved feature descriptor generalizable to other recognition systems?
   (a) What is the recognition accuracy based on the original and additionally trained feature descriptor while using an SVM classifier?

1.3 Outline

In Chapter 2, the theoretical background of the techniques used in this thesis are described in detail. It starts off with an introduction to object recognition in general, after which individual techniques are further explained. Chapter 3 reports on the individual stages of the proposed system (Sections 3.1-3.4) and the derived training steps (Section 3.5). The experimental setup, the performance of the system, and a discussion regarding the results are presented in Chapter 4. Chapter 5 concludes this thesis, where the research questions will be answered and possible future work is presented.

2 Theoretical background

Object recognition systems are often designed using the same underlying pipeline. Usually, the first step is to normalize the images to suit the needs of the system. In this preprocessing stage, images can be transformed to an appropriate resolution, noise can be reduced, or color space transformations can be applied, amongst other steps. In the next step, features are extracted in the feature extraction stage. The goal here is to reduce the amount of information present in the image, but still make the extracted information as distinctive for the image as possible. Some systems have an intermediate step transforming or grouping together features to make a final representation of the image. Finally, in the classification stage, a classifier is used which tries to recognize objects based on the extracted features.

In this chapter, the basic techniques used for the proposed system are described in detail. Because an adaptive feature extraction system is required, artificial neural networks, or just neural networks for short, seem to be an ideal tool. The basics of neural networks are described in Section 2.1. These networks can be trained beforehand on a variety of target functions. One example of a target function could be a local image descriptor, which takes a region of the image as input and transforms it to a feature description. As mentioned in Section 1.1, even though the SURF algorithm [6] is computationally less expensive compared to the SIFT algorithm [30], the SIFT algorithm provides better accuracy [24]. The type of target function for neural networks does not change the required computation time, the topology of the network is the determining factor. Therefore, the SIFT keypoint descriptor will be the target function to train on and is described in Section 2.2.

The bag-of-words approach is adapted for the use in the visual domain (Section 2.3). Because no clear 'words' exist in the visual domain, a clustering procedure is used to determine the visual vocabulary of words. Although soft assignment approaches, like Gaussian mixture models (GMMs), have shown to give better results than hard assignment approaches (e.g., k-means clustering) [2], the computational time needed to train GMMs is much higher than the more simple k-means models. GMMs are typically trained using maximum likelihood estimation (MLE) [13], which requires calculating the full covariance matrix for each cluster. When using the SIFT descriptor this translates to a matrix of size 128×128. Perronnin et al. [37] proposed to use diagonal instead of full covariance matrices for two reasons:

(1) any distribution can be approximated by a weighted sum of Gaussians with diagonal covariances; and

(2) the computational costs is much lower for calculating diagonal covariances.

Although this approach is much faster than the traditional one, k-means is still much faster. Given that k-means itself is used as an initialization step for MLE, this becomes even more evident. Because of the poor scalability of

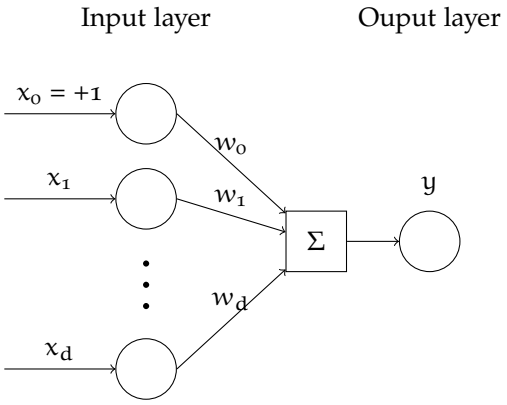


Figure 2.1: The basic perceptron. $x_1, \ldots, x_d$ are the input units, x_0 is the bias unit, which always takes the value +1, y is the output unit, and w_i is the weight from x_i to the output.

Gaussian mixture models, k-means is preferred. k-means is described in more detail in Section 2.4.

Finally, for classification a neural network will be used. Once the error propagation is successful and an improved feature descriptor has been realized, generalizability can be tested. As mentioned earlier, a support vector machine classifier will be utilized for this purpose. A brief explanation of the SVM algorithm is provided in Section 2.5.

2.1 Artificial neural networks

Artificial neural networks take their inspiration from the central nervous systems found in animals (the brain in particular). The brain is a very powerful organ capable of massive parallel processing and is superior in vision, speech recognition, and many other things, when compared to the artificial models currently available. Simulating the workings of the brain can help understand how the brain functions and can possibly lead to very powerful computer systems.

2.1.1 The perceptron

The most basic ANN is the perceptron (see Figure 2.1). It has inputs which can come from any type of source, indicated by $x_i \in \mathbb{R}, i = 1, \ldots, d$. Each input has a corresponding weighted connection $w_i \in \mathbb{R}$ to the output unit y. The output y in the simplest case is the weighted sum of the inputs:

$$y = \sum_{i=1}^{d} (x_i w_i) + w_0 \tag{2.1}$$

x_0 is the bias unit and always takes the value +1. This bias unit makes the model more general. If there was no bias unit and the inputs were all zero the network's output would be zero as well, which might not always be the desired behavior.

The perceptron in its current form can be used to learn linear functions. When $d = 1$, the network can learn a basic line with slope w_1 and intercept w_0.

By adding more inputs the line becomes a plane or a hyperplane and can learn multivariate linear lines.

Apart from regression problems the perceptron can also learn linear discriminant functions to separate two or more classes with a threshold function, for example:

$$y(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{2.2}$$

where a is the weighted sum of inputs. When dealing with only two classes a single output unit can be used. Because the output is linear, the classes to separate should be linearly separable. In the case of more than two classes more than one output unit should be used. Each output unit then acts as a one-versus-all threshold function. In the case that the posterior probability is required a sigmoid function can be used for two classes, like the logistic function:

$$y = \frac{1}{1 + \exp(-a)}, \tag{2.3}$$

or the softmax function for more than two classes:

$$y_i = \frac{\exp(a_i)}{\sum_k \exp(a_k)}, \tag{2.4}$$

where k is the number of classes.

2.1.2 The multilayer perceptron

When the target function to be learned is nonlinear a simple perceptron won't suffice. A famous nonlinear function is the XOR function. This function takes two inputs and has a single output unit. Because of the linear nature of the perceptron it fails in fitting this function. To overcome this problem the network should introduce some kind of nonlinearity.

In Figure 2.2, the structure of a multilayer perceptron is given. For this network a hidden layer is introduced which, as with the input layer, has its own bias unit h_0, with value $+1$. This network can be thought of as multiple layers of perceptrons stacked onto each other. The first layer of perceptrons have $\mathbf{x}$ as the input and $\mathbf{h}$ as the output vector. The second layer has $\mathbf{h}$ as the input and $\mathbf{y}$ as the output vector.

If the output of a hidden unit were to be calculated in the same way as the output of an output unit in a basic perceptron, this network would still not be able to solve nonlinear functions. Combining multiple linear functions in a linear fashion results in just another linear function. Therefore, nonlinearity is introduced by applying a sigmoid-like function to the hidden units' output. This sigmoid function is also necessary for gradient-based learning as these kind of functions are differentiable (hard threshold functions are not). The sigmoid function was already provided earlier in Eq. (2.3), but there are other nonlinear activation functions available. The sigmoid function's output has a range of $[0, +1]$, the hyperbolic tangent function, for example, ranges from $[-1, +1]$.

A multilayer perceptron can have as many hidden units or hidden layers as needed, but more than one hidden layer is generally not necessary as any continuous function can be fitted using a single hidden layer with a sufficient number of units [11].

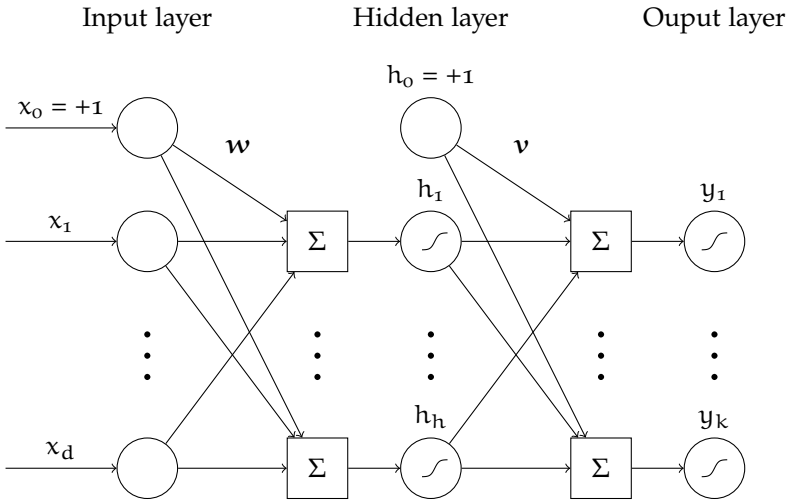


Figure 2.2: The multilayer perceptron. $x_1, \ldots, x_d$ are the input units, x_0 is the bias unit for the input layer, which always takes the value +1, $h_1, \ldots, h_h$ are the hidden units, h_0 is the bias unit for the hidden layer, which always takes the value +1 as well, $\boldsymbol{w}$ is the weight vector from the input to the hidden units, $y_1, \ldots, y_k$ are the output units, and $\boldsymbol{v}$ is the weight vector from the hidden to the output units.

2.1.3 Backpropagation

The parameters of a multilayer perceptron with one hidden layer, from now on referred to as a neural network, are the weight vectors $\boldsymbol{w}$ and $\boldsymbol{v}$. These weights need to be trained in order to approximate the target function. A common learning algorithm for neural networks is the backpropagation algorithm [32]. The idea behind this is to compute the error in the output layer using some predetermined error function and propagate that error back to the weights and use this information to update the weights in the correct direction. Gradient descent is used to update the weight vectors.

Two basic techniques for training are online and batch training. For online learning, the network updates its parameters after each instance that is presented, based on the error of that instance. For batch training, the weights are updated only once, based on the mean error of all instances.

The mean square error (MSE) is usually used as the error term for regression problems:

$$E(\boldsymbol{w}, \boldsymbol{v}|\boldsymbol{x}) = \frac{1}{2P} \sum_{p=1}^{P} \sum_{i=1}^{O} (r_i^p - y_i^p)^2, \tag{2.5}$$

where $\boldsymbol{w}$ and $\boldsymbol{v}$ are the weight vectors for the hidden and output layer, respectively, $\boldsymbol{x}$ is the input vector, P is the number of input patterns, O is the number of output units, r_i^p is the target output and y_i^p is the calculated output for a pattern p for the i-th output unit.

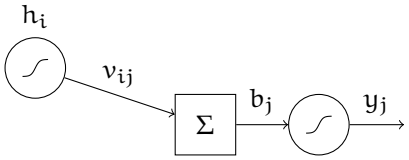


Figure 2.3: Part of a neural network showing the influence of a single weight v_{ij} on the output y_j.

Equation (2.5) is the error term that is used for batch learning, for online learning the weights of the network are updated after presenting each pattern, therefore the error function to minimize is simplified to

$$E^p(\boldsymbol{w}, \boldsymbol{v}|\boldsymbol{x}^p) = \frac{1}{2}\sum_{i=1}^{O}(r_i^p - y_i^p)^2 \tag{2.6}$$

For brevity, the superscript p is omitted in the remainder of this thesis.

Updating the weight vector $\boldsymbol{v}$

To calculate the error with respect to the weights $\boldsymbol{v}$, the partial derivative of the error E with respect to $\boldsymbol{v}$ is derived. In Figure 2.3, a part of a neural network is shown which shows the influence of a single weight v_{ij} on the output. Here, b_j is added for completeness, being the weighted sum of the hidden layer activations with respect to output node y_j. For the activation function the symbol σ will be used for the time being as the type of function to use can differ among various setups. The chain rule is used to derive the partial derivative:

$$\frac{\partial E}{\partial v_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial b_j}\frac{\partial b_j}{\partial v_{ij}} \tag{2.7}$$

The derivation is made for the online variant, after which the update rules for batch training are derived:

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{i=1}^{O}(r_i - y_i)^2\right] = -(r_j - y_j) \tag{2.8}$$

$$\frac{\partial y_j}{\partial b_j} = \frac{\partial}{\partial b_j}\left[\sigma(b_j)\right] = \sigma'(b_j) \tag{2.9}$$

$$\frac{\partial b_j}{\partial v_{ij}} = \frac{\partial}{\partial v_{ij}}\left(\sum_{k=1}^{h} h_k v_{kj}\right) = h_i \tag{2.10}$$

By combining the intermediate results the error for a single weight becomes:

$$\frac{\partial E}{\partial v_{ij}} = -(r_j - y_j)\sigma'(b_j)h_i \tag{2.11}$$

Now the update rule can be constructed using gradient descent:

$$\begin{aligned}\Delta v_{ij} &= -\alpha\frac{\partial E}{\partial v_{ij}} \\ &= \alpha(r_j - y_j)\sigma'(b_j)h_i,\end{aligned} \tag{2.12}$$

where α is the learning rate.

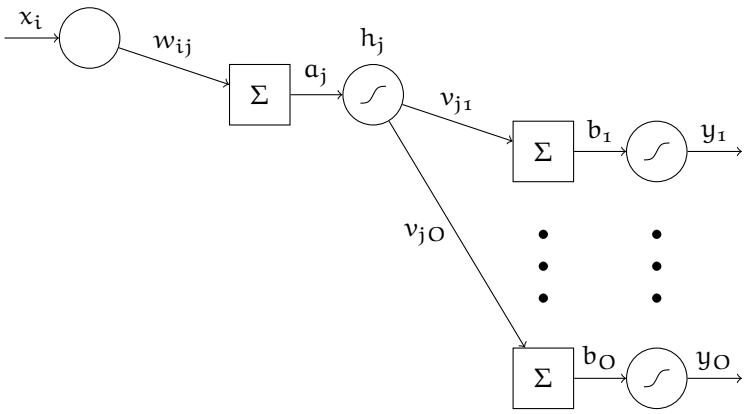


Figure 2.4: Part of a neural network diagram showing the influence of a single weight w_{ij} on the output $\mathbf{y}$.

For batch learning the update rule can be implemented by calculating the weight updates for each pattern, updating the weights only once by the mean of the weight updates.

Updating the weight vector $\boldsymbol{w}$

The update rule for the weight vector $\boldsymbol{w}$ can be derived in a similar way, but the influence of these weights are bigger than those of the second layer. Therefore, the update rule needs to take into account the influence of the weight to all output nodes, see Fig. 2.4 (a_j is added for completeness which corresponds to the weighted sum with respect to hidden unit h_j). For the activation function in the hidden layer the symbol τ is used. Again, for online learning the weights are updated one by one:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^{O} \left(\frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial b_k} \frac{\partial b_k}{\partial h_j} \right) \frac{\partial h_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \tag{2.13}$$

$$\frac{\partial E}{\partial y_k} = \frac{\partial}{\partial y_k} \left[\frac{1}{2} \sum_{l=1}^{O} (r_l - y_l)^2 \right] = -(r_k - y_k) \tag{2.14}$$

$$\frac{\partial y_k}{\partial b_k} = \frac{\partial}{\partial b_k} \left[\sigma(b_k) \right] = \sigma'(b_k) \tag{2.15}$$

$$\frac{\partial b_k}{\partial h_j} = \frac{\partial}{\partial h_j} \left(\sum_{l=1}^{h} h_l v_{lk} \right) = v_{jk} \tag{2.16}$$

$$\frac{\partial h_j}{\partial a_j} = \frac{\partial}{\partial a_j} \left[\tau(a_j) \right] = \tau'(a_j) \tag{2.17}$$

$$\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=d}^{h} x_k w_{kj} \right) = x_i \tag{2.18}$$

When combining the intermediate results the derivative of the error for a single weight becomes:

$$\frac{\partial E}{\partial w_{ij}} = -\tau'(a_j) x_i \sum_{k=1}^{O} (r_k - y_k) \sigma'(b_k) v_{jk} \tag{2.19}$$

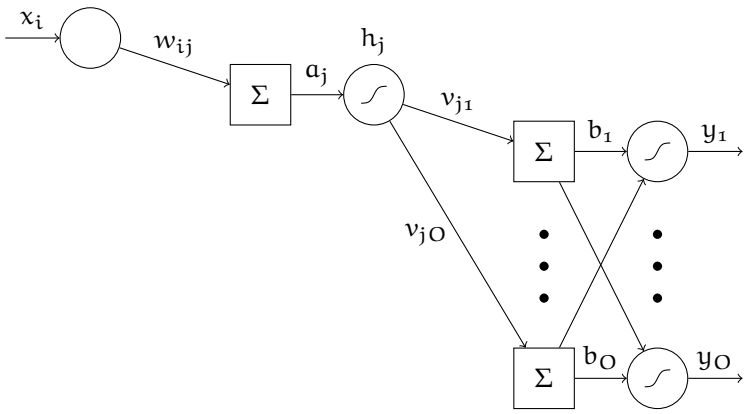


Figure 2.5: Part of a neural network diagram with the softmax activation function at the output layer showing the influence of a single weight w_{ij} on the output $\mathbf{y}$.

Now the update rule can be constructed using gradient descent:

$$\begin{aligned} \Delta w_{ij} &= -\alpha \frac{\partial E}{\partial w_{ij}} \\ &= \alpha \tau'(a_j) x_i \sum_{k=1}^{O} (r_k - y_k) \sigma'(b_k) v_{jk} \end{aligned} \tag{2.20}$$

For batch training again the weights are updated only once based on the mean error.

Softmax & cross-entropy

As mentioned before, the softmax function is often utilized for multiclass discriminant functions, given by Eq. (2.4). Together with the softmax activation function, instead of using the mean square error (MSE) measure, the cross-entropy error function is often used:

$$E_{ce}(\boldsymbol{w}, \boldsymbol{v}|\boldsymbol{x}) = -\sum_{i=1}^{O} r_i \log y_i, \tag{2.21}$$

The update rules can be inferred in a similar way as with a neural network using the MSE function, but with the adaptation of using the softmax and cross-entropy functions:

$$\frac{\partial E_{ce}}{\partial v_{ij}} = \sum_{k=1}^{O} \left(\frac{\partial E_{ce}}{\partial y_k} \frac{\partial y_k}{\partial b_j} \right) \frac{\partial b_j}{\partial v_{ij}} \tag{2.22}$$

$$\frac{\partial E_{ce}}{\partial w_{ij}} = \sum_{l=1}^{O} \left[\sum_{k=1}^{O} \left(\frac{\partial E_{ce}}{\partial y_k} \frac{\partial y_k}{\partial b_l} \right) \frac{\partial b_l}{\partial h_j} \right] \frac{\partial h_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \tag{2.23}$$

Note the additional summation in these update rules. This is because the activation function used in the output layer depends on all weighted sums (see Fig. 2.5).

As it turns out, the derivation from the error towards b_i is rather mathematically convenient. The first part of Eq. (2.22) consists of two cases, one where

$k = j$ and one where $k \neq j$. For Eq. (2.23) this corresponds to $k = l$ and $k \neq l$, respectively. Both cases are derived individually for Eq. (2.22) and later adapted for Eq. (2.23):

$$\frac{\partial E_{ce}}{\partial y_k} = \frac{\partial}{\partial y_k}\left(-\sum_{l=1}^{O} r_l \log y_l\right) = -\frac{r_k}{y_k} \tag{2.24}$$

For $k = j$:

$$\frac{\partial y_k}{\partial b_k} = \frac{\partial}{\partial b_k}\left(\frac{e^{b_k}}{\sum_{i=1}^{O} e^{b_i}}\right) \tag{2.25}$$

For this derivation the quotient rule is used:

$$\text{if} \quad f(x) = \frac{g(x)}{h(x)}, \quad \text{then} \quad f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{[h(x)]^2} \tag{2.26}$$

Substituting Eq. (2.25) into (2.26) results in:

$$g(b_k) = e^{b_k}, \quad g'(b_k) = e^{b_k} \tag{2.27}$$

$$h(b_k) = \sum_{i=1}^{O} e^{b_i}, \quad h'(b_k) = e^{b_k} \tag{2.28}$$

Plugging these in gives:

$$\begin{aligned}
\frac{\partial y_k}{\partial b_k} &= \frac{e^{b_k}\sum_{i=1}^{O} e^{b_i} - e^{b_k}e^{b_k}}{\left(\sum_{i=1}^{O} e^{b_i}\right)^2} \\
&= \frac{e^{b_k}\sum_{i=1}^{O} e^{b_i}}{\sum_{i=1}^{O} e^{b_i}\sum_{i=1}^{O} e^{b_i}} - \frac{e^{b_k}e^{b_k}}{\sum_{i=1}^{O} e^{b_i}\sum_{i=1}^{O} e^{b_i}} \\
&= \frac{e^{b_k}}{\sum_{i=1}^{O} e^{b_i}} - \frac{e^{b_k}}{\sum_{i=1}^{O} e^{b_i}}\frac{e^{b_k}}{\sum_{i=1}^{O} e^{b_i}} \\
&= y_k - y_k y_k
\end{aligned} \tag{2.29}$$

For $k \neq j$:

$$\begin{aligned}
\frac{\partial y_k}{\partial b_j} &= \frac{\partial}{\partial b_j}\left(\frac{e^{b_k}}{\sum_{i=1}^{O} e^{b_i}}\right) \\
&= \frac{\partial}{\partial b_j}\left[e^{b_k}\left(\sum_{i=1}^{O} e^{b_i}\right)^{-1}\right] \\
&= e^{b_k} * -\left(\sum_{i=1}^{O} e^{b_i}\right)^{-2} e^{b_j} \\
&= -\frac{e^{b_k} e^{b_j}}{\left(\sum_{i=1}^{O} e^{b_i}\right)^2} \\
&= -\frac{e^{b_k}}{\sum_{i=1}^{O} e^{b_i}} \frac{e^{b_j}}{\sum_{i=1}^{O} e^{b_i}} \\
&= -y_k y_j
\end{aligned} \tag{2.30}$$

Both cases can be combined using the Kronecker delta function:

$$\frac{\partial y_k}{\partial b_j} = y_k(\delta_{j,k} - y_j), \tag{2.31}$$

where the Kronecker delta function is defined as:

$$\delta_{i,j} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \tag{2.32}$$

Combining these derivations leads to:

$$\begin{aligned}
\sum_{k=1}^{O}\left(\frac{\partial E_{ce}}{\partial y_k}\frac{\partial y_k}{\partial b_j}\right) &= \sum_{k=1}^{O} -\frac{r_k}{y_k} y_k(\delta_{j,k} - y_j) \\
&= \sum_{k=1}^{O} -r_k(\delta_{j,k} - y_j) \\
&= \sum_{k=1}^{O}(r_k y_j - r_k \delta_{j,k}) \\
&= \left(\sum_{k=1}^{O} r_k\right) y_j - r_j \\
&= y_j - r_j
\end{aligned} \tag{2.33}$$

where $\sum_{k=1}^{O} r_k = 1$, as each image only has one corresponding object label. The update rules become:

$$\begin{aligned}
\Delta v_{ij} &= -\alpha \frac{\partial E_{ce}}{\partial v_{ij}} \\
&= -\alpha(y_j - r_j)h_i \\
&= \alpha(r_j - y_j)h_i
\end{aligned} \tag{2.34}$$

for the second layer of weights, and

$$\begin{aligned}\Delta w_{ij} &= -\alpha \frac{\partial E_{ce}}{\partial w_{ij}} \\ &= \alpha \tau'(a_j) x_i \sum_{k=1}^{O} (r_k - y_k) v_{jk} \end{aligned} \tag{2.35}$$

for the first layer of weights.

2.1.4 Resilient propagation (RPROP)

Resilient propagation (RPROP) is a batch learning scheme which performs adaptations on individual weight steps [39]. This training method specifies an update value for each individual weight and updates this over time based on the error gradient. In comparison to original batch training it shows significant improvement in learning time [39].

RPROP does not use the gradient magnitude to compute how much to update the weights, but only uses its sign. The algorithm starts with a predefined update value Δ_{ij} for each individual weight w_{ij}. At each iteration of training (also called an epoch) the mean gradient of each weight is computed over all samples. If the sign of a gradient is equal to the sign of the gradient in the previous epoch, then it seems that the weight is updated in the correct direction. Therefore, the update value is increased by a factor η^+. If, on the other hand, the sign flips, the update constant is decreased with a factor η^-:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ * \Delta_{ij}^{(t-1)}, & \text{if} \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^- * \Delta_{ij}^{(t-1)}, & \text{if} \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \Delta_{ij}^{(t-1)}, & \text{else} \end{cases} \tag{2.36}$$

$$\text{where } 0 < \eta^- < 1 < \eta^+$$

$\frac{\partial E}{\partial w_{ij}}^{(t)}$ corresponds to the error with respect to weight w_{ij} at epoch, or time, t. Similarly $\frac{\partial E}{\partial w_{ij}}^{(t-1)}$ corresponds to the error at epoch $t-1$. Note that if these errors are multiplied and share the same sign the result is positive, if the sign flips the result is negative.

The update values start with some initial value Δ_0 and are bounded by Δ_{min} and Δ_{max}. Riedmiller and Braun suggests using $\eta^+ = 1.2$ and $\eta^- = 0.5$, as these values provided good overall results [39].

After the update values are updated, the weight updates become:

$$\textbf{if} \quad \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} \geqslant 0 \quad \textbf{then} \quad \Delta w_{ij}^{(t)} := -\text{sign}(\frac{\partial E}{\partial w_{ij}}^{(t)}) * \Delta_{ij}^{(t)} \tag{2.37}$$

$$\textbf{if} \quad \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \quad \textbf{then} \quad \Delta w_{ij}^{(t)} := 0 \tag{2.38}$$

Variations on RPROP

RPROP knows a few adaptations of which two are RPROP$^+$ and iRPROP$^+$ [21]. The basic idea of RPROP$^+$ is that if at some point the error goes up,

it's better to take a step back and revert the weight updates, this is called weight-backtracking. However, this adaptation appeared to be counterproductive. iRPROP$^+$ leans on the same idea, but with the fact that when a weight update does not lead to a change of sign in the derivative, this update is taking the weight closer to its optimum value and therefore does not have to be reverted. This leads to the following adaptation of update rule (2.38) where only the weights are reverted that have caused changes in sign of the derivative in case of an increase in error:

$$\begin{aligned} \textbf{if} \quad & \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \quad \textbf{and} \quad E^{(t)} > E^{(t-1)} \\ & \textbf{then} \quad \Delta w_{ij}^{(t)} := -\Delta w_{ij}^{(t-1)} \end{aligned} \tag{2.39}$$

Of the proposed adaptations in [21] iRPROP$^+$ yielded the best results.

2.1.5 Overfitting

Overfitting is a well-known phenomenon which applies to ANNs as well. Overfitting occurs when the network is trained too long on the same data or has too many trainable parameters and as a consequence is starting to follow the train data too closely, losing any generalizability.

A common approach to learning is to divide the training set into a train, validation and a test set. The network is trained on the train data and every so many epochs the network is validated on the validation set. When the validation error goes up training is stopped. This is called early stopping. Accuracy is finally measured by testing on the test set.

Another way to increase generalizability is to increase the amount of training data. Unfortunately, more train data is not always available, but the amount of train data can also be increased by artificially generated data. Another way is to reduce the number of trainable parameters by, for example, decreasing the size of the network. However, large networks have the potential to be more powerful than small networks, so that's not always a desirable solution.

Regularization

Other techniques are available to decrease overfitting. One of them is adding a regularization term, where the most common ones are the L1 and L2 norm. Such a regularization term can be added to any error function (e.g., Eq. (2.5) and (2.21)) and can be written as:

$$E = E_0 + f_{reg}(\boldsymbol{w}), \tag{2.40}$$

where E_0 is the original error function and $f_{reg}(\boldsymbol{w})$ is the regularization function applied on all the weights except the biases. This corresponds to

$$f_{L1}(\boldsymbol{w}) = \frac{\lambda}{P} \sum_{w} |w| \tag{2.41}$$

for the L1 norm and

$$f_{L2}(\boldsymbol{w}) = \frac{\lambda}{2P} \sum_{w} w^2 \tag{2.42}$$

for the L2 norm, where λ determines the amount of regularization and P is the number of input patterns.

The idea behind these regularization techniques is to penalize large weights. By adding this term all weights tend to go to zero, making the model more simple. Large weights are only allowed if they considerably decrease the first term of the error function. When λ is small the preference will be to minimize the original error function, when λ is large small weights are preferred.

When adding this regularization term to the error function the update rules of the weights need to be updated. The partial derivatives become

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E_o}{\partial w_{ij}} + \frac{\lambda}{P}\text{sign}(w_{ij}) \tag{2.43}$$

and

$$\frac{\partial E}{\partial v_{ij}} = \frac{\partial E_o}{\partial v_{ij}} + \frac{\lambda}{P}\text{sign}(v_{ij}) \tag{2.44}$$

for the L1 norm, where sign(w) is the sign of w (i.e., +1 if w is positive, -1 if w is negative). If $w = 0$ the L1 term isn't differentiable and no regularization will take place. The idea behind regularization is to reduce the weights, when a weight is already zero the weight cannot be decreased anymore, so this poses no problems. Intuitively, the regularization terms for the L1 norm bring the weights closer to zero each epoch, independent of the size of the weight. For the L2 norm the partial derivatives become

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E_o}{\partial w_{ij}} + \frac{\lambda}{P}w_{ij} \tag{2.45}$$

and

$$\frac{\partial E}{\partial v_{ij}} = \frac{\partial E_o}{\partial v_{ij}} + \frac{\lambda}{P}v_{ij} \tag{2.46}$$

In this case, the higher the weights, the more influence the regularization part has. Larger weights are pulled harder towards zero, whereas small weights are only pulled a little. The partial derivatives with respect to the biases remain unaffected for both the L1 and L2 norm.

2.2 Scale invariant feature transform (SIFT)

The SIFT algorithm [30] transforms an image to a collection of local image descriptors. It does this by first detecting stable keypoints in the image. The descriptions of these keypoints are constructed in such a way that they are invariant to scale, rotation and partially invariant to affine transformations and illumination changes.

Keypoint detection will be ignored for now as a fixed grid over the entire image will be used instead. This saves processing power and should work identically well [3]. The fixed grid will be implemented as a sliding window, where the center of the window will function as the keypoint in the SIFT algorithm. The sliding window approach ensures that the grid has overlapping blocks to be able to capture more detail. The size of this window, or image patch, in the SIFT algorithm is usually set to 16×16 pixels.

2.2.1 Assigning keypoint orientation

The first step is to assign an orientation to the keypoint. This orientation is used in a later step to obtain invariance to rotation. To determine the orientation, a histogram is created consisting of 36 bins, each bin covering 10 degrees of a circle. The histogram is formed from the gradient orientations of the neighboring points. For each point in the window, the gradient magnitude and orientation are calculated using pixel differences:

$$m(x,y) = \sqrt{(G(x+1,y) - G(x-1,y))^2 + (G(x,y+1) - G(x,y-1))^2} \tag{2.47}$$

specifies the magnitude and

$$\theta(x,y) = \tan^{-1}\left(\frac{G(x,y+1) - G(x,y-1)}{G(x+1,y) - G(x-1,y)}\right) \tag{2.48}$$

the orientation, where $G(x,y)$ is the pixel intensity at position (x,y) in the Gaussian smoothed grayscale image.

Each pixel point is weighted by its gradient magnitude and by a Gaussian-weighted circular window with $\sigma = 1.5$. When the histogram is created the highest peak is detected and used as the keypoint's orientation. In the SIFT algorithm additional keypoints are created for any other peak within 80% of the height of the highest peak and is given that orientation.

2.2.2 Descriptor computation

This step creates a descriptor for each keypoint that is designed to be as distinctive as possible. Again, the gradient magnitude and orientation are used from the surrounding keypoint pixels in the window. A Gaussian weighting function is also used, this time with σ being half the width of the window. To achieve invariance to rotation the keypoint orientation is subtracted from the window orientations.

Next, the window is divided in 4×4 cells. For each cell a histogram is created consisting of 8 orientation bins (each covering 45 degrees). In a similar way as described above, each histogram is filled with the weighted magnitudes of the pixels. The 16 histograms are concatenated to form a 128-dimensional descriptor.

2.2.3 Normalization

In the last step, the descriptor is normalized to unit length:

$$\hat{\mathbf{u}} = \frac{\mathbf{u}}{\|\mathbf{u}\|} \tag{2.49}$$

Finally, values higher than 0.2 are thresholded and given the value 0.2 to overcome some illumination effects. After that, the descriptor is normalized again to unit length.

2.3 Bag of visual words

The idea behind the bag-of-visual-words approach is based on a popular text classification method called *bag-of-words* [23, 31]. In text classification, word

frequency information is gathered and stored in a histogram. Based on this histogram a classifier can determine the semantic context of the text.

Analogously, this is applied to the visual domain [40]. The *visual words* are local image descriptors extracted from the image. Creating a frequency histogram of raw image descriptors is hard as no predefined vocabulary is available. Instead, the vocabulary can be created using a clustering approach. The number of words in the visual vocabulary then depends on the number of clusters used.

To create the histogram, each visual word is compared to the established clusters and the amount of similarity is added to the histogram. The resulting histogram is given to a classifier which determines the object label. For more details see Section 3.3.

2.4 k-means clustering

k-means clustering is a popular vector quantization method that seeks to minimize the total squared distance between points and their closest cluster. It's widely used primarily due to its intuitive nature, speed and simplicity. It uses a hard assignment scheme, meaning that a data sample only belongs to a single cluster:

$$b_i^t = \begin{cases} 1 & \text{if } \|\mathbf{x}_t - \mathbf{c}_i\| = \min_j \|\mathbf{x}_t - \mathbf{c}_j\| \\ 0 & \text{otherwise} \end{cases}, \tag{2.50}$$

where $\mathbf{x}_t$ is the t-th data sample and $\mathbf{c}_i$ is the i-th cluster center. Given the membership values, b, the total reconstruction error can be defined as:

$$E(\{\mathbf{c}_i\}_{i=1}^k | \mathcal{X}) = \sum_t \sum_i b_i^t \|\mathbf{x}_t - \mathbf{c}_i\|^2, \tag{2.51}$$

which intuitively translates to the total sum of squared distances between each point and their closest cluster.

k-means is an iterative algorithm, at each iteration the membership values are calculated by Eq. (2.50) and the best estimate of the center of a cluster $\mathbf{c}_i$ is calculated by taking the derivative of Eq. (2.51) with respect to $\mathbf{c}_i$ and setting it equal to 0, which results in:

$$\mathbf{c}_i = \frac{\sum_t b_i^t \mathbf{x}_t}{\sum_t b_i^t}, \tag{2.52}$$

which intuitively translates to the mean of all data samples belonging to a cluster. k-means converges when the cluster centers do not change anymore after a single iteration.

2.4.1 Accelerated k-means clustering

The k-means algorithm is very fast for small datasets, but when dealing with a large amount of data samples and clusters this process becomes very slow. At each iteration, the membership values have to be computed and the cluster centers need to be updated. To determine the membership of each data sample, that sample is compared to each cluster center. This process can be speeded up by avoiding unnecessary distance computations when assigning data points to clusters by using the triangle inequality [16]. The more clusters k there are the, more effective this method becomes, but the more storage is required.

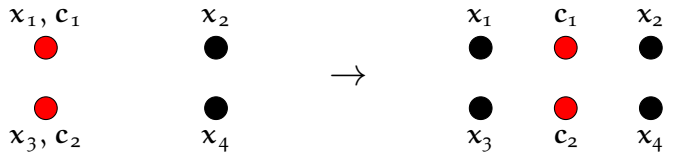


Figure 2.6: Example of bad k-means clustering due to poor initialization. The black dots are the data points, the red dots are the clusters. On the left: the initial clusters are chosen using random data points. On the right: the result of one iteration of the k-means algorithm.

The idea behind this algorithm is that when a cluster center does not move much over a single iteration, most of the point-to-center calculations can be avoided. The triangle inequality is used to determine which distance calculations can be omitted. This property states that for any three points x, y and z, $d(x,z) \leqslant d(x,y) + d(y,z)$. That is, the length of a single side of a triangle never exceeds the sum of the length of the two other sides.

Let $\mathbf{x}_t$ be a data point, $\mathbf{c}_{b^t}$ its current center and $\mathbf{c}$ another cluster center, then:

$$\|\mathbf{x}_t - \mathbf{c}_{b^t}\| \leqslant \frac{1}{2}\|\mathbf{c} - \mathbf{c}_{b^t}\| \quad \Rightarrow \quad \|\mathbf{x}_t - \mathbf{c}_{b^t}\| \leqslant \|\mathbf{x}_t - \mathbf{c}\| \tag{2.53}$$

This means that when the distance between $\mathbf{x}_t$ and its current center $\mathbf{c}_{b^t}$ is smaller than halve the distance of $\mathbf{c}_{b^t}$ to another center $\mathbf{c}$, then $\mathbf{c}$ can be skipped when computing the membership of $\mathbf{x}_t$. In order to use this property all the inter-center distances needs to be computed, but the number of clusters usually is just a small fraction of the number of data points, so overall this will reduce computation time.

Instead of using $\|\mathbf{x}_t - \mathbf{c}_{b^t}\|$ as the condition, Elkan goes a step further by using lower and upper bounds. For full details, see [16].

2.4.2 k-means initialization

To calculate the labels b^t the clusters should already be initialized. Therefore, an initialization procedure is required. The simplest one is to create clusters with random initial values or to appoint unique data points to the clusters.

In Figure 2.6, an example is given using random data points as initial clusters, the data points itself form a rectangle. After a single iteration, k-means has reached convergence, but the resulting clustering is far from optimal. Imagine stretching the width of the rectangle horizontally. The relative position of the clusters will remain the same, but the squared distance from data points to each cluster will be bigger the further it is stretched.

k-means++

To avoid the sometimes poor clusterings found by the k-means algorithm, k-means++ initialization was introduced [4]. The idea behind this method is to spread out the initial clusters over the data. First, a random data point is chosen to be the first cluster center. For the subsequent clusters, the distance d for each data point to its nearest center is calculated. The next cluster will then be chosen randomly, using a weighted probability distribution with probabilities proportional to d^2. This step is repeated until k clusters have been chosen. This

implies that the further a data point is from the already chosen centers, the higher the chance is that this point is chosen to be the next cluster.

Although the initialization step takes longer in computation time, the k-means++ method has proven to often increase both speed and accuracy of the k-means algorithm [4].

k-means‖

Although k-means++ provides a good initialization for k-means, the computation time involved in the choosing of the cluster centers can be very long. After each newly chosen cluster center, the closest cluster for each data point has to be determined again. The sequential nature of k-means++ limits its applicability to large data sets and a large number of clusters.

To make it more scalable, Bahmani et al. introduced the k-means‖ algorithm [5]. Compared to k-means++, k-means‖ only needs a logarithmic number of passes to obtain a near optimal solution, making it a lot faster. It uses an oversampling factor l, which specifies the expected number of points sampled at each iteration.

First, as with k-means++ the first center is chosen randomly from the available data points, this is the first sampled point. The number of iterations is dependent on the initial cost E^*, given by Eq. (2.51). Here, only the single sampled point is used as being a cluster center. Next, additional points are sampled with probability:

$$p_{x^t} = \frac{l * \sum_i b_i^t \|\boldsymbol{x}_t - \mathbf{c}_i\|^2}{E(\mathbf{C})}, \tag{2.54}$$

where $\mathbf{C}$ is the union set of sampled points from the previous iterations and $E(\mathbf{C})$ is the total reconstruction error with respect to $\mathbf{C}$.

This process is repeated $\log(E^*)$ times. Usually the total number of sampled points is larger than the required number of clusters. When all iterations are completed, each sampled point, or sampled cluster center, is weighted by the number of points belonging to it. As a final step, these weighted points are reclustered into k clusters, for example, by using k-means++.

2.4.3 Empty clusters

Having empty clusters is a problem that can occur when using k-means. When an empty cluster is formed, Eq. (2.52) will fail for that cluster. To get rid of them, a random data point can be used for the cluster. Usually, a new point is sampled which is far away from already created clusters, for example, by using k-means++.

2.5 Support vector machines

A support vector machine (SVM) [9] is a supervised learning method capable of regression and both binary and multiclass classification, amongst other tasks. SVMs use a rather different approach than most other classifiers. Instead of estimating the class densities and posterior probabilities, it estimates only the class boundaries. These boundaries can be expressed by the so-called support vectors, training instances which lie on the boundaries of a class. The optimal separating hyperplane for two classes is then defined as lying in the middle of

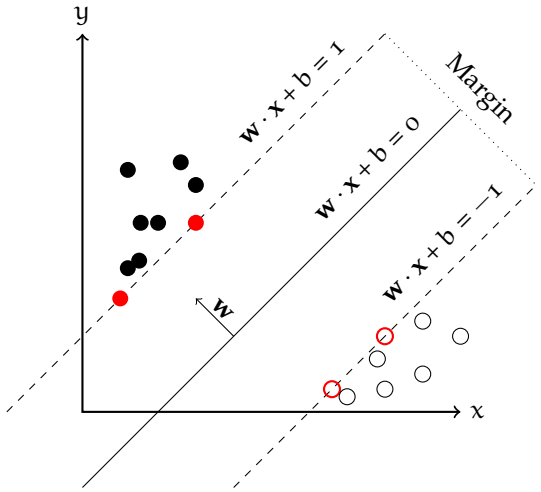


Figure 2.7: An example of a maximum margin hyperplane for an SVM separating two classes. The filled dots mark the positive class (+1) and the open dots the negative class (−1). The dots indicated by a red color are the support vectors. The solid line is the separating hyperplane.

the margin between the support vectors. The main idea is illustrated in Fig. 2.7, which shows an example separating two classes.

How such an hyperplane can be derived for two or more classes is beyond the scope of this thesis. For more details, see [9].

2.5.1 Kernels

In the example above, the classes were linearly separable. If this would not be the case, then no linear hyperplane can be defined to separate the classes. One solution is to take a hyperplane which simply induces the lowest error. Another solution is to map the data to a new space by some nonlinear transformation and then perform linear separation on the transformed data. Functions that transform the data are usually referred to as kernel functions.

A few popular kernel function are the linear kernel (i.e., no transformation), the polynomial kernel, the radial-basis function (RBF), and the sigmoidal function. Each type of kernel has a different set of parameters which have to be determined beforehand (e.g., through a grid-search algorithm). In this section, only the two most important parameters for the RBF kernel will be touched briefly.

2.5.2 RBF kernel parameters

Often, an SVM is not able to find the perfect separating hyperplane that separates two classes with zero error without introducing a model so complex that it loses all generalizability. In these cases a trade-off has to be made between complexity and error, which is controlled by the parameter C.

A second parameter for the RBF kernel is the γ parameter. γ defines the radius of the spherical kernel to apply. The higher γ, the smoother the boundaries will become.

3 Methods

3.1 Preprocessing

Preprocessing is applied to make an image more suitable for feature extraction. As the result of the SIFT descriptor will be used as the target function to extract features, a necessary operation is to first convert the image to grayscale, as is done with SIFT. Conversion is applied using the standard OpenCV[1] conversion function, which corresponds to:

$$I = 0.299R + 0.587G + 0.114B,$$

where I is the resulting grayscale intensity and R, G, and B are the intensities in the individual RGB channels.

For calculating the gradient magnitude and orientations (Eq. (2.47) and (2.48)), SIFT uses a Gaussian smoothed image. Therefore, for the proposed system, each image is smoothened as well. A 3×3 Gaussian filter is chosen for this purpose.

3.2 Neural-SIFT

An important part of the SIFT algorithm includes detecting salient, but stable keypoints. Instead of detecting keypoints a fixed partitioning scheme will be used for the reasons mentioned earlier in Section 2.2. The grid is implemented to have overlapping image patches where each patch is shifted 8 pixels horizontally or vertically (i.e., a sliding window function is applied).

Another modification to the SIFT algorithm is that the keypoint's orientation will not be used to make the keypoint rotation invariant. The reason for this is that preliminary experiments showed that SIFT performed better on the datasets (introduced in Section 4.1) without this step than with. This could be due to the nature of the datasets in which not many rotated objects occur. Another consideration for this is that the function to learn as a consequence becomes easier to fit using a neural network. All other aspects of the SIFT descriptor remain the same.

3.2.1 Network topology

In creating the SIFT descriptor a fixed-sized window around a keypoint is used. Typically, the size is chosen to be 16×16 pixels. For each point in the window, gradient magnitudes and orientations are calculated using the four direct adjacent points. This means that by providing only the 16×16 pixels for a given window as input, the neural network receives incomplete information. Therefore, the neural network receives a 18×18 window as input.

The target function of the network is the local image descriptor function used in the SIFT algorithm: the SIFT descriptor. The dimensionality of the SIFT

[1] OpenCV is an open source computer vision library: http://opencv.org/

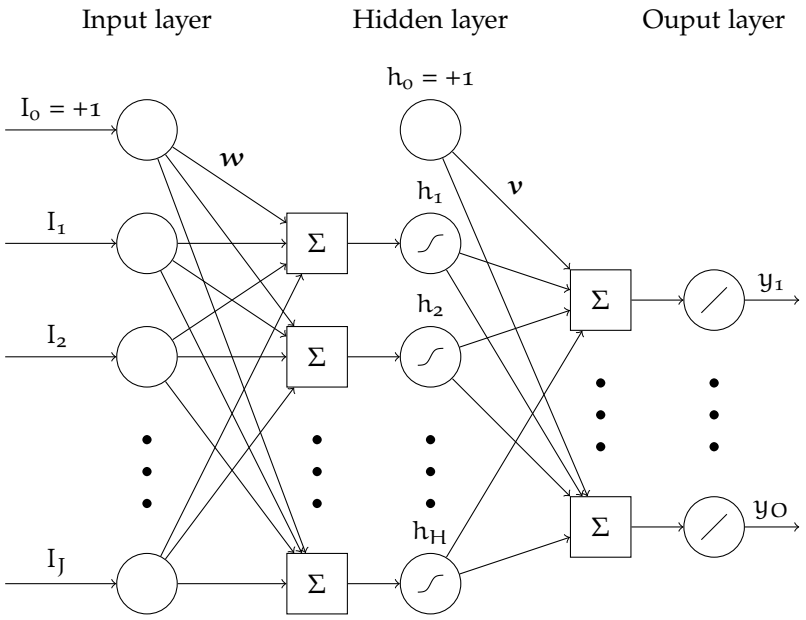


Figure 3.1: Neural-SIFT diagram for feature extraction. I is the input image (I_0 being the bias unit), h the hidden layer (h_0 being the bias unit), and y the output layer. $\boldsymbol{w}$ and $\boldsymbol{v}$ are the weight vectors for the hidden and output layer, respectively.

descriptor is 128 (see Section 2.2.2), therefore the output layer of the neural network consists of 128 units as well.

Cybenko proved that a single layer is enough to fit any continuous function [11]. Therefore, only a single hidden layer is used for this network. One important thing to consider before establishing the number of hidden units to use is that when applying full backpropagation in a later stage, the network will be further trained on training data. When the number of hidden units is large, overfitting can occur more easily. So, even though a smaller train and validation error for this network can be obtained by using more hidden units, using less hidden units may give better results at a later stage. The number of units as well as the type of activation function to use in the hidden layer are determined empirically, taking these aspects into consideration.

For the output layer, using a sigmoid-like function does not seem to make much sense as the network is dealing with a regression problem (i.e., a continuous target function), not classification. Therefore, the linear activation function is used in the output layer.

Even though the SIFT descriptor is computed using local neighborhood pixels, all layers are fully connected. This enables the network to possibly learn more complex functions when the error from the classifier is included. Both the input and the hidden layer have a single bias unit which always takes the value +1 as input. The network topology is shown in Figure 3.1.

3.2.2 Training the network

The target function of this network is of continuous nature. The objective function to minimize, therefore, is chosen to be the mean square error (MSE) using online learning (Eq. (2.6)).

The training size is of great influence to which training algorithm to use. A typical train image contains around 900 patches which translates to 900 train instances per image. Considering 10 object classes with 15 training images per class, this translates to roughly 135,000 train instances in total. For regression problems, online gradient descent learning is well suitable and is often faster than using batch-type training algorithms like RPROP. Especially when the number of training samples is high, updating the network after each instance can speed up training quite substantially.

As discussed earlier, overfitting is a problem often encountered while training neural networks (see Section 2.1.5). To minimize overfitting, a regularization term is added to the error function. The type of regularization and the value of λ to use is determined empirically.

The update rules are:

$$\begin{aligned}
\Delta v_{ij} &= -\alpha \frac{\partial E}{\partial v_{ij}} \\
&= \alpha \left((r_j - y_j)\sigma'(b_j)h_i - \frac{\lambda}{P}\text{sign}(v_{ij}) \right) \quad \text{or} && (3.1) \\
&= \alpha \left((r_j - y_j)\sigma'(b_j)h_i - \frac{\lambda}{P}v_{ij} \right) && (3.2)
\end{aligned}$$

for the second layer of weights, and

$$\begin{aligned}
\Delta w_{ij} &= -\alpha \frac{\partial E}{\partial w_{ij}} \\
&= \alpha \left(\tau'(a_j)I_i \sum_{k=1}^{O} \left[(r_k - y_k)\sigma'(b_j)v_{jk} \right] - \frac{\lambda}{P}\text{sign}(w_{ij}) \right) \quad \text{or} && (3.3) \\
&= \alpha \left(\tau'(a_j)I_i \sum_{k=1}^{O} \left[(r_k - y_k)\sigma'(b_j)v_{jk} \right] - \frac{\lambda}{P}w_{ij} \right) && (3.4)
\end{aligned}$$

for the first layer of weights. Here α is the learning rate, σ and τ are the activation functions for the output and hidden layer, respectively. Equations (3.1) and (3.3) are applied when using the L1 norm, (3.2) and (3.4) are used when the L2 norm is chosen. The weights of the network are initialized using the Nguyen-Widrow algorithm [35]. This method initializes the weights so that the active region of each unit will be distributed roughly uniformly over the layer's input space.

Stopping criterion

The number of training examples, as mentioned above, is quite large for a typical dataset. Therefore, the network is expected to generalize quite well. To guarantee convergence, however, the learning rate α is multiplied by a factor of 0.997 each epoch. The number of epochs to train is determined empirically.

3.3 Bag of visual words

3.3.1 Clustering

After the neural-SIFT network is trained, the visual vocabulary is constructed by creating clusters which represent the data as closely as possible. The clus-

ters are determined using the k-means algorithm which is described in great detail in Section 2.4. This algorithm involves choosing the distance metric and the number of clusters to use. As distance metric the widely used squared Euclidean distance is chosen (see Eq. (3.7)). The number of clusters to use is determined empirically.

3.3.2 Creating the image histogram

The next step is to create an image histogram which serves as input for the classifier, for which numerous methods are available. Philbin et al. transformed the distances from a feature vector to each cluster centroid to a sort of similarity value according to:

$$\exp\left(-\frac{d^2}{2\sigma^2}\right), \tag{3.5}$$

where d is the distance between the feature vector and a cluster centroid and σ is a parameter which specifies how much to penalize distance [38]. van Gemert et al. proposed to use the codeword uncertainty method (UNC) which calculates the probability of a feature vector belonging to a certain cluster [45]. The function is given by:

$$UNC(\mathbf{c}_h) = \frac{1}{N}\sum_{i=1}^{N}\frac{K_\sigma(d(\mathbf{c}_h, \mathbf{f}_i))}{\sum_{j=1}^{|V|} K_\sigma(d(\mathbf{c}_j, \mathbf{f}_i))}, \tag{3.6}$$

where K_σ is the one-dimensional Gaussian kernel, N is the number of feature vectors in the image, $|V|$ is the number of clusters, $\mathbf{c}_i$ is the ith cluster and $\mathbf{f}_i$ is the ith feature vector.

For this system a hybrid approach is used in which similarities are calculated in a similar fashion as in Eq. (3.5) and the amount of probability mass is normalized similarly as in Eq. (3.6).

Figure 3.2 shows the topology of constructing the image histogram. In this figure, a neural-SIFT output vector corresponds to a feature vector $\mathbf{f}$. Each feature vector is compared to each trained cluster by using the squared Euclidean distance metric. The distance $d_{i,j}$ between a feature vector $\mathbf{f}_i$ and a cluster center $\mathbf{c}_j$ is defined as:

$$d_{i,j} = \|\mathbf{f}_i - \mathbf{c}_j\|^2 \tag{3.7}$$

The resulting distances are consequently transformed to similarity values. The similarity $s_{i,j}$ between a feature vector $\mathbf{f}_i$ and a cluster center $\mathbf{c}_j$ is given by:

$$s_{i,j} = \frac{\exp(-\zeta * d_{i,j})}{\sum_{k=1}^{C}\exp(-\zeta * d_{i,k})}, \tag{3.8}$$

where ζ is a constant specifying how much to penalize distance. The bigger ζ, the more the distance is penalized and the more this function approximates the max function. Intuitively, the smaller the distance between a feature vector and a cluster, the higher the similarity will become. The similarity values are normalized over all clusters such that the sum of similarities for a single feature vector equals 1.

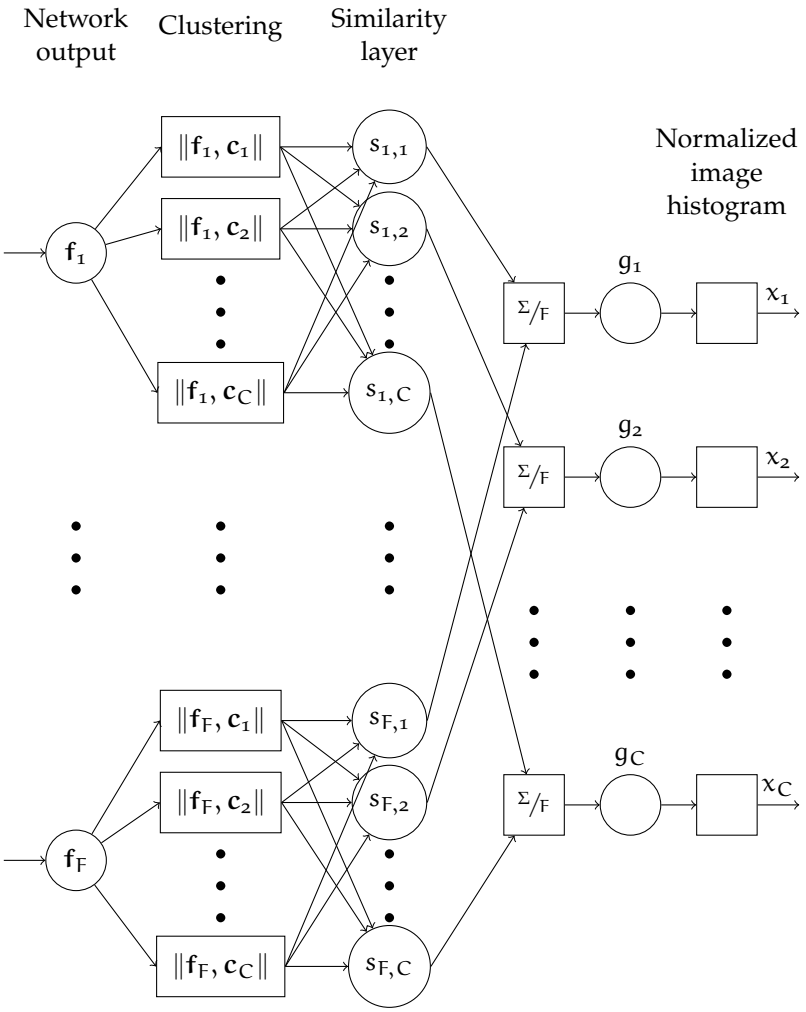


Figure 3.2: Diagram showing the process of creating the image histogram. Here, the neural-SIFT network output vector corresponds to a feature vector $\mathbf{f}$. For each feature vector the distance to each cluster $\mathbf{c}$ is calculated and then transformed to similarity values. Next, all these values are summed and divided by the number of instances F, resulting in a normalized image histogram $\mathbf{g}$. The last step is to further normalize the image histogram to the range $[-1 : +1]$ for the classifier.

When for each patch in the image the vector of similarity values has been computed, they are summed and divided by the number of windows (F) to obtain the required image histogram entry g_j for cluster j:

$$g_j = \frac{1}{F} \sum_{i=1}^{F} s_{i,j} \tag{3.9}$$

To make this histogram more suitable for the classifier input, the histogram is normalized to the range $[-1 : +1]$ for each input x_i by:

$$x_i = \frac{2 * (g_i - min_i)}{max_i - min_i} - 1, \tag{3.10}$$

where min_i and max_i are the lowest and highest value of g_i over all the training data, respectively.

3.4 Neural classifier

When the clusters have converged, the final stage of this system can be trained. This part involves mapping the input to actual object labels.

3.4.1 Network topology

The result of the second stage, bag of visual words, is a histogram of visual word frequencies. The dimensionality of this histogram equals the number of clusters used, C. The number of input units for the neural network classifier, therefore, equals C. The number of output units in the output layer equals the number of object classes N. This directly reflects the flexibility of the system. To add more object classes to the database, only the neural classifier has to be updated by adding additional output units and connections.

As with the neural-SIFT network, only one hidden layer is used. Again, the number of hidden units and the activation functions used were set experimentally. The difference with respect to the neural-SIFT network is that the target function in this case is a binary function where each output unit corresponds to a one-versus-all classifier. Therefore, the softmax function is used at the output layer (see Eq. (2.4)).

Again, all layers are fully connected and have an additional bias unit. The topology of the neural network classifier is shown in Figure 3.3.

3.4.2 Training the network

Training the network is similar to that described in Section 3.2.2. The difference is that for training the neural-SIFT network the MSE function is used, for this network the cross-entropy error function is applied (see Section 2.1.3: softmax & cross-entropy).

As with the neural-SIFT network a regularization term is added to the error function. The type of regularization and the value of λ to use is determined empirically.

The error at the weights of the neural classifier network becomes:

$$\frac{\partial E_{ce}}{\partial v'_{ij}} = (r'_j - y'_j)h'_i + \frac{\lambda}{P} sign(v'_{ij}) \tag{3.11}$$

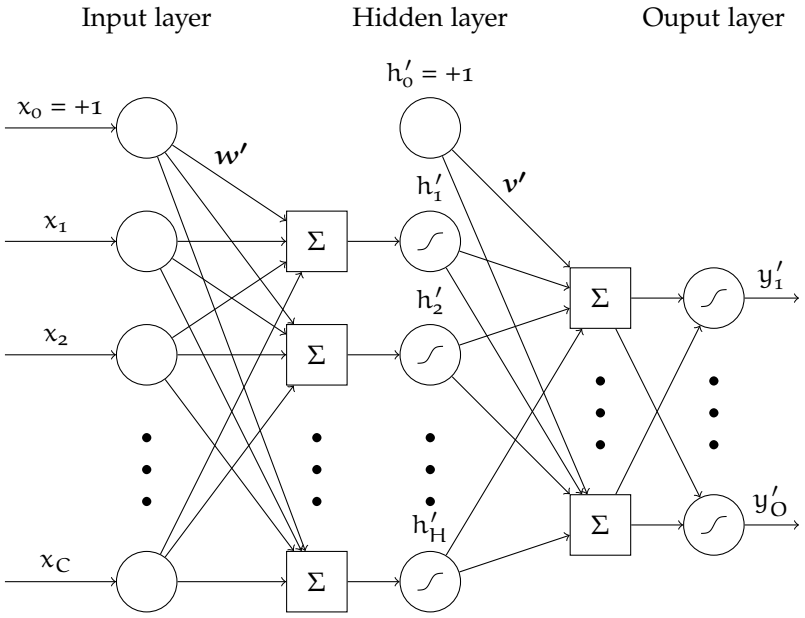


Figure 3.3: Neural classifier diagram. x is the input image histogram (x_0 being the bias unit), h' the hidden layer (h'_0 being the bias unit), and y' is the output layer which translates to the classification output. $\boldsymbol{w'}$ and $\boldsymbol{v'}$ are the weight vectors for the hidden and output layer, respectively.

and

$$\frac{\partial E_{ce}}{\partial w'_{ij}} = \varphi'(a'_j)x_i \sum_{k=1}^{N} \left[(r'_k - y'_k)v'_{jk}\right] + \frac{\lambda}{P}\text{sign}(w'_{ij}) \tag{3.12}$$

when using L1 norm regularization, or

$$\frac{\partial E_{ce}}{\partial v'_{ij}} = (r'_j - y'_j)h'_i + \frac{\lambda}{P}v'_{ij} \tag{3.13}$$

and

$$\frac{\partial E_{ce}}{\partial w'_{ij}} = \varphi'(a'_j)x_i \sum_{k=1}^{N} \left[(r'_k - y'_k)v'_{jk}\right] + \frac{\lambda}{P}w'_{ij} \tag{3.14}$$

when using the L2 norm. Here, φ is the activation function for the hidden layer.

Using gradient descent the update rules become:

$$\Delta v'_{ij} = -\beta \frac{\partial E_{ce}}{\partial v'_{ij}} \tag{3.15}$$

and

$$\Delta w'_{ij} = -\beta \frac{\partial E_{ce}}{\partial w'_{ij}}, \tag{3.16}$$

where β is the learning rate.

Instead of using online learning, the iRPROP$^+$ training algorithm is used for training the neural classifier. As mentioned in Section 2.1.4, this training algorithm outperforms the classic batch training algorithm significantly in learning

time and has the additional benefit that no learning rate has to be specified beforehand [39].

By changing the training algorithm, the update equations (3.15) and (3.16) need to be changed as well. These equations can be easily adopted to the ones mentioned in Section 2.1.4 (i.e., a single update equation is changed to Equations (2.37) and (2.39)). The weights of the network are initialized using the Nguyen-Widrow algorithm [35].

Stopping criterion

Whenever a new lowest validation error is observed, the weights of the network at that state are saved. When not encountering a better validation error within 50 epochs of the current minimum, training is terminated and the stored weights are applied.

3.5 Full backpropagation

When all the parts of the system are individually trained, the system as a whole is trained by propagating the error from the classifier output all the way back to the neural-SIFT weights $\boldsymbol{w}$ and $\boldsymbol{v}$. Using gradient descent, the following partial derivatives have to be solved:

$$\Delta v_{ij} = -\gamma \frac{\partial E_{ce}}{\partial v_{ij}} \tag{3.17}$$

$$\Delta w_{ij} = -\gamma \frac{\partial E_{ce}}{\partial w_{ij}}, \tag{3.18}$$

where γ is the learning rate. The update rule for v_{ij} is considered first as this one is closer to the output. To solve Eq. (3.17), the chain rule is again used, but split up in parts. Each part of the partial derivative corresponds to one of the subsystems: neural network classifier; image histogram construction using the bag of visual words approach; and neural-SIFT, respectively. Note that the neural-SIFT network is used multiple times when creating the image histograms (see Fig. 3.2). Therefore, a single weight of the neural-SIFT network can be updated based on multiple errors.

3.5.1 Neural classifier

All parts will be derived individually, starting with the neural network classifier part. Here, the derivation is first made towards a single network input:

$$\frac{\partial E_{ce}}{\partial x_k} = \sum_{m=1}^{J} \left(\sum_{l=1}^{N} \left[\sum_{k=1}^{N} \left(\frac{\partial E_{ce}}{\partial y'_k} \frac{\partial y'_k}{\partial b'_l} \right) \frac{\partial b'_l}{\partial h'_m} \right] \frac{\partial h'_m}{\partial a'_m} \frac{\partial a'_m}{\partial x_k} \right) \tag{3.19}$$

A big part of this equation was already derived and is provided in Section 2.1.3. The only difference being that in this case the derivation has to be made towards the network input $\boldsymbol{x}$:

$$\frac{\partial a'_m}{\partial x_k} = \frac{\partial}{\partial x_k} \left(\sum_{l=1}^{C} x_l w'_{lm} \right) = w'_{km} \tag{3.20}$$

By incorporating this intermediate result, the error at an input node x_k of the neural network classifier becomes:

$$\frac{\partial E_{ce}}{\partial x_k} = -\sum_{m=1}^{J}\left(\varphi'(a'_m)w'_{km}\sum_{l=1}^{N}(r'_l - y'_l)v'_{ml}\right) \tag{3.21}$$

3.5.2 Bag of visual words

Next, the part corresponding to creating the image histogram is derived. The derivation is made towards a single entry in a feature vector, $f_{f,e}$:

$$\frac{\partial E_{ce}}{\partial f_{f,e}} = \sum_{n=1}^{C}\left(\sum_{k=1}^{C}\left[\frac{\partial E_{ce}}{\partial x_k}\frac{\partial x_k}{\partial g_k}\frac{\partial g_k}{\partial s_{f,k}}\frac{\partial s_{f,k}}{\partial d_{f,n}}\right]\frac{\partial d_{f,n}}{\partial f_{f,e}}\right) \tag{3.22}$$

This equation is built up in the same manner as Eq. (3.19). A single entry in the feature vector f_f, here denoted as $f_{f,e}$, has influence on the distance towards each cluster. The inner summation corresponds to the influence of a single distance on each similarity value.

First, normalization to the range $[-1, +1]$ is derived:

$$\begin{aligned}\frac{\partial x_k}{\partial g_k} &= \frac{\partial}{\partial g_k}\left(\frac{2*(g_k - min_k)}{max_k - min_k} - 1\right)\\ &= \frac{\partial}{\partial g_k}\left(\frac{2}{max_k - min_k}g_k - \frac{2*min_k}{max_k - min_k} - 1\right)\\ &= \frac{2}{max_k - min_k}\end{aligned} \tag{3.23}$$

The following part corresponds to summing the similarity values and dividing the resulting histogram by the number of patches in the image:

$$\frac{\partial g_k}{\partial s_{f,k}} = \frac{\partial}{\partial s_{f,k}}\left(\frac{1}{F}\sum_{l=1}^{F}s_{l,k}\right) = \frac{1}{F} \tag{3.24}$$

At this point, the error is split up and given to each individual image patch. This means that the subsequent equations need to be applied for each image patch.

The next partial derivative is very similar to Eq. (2.4). It is again a softmax function, but this time there is a function within each exponential function. Therefore, the chain rule can be used. The derivative of the softmax function was already derived in (2.31). The part that is left to derive is:

$$\frac{\partial}{\partial d_{f,k}}(-\zeta d_{f,k}) = -\zeta \tag{3.25}$$

Combining these results in:

$$\frac{\partial s_{f,k}}{\partial d_{f,n}} = -\zeta s_{f,k}(\delta_{k,n} - s_{f,n}) \tag{3.26}$$

Finally, the distance computation derivative (i.e., squared Euclidean distance) is derived:

$$\begin{aligned}\frac{\partial d_{f,n}}{\partial f_{f,e}} &= \frac{\partial}{\partial f_{f,e}}\|\mathbf{f}_f - \mathbf{c}_n\|^2 \\ &= \frac{\partial}{\partial f_{f,e}}\sum_{l=1}^{O}(f_{f,l} - c_{n,l})^2 \\ &= 2(f_{f,e} - c_{n,e}) \end{aligned} \tag{3.27}$$

Now that the error at a single element is derived, the error of each neural-SIFT network can be determined. As each of the weights v_{ij} only influences one particular output unit y_j, only the error for one of the output units is necessary. When considering updating the weights $\boldsymbol{w}$, simply the sum of the errors at the output nodes is taken, in the same way as was done in Eq. (2.13).

3.5.3 Neural-SIFT

The final component of back propagating the error from the classifier back to the feature extractor is going through the neural-SIFT network, which was already derived before, see Eq. (3.1) and (3.3) for updating the weights $\boldsymbol{v}$ and $\boldsymbol{w}$, respectively. The only difference on this part is that the error should not be considered with respect to E, but to E_{ce}.

As with the previous neural network update schemes, regularization can be applied here as well. Which regularization method and regularization constant λ to use is determined experimentally.

The error at the weights of the neural-SIFT network becomes:

$$\frac{\partial E_{ce}}{\partial v_{ij}} = \frac{\partial E_{ce}}{\partial y_j}\sigma'(b_j)h_i + \frac{\lambda}{P}\text{sign}(v_{ij}) \tag{3.28}$$

and

$$\frac{\partial E_{ce}}{\partial w_{ij}} = \tau'(a_j)I_i\sum_{k=1}^{O}\left[\frac{\partial E_{ce}}{\partial y_k}\sigma'(b_k)v_{jk}\right] + \frac{\lambda}{P}\text{sign}(w_{ij}) \tag{3.29}$$

when using L1 norm regularization, or

$$\frac{\partial E_{ce}}{\partial v_{ij}} = \frac{\partial E_{ce}}{\partial y_j}\sigma'(b_j)h_i + \frac{\lambda}{P}v_{ij} \tag{3.30}$$

and

$$\frac{\partial E_{ce}}{\partial w_{ij}} = \tau'(a_j)I_i\sum_{k=1}^{O}\left[\frac{\partial E_{ce}}{\partial y_k}\sigma'(b_k)v_{jk}\right] + \frac{\lambda}{P}w_{ij} \tag{3.31}$$

when using the L2 norm.

Using gradient descent, the update rules become:

$$\Delta v_{ij} = -\gamma\frac{\partial E_{ce}}{\partial v_{ij}} \tag{3.32}$$

and

$$\Delta w_{ij} = -\gamma\frac{\partial E_{ce}}{\partial w_{ij}}, \tag{3.33}$$

where γ is the learning rate.

3.5.4 Training procedure

The question arises which training algorithm to use for this learning scheme. Suppose there are 100 training images, the number of classes or the distribution over them is not relevant here. In one epoch of full backpropagation, each image is presented once to the system. This corresponds to 100 classification errors which can be propagated back towards the neural-SIFT network. When each image is partitioned in 900 image patches, each weight in the network can be updated a total of 90,000 times, when using online learning. For a single image, this corresponds to 900 updates per weight.

The problem with this approach is that the 900 image patches of a single image all share the same underlying neural-SIFT network. When using online learning, after updating the weights for the first image patch, the weights of the network for the second image patch are different than the ones used in calculating the error. Therefore, it makes sense to use the mean of the calculated errors of a single image to update the neural-SIFT network. It will then be updated only once for each image.

When dealing with the 100 images in the given example, this corresponds to 100 network updates per epoch. This number of updates is so low for a single epoch that rather batch training should be used. Batch training minimizes the true error of the entire training set, whereas online learning approximates this true error by using multiple updates. Batch training thus ensures that the error of the training set calculated after each epoch always goes down or stays the same. With using online learning this is not guaranteed.

Instead of using regular batch training, iRPROP$^+$ training is applied for full backpropagation, as was done in training the neural classifier. By changing the training algorithm, the update equations (3.32) and (3.33) need to be changed as well. These equations can be easily adopted to the ones mentioned in Section 2.1.4 (i.e., a single update equation is changed to Equations (2.37) and (2.39)).

Multiple iterations

As with the neural classifier, the settings at the lowest validation error are stored. When no lower validation error is encountered within 50 epochs of the current lowest error, training is stopped and the stored settings are applied. This marks one complete iteration of full backpropagation.

After one single iteration of full backpropagation the system reaches a point where performance cannot be improved anymore by using regular training schemes, but at this point the local image descriptor is updated. This translates to a clustering which does not entirely match with the underlying feature vectors. Therefore, a possible increase in performance can be achieved by updating the clustering and consequently the neural classifier. When this retraining is complete, a second iteration of full backpropagation can start. This cycle of full backpropagation and retraining can be repeated indefinitely, until no more improvement can be obtained.

The existing clustering will be updated given the updated neural-SIFT network and the previous clustering. For updating the neural classifier, a much lower regularization constant λ will be used to prevent the classifier network error from increasing rapidly at the start. This is necessary because the individual learning rates used in iRPROP$^+$ are low at first, leading to more influence for the regularization part.

4 Experiments & results

This chapter is divided into five parts. Section 4.1 provides a short overview of the used datasets. and Section 4.2 describes the training results of the proposed system without the full backpropagation part. Results are compared to systems using the SIFT descriptor, instead of the neural network based descriptor. In Section 4.3, full backpropagation is added to the pipeline. Results are compared to the SIFT descriptor and to the original neural-SIFT descriptor. To investigate the influence of initializing the neural-SIFT network on the SIFT descriptor, Section 4.4 compares the proposed system with using a randomly initialized untrained neural network as feature descriptor, both using the full backpropagation training step. Finally, in Section 4.5 generalizability of the Neural-SIFT descriptor is tested with the use of an SVM classifier.

4.1 Datasets

For the experiments described in this chapter two datasets were used for evaluation and are described below.

4.1.1 Caltech-101

The Caltech-101 dataset consists of images of 101 different object classes. Each class contains about 40-800 images, where most of the classes contain around 50 images. The resolution of each image varies around 300×200 pixels and can be in both portrait or landscape mode. Of all images most have little or no clutter, objects tend to be centered in each image and objects are presented in a stereotypical pose.

For the experiments 10 classes were selected. These include: airplanes, cameras, car sides, cellphones, cups, helicopters, motorbikes, scissors, umbrellas, and watches. For evaluating the different methods, 15 training, 15 validation, and 15 test images were used for each class. This equals the settings used in [2]. Sample images of this dataset are shown in Figure 4.1.

4.1.2 Corel-1k

The Corel-1k dataset consists of 1000 images sorted in 10 classes, with 100 images per class. The classes are: African people, beaches, busses, dinosaurs, elephants, flowers, food, horses, monuments, and mountains. The resolution of the images are all 384×256 pixels, either in portrait or in landscape mode.

All classes were selected for the experiments. From the available 100 images per class, 60 were used for training, 20 for validation, and 20 for testing. Sample images of this dataset are shown in Figure 4.2.

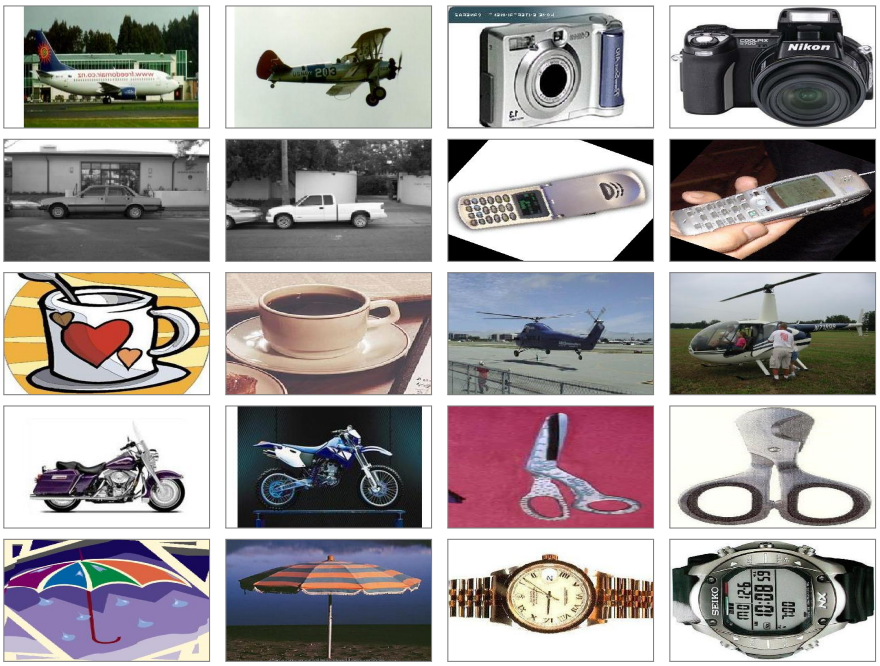


Figure 4.1: Example images of the Caltech-101 dataset showing two images of each of the classes: airplanes, cameras, car sides, cellphones, cups, helicopters, motorbikes, scissors, umbrellas, and watches.

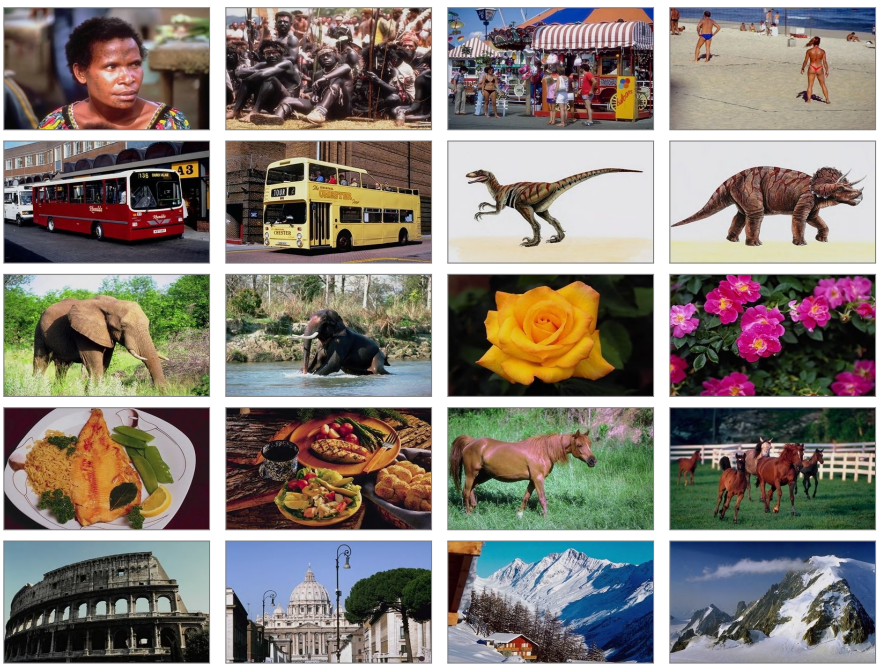


Figure 4.2: Example images of the Corel-1k dataset showing two images of each of the classes: African people, beaches, buses, dinosaurs, elephants, flowers, food, horses, monuments, and mountains.

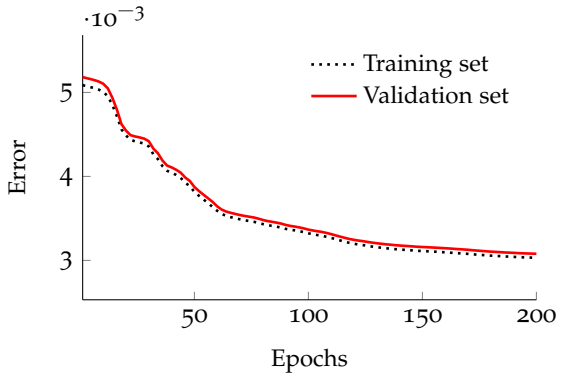


Figure 4.3: Typical error curve for the neural-SIFT network for a single fold.

4.2 Training without full backpropagation

For each individual stage of the system a parameter sweep is used to look for optimal parameters. This sweep is applied to the Caltech-101 dataset only, the resulting optimal settings are consequently applied for the Corel-1k dataset. This section describes this process and reports on the achieved accuracy without the use of full backpropagation. The bag of visual words approach is applied to the SIFT descriptor as well and performance is compared.

For each of the experiments presented in this chapter 10-fold cross-validation is applied to more accurately determine the optimal parameters and to predict the accuracy of the system.

4.2.1 Neural-SIFT feature descriptor

The biggest part of the Neural-SIFT network topology has already been established earlier in Section 3.2.1. The only aspects yet to consider are the number of hidden units and the activation functions to use in the hidden layer. In Section 3.2.1, a brief discussion was given about the number of hidden units to use in the neural-SIFT network. The number of hidden units play an important part, not only in this phase of training, but also when full backpropagation is added. The more units are added now, the higher the chance of overfitting at a later stage. The number of hidden units was set experimentally to a relatively low number, being 50 units. This resulted in still a low error during training of the neural-SIFT network and less overfitting during full backpropagation, when compared to using 100-150 hidden units.

Which activation functions to use was set experimentally. Two functions for the hidden layer were considered: the logistic function (Eq. (2.3)) and the hyperbolic tangent function. For the output layer the linear activation is used as was discussed in Section 3.2.1. For each function a steepness can be defined which controls the steepness of the function. A steepness of 1.0 for the linear function and a steepness of 0.5 for the logistic and hyperbolic tangent function results in the original activation function. A few steepness values were considered, namely 0.25, 0.50, and 0.75. For all settings the learning rate was set to 0.01 and regularization was applied using the L2 norm where λ was set experimentally to $\lambda = 1.0 * 10^{-5}$.

Preliminary experiments showed that 200 epochs was a sufficient amount of epochs for the network to stabilize. Training is therefore terminated after a

Table 4.1: Validation MSE for the Neural-SIFT network using different activation functions averaged over 10 folds. LF is short for the logistic function, HT for the hyperbolic tangent function, and L corresponds to the linear activation function. Hidden and output correspond to the hidden and output layer, respectively. The best result is underlined.

		Hidden					
	$\times 10^{-3}$	$LF_{0.25}$	$LF_{0.50}$	$LF_{0.75}$	$HT_{0.25}$	$HT_{0.50}$	$HT_{0.75}$
Output	$L_{0.25}$	3.152	3.050	<u>3.005</u>	3.076	3.042	3.034
	$L_{0.50}$	3.270	3.239	3.112	3.195	3.129	3.155
	$L_{0.75}$	3.560	3.520	3.415	3.381	3.354	3.355

fixed amount of 200 epochs. An example error curve for training the network is shown in Figure 4.3. After about 140 epochs the train and validation error go down just marginally. The results for each setting are shown in Table 4.1.

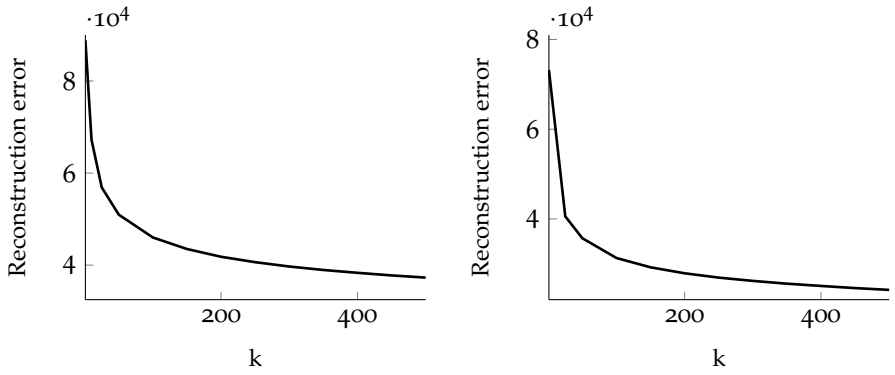
Results indicate that the logistic function with a steepness of 0.75 at the hidden layer, together with a linear function with a steepness of 0.25 at the output layer gives the lowest validation error. These settings are used for the remainder of this chapter. Increasing the steepness even further for the activation function in the hidden layer and lowering the steepness for the output layer did not decrease the validation error any further. That a low steepness of 0.25 for the output layer gives good results is not entirely unexpected, as normalization to unit length is the final step for creating the SIFT descriptor. This ensures that the maximum value of an element in the descriptor is minimized to a low value.

4.2.2 Clustering

For clustering, the only hyperparameter to determine is the number of clusters to use. To determine this parameter the so-called 'elbow' method can be applied. For this method, the reconstruction error given by Eq. (2.51) is plotted as a function of the number of clusters k. When k is large enough, additional clusters will only divide existing small clusters, in which case the reconstruction error decreases only marginally. The optimal number of clusters to use can be found by locating the 'elbow', the point where adding more clusters results in no big decrease in reconstruction error.

The average reconstruction error over 10 folds as a function of k is depicted in Figure 4.4. This is shown for either using the SIFT descriptor (Fig. 4.4a) or for using the neural-SIFT network as feature descriptor (Fig. 4.4b). For both methods, the reconstruction error drops quickly up to about adding 100 clusters, after that the error goes down only marginally. The elbow is then identified to be at $k = 100$ clusters. For the experiments, $k = 100$ and $k = 300$ clusters are chosen for testing.

In creating the input for the classifier, also the parameter ζ needs to be determined (see Eq. (3.8)). ζ determines how much to penalize distance in calculating the similarity values for each cluster. The best choice of this parameter is described below, together with choosing the optimal parameters for the neural classifier.



(a) SIFT descriptor **(b)** Neural-SIFT

Figure 4.4: The reconstruction error as a function of the number of clusters k for two feature descriptors. Error is averaged over 10 folds.

4.2.3 Neural classifier

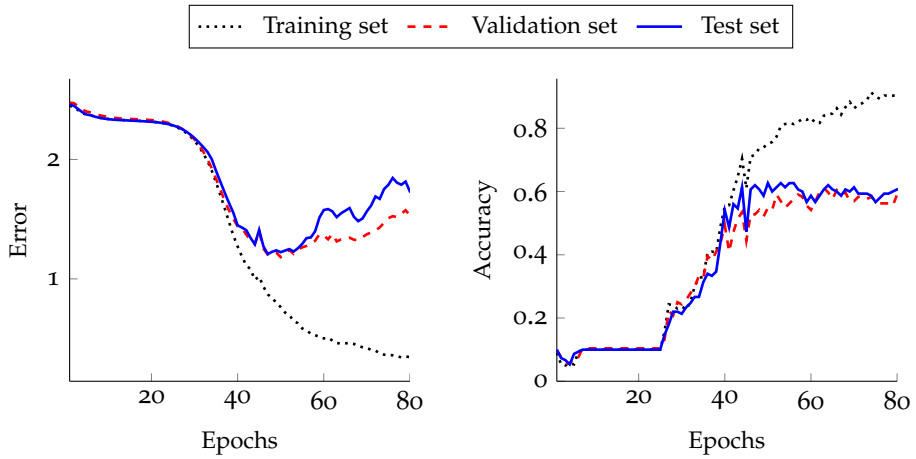
As with the Neural-SIFT network, the network topology of the neural classifier has been established for a big part already (see Section 3.4.1). The number of hidden units and the type of activation function to use in the hidden layer still needs to be determined. Again, only the logistic function and the hyperbolic tangent function were considered together with the same range of steepness values. For determining the optimal value of ζ, the values 1, 3, and 5 were considered, where $\zeta = 1$ corresponds to the original softmax function. For each setting, regularization was applied using the L2 norm where λ was set experimentally to $\lambda = 0.100$.

A typical error curve for training the classifier is shown in Figure 4.5. Around epoch 50 the validation error has reached its local minimum. After that, the validation error starts to go up and overfitting occurs. As mentioned in Section 3.4.2, training is stopped when no lower validation error is encountered within 50 epochs of the current minimum. The results for the different settings using the neural-SIFT network as feature descriptor and $k = 100$ clusters are shown in Table 4.2.

The hyperbolic tangent function with steepness 0.25, 75 hidden units, and $\zeta = 5$ showed the best results. However, the results also indicate that the higher ζ, the lower the validation error becomes. Also decreasing the steepness, increasing the number of hidden units, and using the hyperbolic tangent function instead of the logistic function decreases the error. Therefore, a more thorough parameter sweep was done. 25 until 100 hidden units were considered, ζ values ranging from 6 to 12, and steepness values as low as 0.05 were tested.

The results are shown in Table 4.3 for using the neural-SIFT network as feature descriptor and in Table 4.4 for the SIFT descriptor, both with 100 clusters. Results show that using 75 hidden units with a steepness of 0.10 and ζ of 10 gives the best results for using the neural-SIFT feature descriptor. Changing the steepness to 0.25 gives the best results for using the SIFT descriptor. No further improvement was achieved when setting the steepness to higher values. These settings were used for the experiments involving both datasets.

For $k = 300$ clusters, a similar parameter sweep was applied. For using the neural-SIFT feature descriptor the best settings were similar to the ones using 100 clusters. The best results were obtained using 75 hidden units, the hyperbolic tangent function with a steepness of 0.25, and using $\zeta = 12$. While



(a) Error **(b)** Accuracy

Figure 4.5: Typical error and accuracy curve for the neural classifier for a single fold on the Caltech-101 dataset.

Table 4.2: Initial parameter sweep showing the validation MSE for the neural classifier network while using the neural-SIFT feature descriptor and $k = 100$ clusters. Settings include using different ζ-values for image histogram creation, activation functions, and number of units in the hidden layer, averaged over 10 folds. LF is short for the logistic function and HT for the hyperbolic tangent function. The best result is underlined.

	25 units			**50 units**			**75 units**		
	$\zeta = 1$	$\zeta = 3$	$\zeta = 5$	$\zeta = 1$	$\zeta = 3$	$\zeta = 5$	$\zeta = 1$	$\zeta = 3$	$\zeta = 5$
$LF_{0.25}$	1.527	1.383	1.255	1.563	1.332	1.206	1.540	1.369	1.205
$LF_{0.50}$	1.583	1.378	1.238	1.554	1.341	1.203	1.556	1.353	1.191
$LF_{0.75}$	1.609	1.370	1.245	1.548	1.342	1.236	1.566	1.341	1.193
$HT_{0.25}$	1.452	1.253	1.177	1.463	1.244	1.184	1.426	1.245	<u>1.152</u>
$HT_{0.50}$	1.457	1.283	1.203	1.483	1.266	1.193	1.434	1.255	1.191
$HT_{0.75}$	1.498	1.268	1.179	1.483	1.294	1.224	1.470	1.288	1.220

using 25 units, the hyperbolic tangent function with a steepness of 0.25, and ζ of 10 provided the best results for using the SIFT descriptor. These settings were applied to the Corel dataset as well.

4.2.4 Neural-SIFT versus the SIFT descriptor

Now that the parameters have been established, performance between the two local image descriptors can be compared. The accuracies of the test set for both datasets are shown in Table 4.5. Results show that the SIFT descriptor outperforms the neural-SIFT network on both datasets for $k = 100$ clusters. For $k = 300$, however, neural-SIFT performs slightly better than SIFT for the Caltech-101 dataset. Noticeable is that the accuracies are close to each other, which is not entirely surprising as the neural-SIFT descriptor is an approximation of the SIFT descriptor.

Table 4.3: Validation MSE for the neural classifier network with different settings while using the neural-SIFT descriptor and $k = 100$ clusters. Settings include using different ζ-values for image histogram creation, activation functions, and number of units in the hidden layer, averaged over 10 folds. LF is short for the logistic function and HT for the hyperbolic tangent function. The best result is underlined.

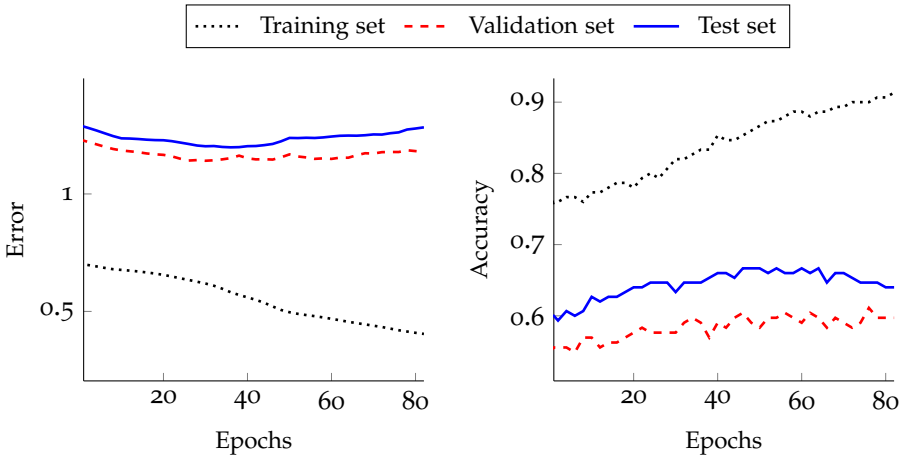
	25 units				**50 units**			
	$\zeta = 6$	$\zeta = 8$	$\zeta = 10$	$\zeta = 12$	$\zeta = 6$	$\zeta = 8$	$\zeta = 10$	$\zeta = 12$
$HT_{0.05}$	1.162	1.175	1.136	1.173	1.169	1.159	1.151	1.146
$HT_{0.10}$	1.183	1.152	1.152	1.131	1.155	1.133	1.153	1.145
$HT_{0.15}$	1.185	1.161	1.152	1.154	1.178	1.148	1.163	1.165
$HT_{0.25}$	1.158	1.192	1.129	1.133	1.182	1.133	1.149	1.156
	75 units				**100 units**			
	$\zeta = 6$	$\zeta = 8$	$\zeta = 10$	$\zeta = 12$	$\zeta = 6$	$\zeta = 8$	$\zeta = 10$	$\zeta = 12$
$HT_{0.05}$	1.171	1.161	1.142	1.131	1.140	1.160	1.136	1.154
$HT_{0.10}$	1.157	1.144	<u>1.121</u>	1.142	1.136	1.129	1.141	1.140
$HT_{0.15}$	1.166	1.169	1.122	1.138	1.162	1.155	1.170	1.146
$HT_{0.25}$	1.179	1.151	1.173	1.173	1.147	1.144	1.136	1.158

Table 4.4: Validation MSE for the neural classifier network with different settings while using the SIFT descriptor and $k = 100$ clusters. Settings include using different ζ-values for image histogram creation, activation functions, and number of units in the hidden layer, averaged over 10 folds. LF is short for the logistic function and HT for the hyperbolic tangent function. The best result is underlined.

	25 units				**50 units**			
	$\zeta = 6$	$\zeta = 8$	$\zeta = 10$	$\zeta = 12$	$\zeta = 6$	$\zeta = 8$	$\zeta = 10$	$\zeta = 12$
$HT_{0.05}$	1.119	1.099	1.100	1.104	1.127	1.135	1.114	1.096
$HT_{0.10}$	1.083	1.116	1.082	1.131	1.130	1.150	1.127	1.105
$HT_{0.15}$	1.108	1.123	1.105	1.088	1.135	1.113	1.134	1.116
$HT_{0.25}$	1.132	1.121	1.077	1.154	1.141	1.110	1.100	1.106
	75 units				**100 units**			
	$\zeta = 6$	$\zeta = 8$	$\zeta = 10$	$\zeta = 12$	$\zeta = 6$	$\zeta = 8$	$\zeta = 10$	$\zeta = 12$
$HT_{0.05}$	1.144	1.138	1.135	1.111	1.154	1.146	1.131	1.100
$HT_{0.10}$	1.107	1.096	1.113	1.105	1.087	1.128	1.084	1.107
$HT_{0.15}$	1.132	1.149	1.117	1.104	1.117	1.137	1.107	1.080
$HT_{0.25}$	1.136	1.151	<u>1.065</u>	1.127	1.163	1.127	1.109	1.115

Table 4.5: Average classification accuracy over 10 folds for using either the SIFT descriptor or the neural-SIFT network as local image descriptor. Performance is based on the test set. Numbers in parentheses show the difference in performance compared to the SIFT descriptor.

	100 clusters		300 clusters	
	SIFT	**Neural-SIFT**	**SIFT**	**Neural-SIFT**
Caltech-101	62.80%	62.47% (−0.33%)	65.33%	68.53% (+3.20%)
Corel-1k	83.15%	81.65% (−1.50%)	86.25%	85.10% (−1.15%)



(a) Error **(b)** Accuracy

Figure 4.6: Example error and accuracy curve for the first iteration of full backpropagation for a single fold (Caltech-101 dataset).

4.3 Full backpropagation training

In this section, full backpropagation training is added. First, results are shown for applying one single iteration of full backpropagation. This approach is extended with multiple iterations in the subsequent section and performance is compared to the SIFT descriptor.

4.3.1 Single iteration

After each individual part of the system is trained, the system reaches a point where traditional training cannot push the performance any further. At this point, the procedure explained in Section 3.5 is applied. First, a single iteration of full backpropagation is considered. A single iteration can consist of multiple training epochs. Regularization was also applied to full backpropagation training, where λ was set experimentally to $\lambda = 0.001$ using the L2 norm. An example error curve for the training, validation and test set for a single fold is shown in Figure 4.6.

The train error goes down continuously accompanied with a steady increase in accuracy. Up to about epoch 30, both validation and test error go down together with the train error. This coincides with an increase in both validation and test accuracy. After around epoch 30, overfitting starts to occur and both

Table 4.6: Results of one single iteration of full backpropagation for both datasets using $k = 100$ clusters, averaged over 10 folds. The + (up) and − (down) sign indicates the direction of the corresponding update.

Caltech-101	**Error update** (+/−)	**Accuracy update** (+/−)
Training set	0.529 → 0.432 (−)	0.849 → 0.893 (+)
Validation set	1.013 → 0.981 (−)	0.660 → 0.680 (+)
Test set	1.201 → 1.185 (−)	0.625 → 0.639 (+)
Corel-1k	**Error update** (+/−)	**Accuracy update** (+/−)
Training set	0.417 → 0.326 (−)	0.876 → 0.913 (+)
Validation set	0.546 → 0.466 (−)	0.836 → 0.864 (+)
Test set	0.599 → 0.526 (−)	0.817 → 0.839 (+)

the validation and test set error go up. In a similar way as with training the neural classifier, training was stopped when the validation error did not reach a new minimum within 50 epochs. The weights stored at the lowest encountered validation error were applied at the end of training.

Figure 4.6 showed the results for one single fold, the results averaged over all 10 folds for both datasets using 100 clusters are shown in Table 4.6. On average the train, validation, and test set errors went down and accuracy went up for both datasets. This indicates that applying full backpropagation can push the system further than was possible with regular training. After one single iteration, test set accuracy went up by 1.40% for the Caltech-101 dataset and 2.20% for the Corel-1k dataset.

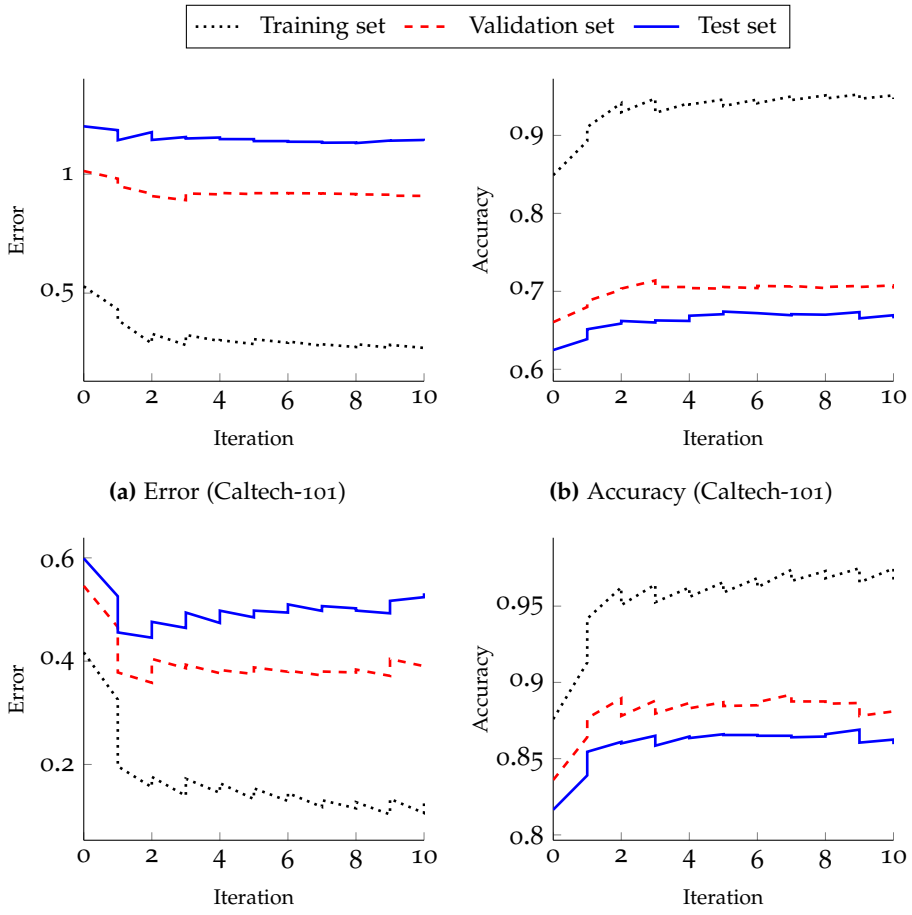
For the $k = 300$ clusters setting the, test set accuracy for the Caltech-101 dataset went up from 68.53% to 69.80% (+1.27%). For the Corel-1k dataset, accuracy went up from 85.10% to 86.70% (+1.60%).

4.3.2 Multiple iterations

After one single iteration, this training scheme cannot improve performance any further. But, as described in Section 3.5.4, additional iterations can be used to possibly push the system even further. The procedure involves updating the current clustering and the neural classifier based on the improved neural-SIFT network. When these are retrained, a second iteration of full backpropagation can begin. This cycle can be repeated over and over again, until no further improvement can be achieved.

From the second iteration onwards, a lower regularization constant λ for the neural classifier is used for reasons mentioned in Section 3.5.4. λ was set experimentally to $\lambda = 0.001$ for both datasets. A maximum of 10 iterations is used for each experiment. This proved to be a sufficient number of iterations for the error to stabilize or to see overfitting on the training data occur.

The development of the errors and accuracies over multiple iterations for both datasets using 100 clusters is shown in Figure 4.7. For the Caltech-101 dataset the validation error went down each single iteration, but went up or down during retraining the clusters and classifier (indicated by the vertical updates at each iteration point). On average, the lowest validation error was reached at iteration 3, before retraining. From iteration 4 onwards the valida-



(a) Error (Caltech-101)

(b) Accuracy (Caltech-101)

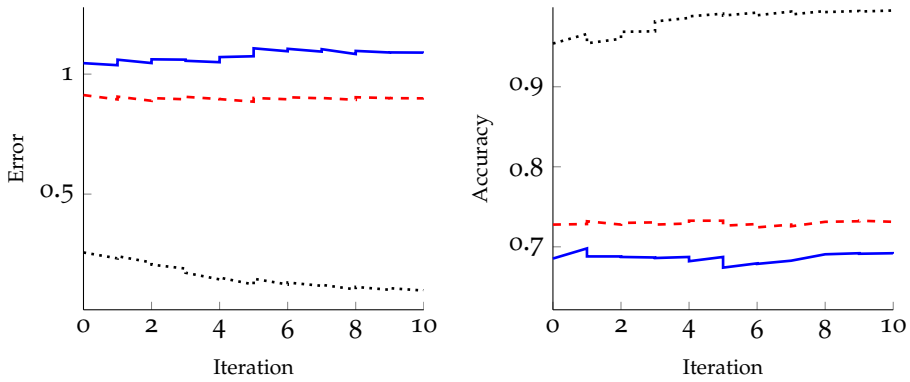
(c) Error (Corel-1k)

(d) Accuracy (Corel-1k)

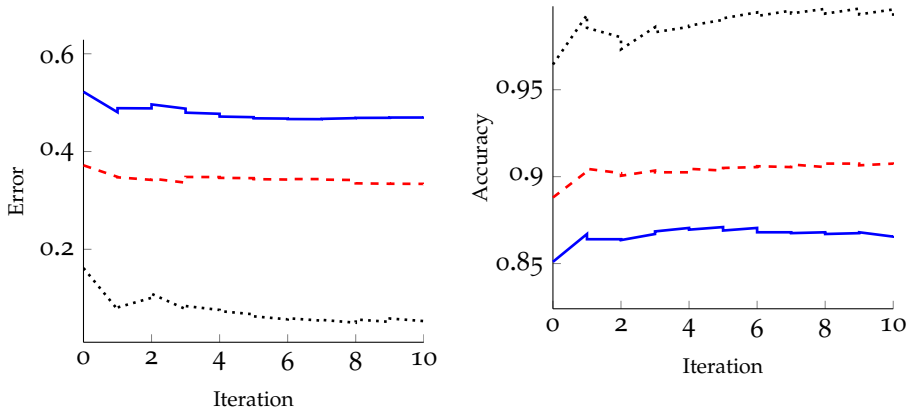
Figure 4.7: Full backpropagation training progress averaged over 10 folds using k = 100 clusters. **(a)** and **(b)** show the training progress for the Caltech-101 dataset, the progress for the Corel-1k dataset is shown in **(c)** and **(d)**. Vertical updates indicate the error and accuracy after retraining the clusters and the neural classifier.

tion error stabilized and for most folds no further training was possible while not letting the validation error increase. For the Corel-1k dataset the lowest average validation error was observed at iteration 2, before retraining. After that point, retraining most of the time led to an increase of validation error. The training error was still going down which indicates possible overfitting on the training set.

Figure 4.8 shows the development of the errors and accuracies over multiple iterations in a similar way, but for using 300 clusters instead. On average, the validation set error for the Caltech-101 dataset reached its lowest point at iteration 5, before retraining. The test set error went down during the first iteration, but showed an increase after that. The validation error for the Corel-1k dataset went down at first, went up around iteration 3, but eventually obtained the lowest validation set error at iteration 10. On average, the test set showed a decrease in error up to iteration 7. For both datasets the train error was really low at the end, which increases the chance of overfitting and therefore makes it more difficult for the validation and test set errors to go down with it.



(a) Error (Caltech-101) **(b)** Accuracy (Caltech-101)



(c) Error (Corel-1k) **(d)** Accuracy (Corel-1k)

Figure 4.8: Full backpropagation training progress averaged over 10 folds using k = 300 clusters. **(a)** and **(b)** show the training progress for the Caltech-101 dataset, the progress for the Corel-1k dataset is shown in **(c)** and **(d)**. Vertical updates indicate the error and accuracy after retraining the clusters and the neural classifier.

The exact figures for all settings, before applying full backpropagation and at the point where the validation error was the lowest, are shown in Table 4.7. When comparing these results with Table 4.6, multiple iterations of full backpropagation can push the accuracy of the system even further than when using just a single iteration. Except for the Caltech-101 dataset with 300 clusters, in each setting the error went down and accuracy went up for both image sets. As one would expect, the train error went down quite substantially in each setting and, as a consequence, the train accuracy got the highest boost. In all cases the validation set error is lower than the test set error, which is to be expected when fine-tuning on the validation set. This also corresponds to a higher accuracy for the validation set compared to the test set.

The largest improvement was achieved when using only 100 clusters for the Caltech-101 dataset. Here, the accuracy of the test set increased from 62.47% to 66.13% (corresponding to an increase of +4.66%). For the Corel-1k dataset accuracy increased by +4.50% going from an initial score of 81.65% to 86.15%. When using 300 clusters the accuracy on the test set for the Caltech-101 dataset increased from 68.53% to 70.20% (increase of +1.67%). The Corel-1k showed an increase from 85.10% to 86.85% (+1.75%).

Table 4.7: Results of applying multiple iterations of full backpropagation for both datasets, averaged over 10 folds. The + (up) and − (down) sign indicates the direction of the corresponding update.

100 clusters		
Caltech-101	**Error update** (+/−)	**Accuracy update** (+/−)
Training set	0.529 → 0.270 (−)	0.849 → 0.946 (+)
Validation set	1.013 → 0.878 (−)	0.660 → 0.719 (+)
Test set	1.201 → 1.168 (−)	0.625 → 0.661 (+)
Corel-1k	**Error update** (+/−)	**Accuracy update** (+/−)
Training set	0.417 → 0.119 (−)	0.876 → 0.973 (+)
Validation set	0.546 → 0.344 (−)	0.836 → 0.900 (+)
Test set	0.599 → 0.480 (−)	0.817 → 0.862 (+)
300 clusters		
Caltech-101	**Error update** (+/−)	**Accuracy update** (+/−)
Training set	0.257 → 0.104 (−)	0.954 → 0.996 (+)
Validation set	0.913 → 0.852 (−)	0.728 → 0.738 (+)
Test set	1.046 → 1.055 (+)	0.685 → 0.702 (+)
Corel-1k	**Error update** (+/−)	**Accuracy update** (+/−)
Training set	0.162 → 0.055 (−)	0.965 → 0.995 (+)
Validation set	0.372 → 0.325 (−)	0.888 → 0.911 (+)
Test set	0.522 → 0.470 (−)	0.851 → 0.869 (+)

4.3.3 Improved neural-SIFT versus the SIFT descriptor

Performance of using the improved neural-SIFT descriptor is compared to the performance of the SIFT descriptor in Table 4.8. Before full backpropagation the SIFT descriptor obtained higher accuracy rates for both datasets in the 100 clusters setting, and achieved higher accuracy for the Corel-1k dataset for the 300 cluster setting. Only the Caltech-101 dataset with 300 clusters showed a better performing neural-SIFT descriptor. After applying multiple iterations of full backpropagation, however, the improved neural-SIFT descriptor takes over and obtains better performance in all settings. For the Caltech-101 dataset increases of +3.33% and +4.87% were realized for the 100 and 300 cluster setting, respectively. A student t-test was conducted to compare the two descriptors. For the 100 clusters setting there was a significant difference in the scores for the SIFT ($M = 62.80, SEM = 0.95$) and the neural-SIFT ($M = 66.13, SEM = 1.24$) descriptor conditions; $t(18) = 2.13, p = 0.047$. Also the 300 clusters setting showed a significant difference between the SIFT ($M = 65.33, SEM = 1.30$) and the neural-SIFT ($M = 70.20, SEM = 0.57$) descriptor conditions; $t(18) = 3.43, p = 0.003$. For the Corel-1k dataset increases of +3.00% and +0.60% were realized. The student t-test could not be used here because we observed that the outcomes do not follow a normal distribution, so the Binomial test was used instead. The results indicated that for both the 100 and 300 clusters setting there were 8 wins, 2 losses. The cumulative Binomial test then gives $P(X \geqslant 8 \mid N = 10) = 0.054$, which is almost significant at the $p = 0.05$ level.

Table 4.8: Average classification accuracy of the test set over 10 folds for using either the SIFT descriptor, the neural-SIFT network or the improved neural-SIFT network as local image descriptor. Numbers in parentheses show the difference in performance when compared to the SIFT descriptor.

	100 clusters		
	SIFT descriptor	**Neural-SIFT**	**Improved neural-SIFT**
Caltech-101	62.80%	62.47% (−0.33%)	66.13% (+3.33%)*
Corel-1k	83.15%	81.65% (−1.50%)	86.15% (+3.00%)
	300 clusters		
	SIFT descriptor	**Neural-SIFT**	**Improved neural-SIFT**
Caltech-101	65.33%	68.53% (+3.20%)	70.20% (+4.87%)**
Corel-1k	86.25%	85.10% (−1.15%)	86.85% (+0.60%)

* The mean difference is significant at the 0.05 level

** The mean difference is significant at the 0.005 level

4.3.4 Confusion matrices

The best performance for both datasets was encountered while using 300 clusters. For this setting the confusion matrices for both datasets are constructed while using the original or improved neural-SIFT descriptor.

A confusion matrix is a $C \times C$ matrix, where C is the number of classes, which shows the correct and wrong classifications of the classifier. It allows one to see which classes are often misclassified and, more specifically, which classes are often confused with one another. In a confusion matrix the diagonal values correspond to correct classifications, the off-diagonal values correspond to incorrect classifications. From these numbers the precision and recall statistics can be calculated. For classification, the precision is the fraction of images predicted to have a certain class that actually belong to that class. For example, if 100 images have been classified as class A, while of these images only 75 actually belong to this class, then the precision for class A equals 75/100 = 0.750. Recall is the fraction of images that are of a specific class that are correctly classified. If there are 50 images that belong to class B and 40 of them have been correctly classified, the recall for class B is 40/50 = 0.800.

The confusion matrices for the Caltech-101 dataset are shown in Table 4.9. For using the original neural-SIFT descriptor the class *cups* showed the worst performance, only 59 out of the possible 150 images were correctly classified. The highest confusion for that class was found to be with the *helicopters* and *cameras* class. Two classes that are semantically highly similar were also often confused with one another, being the *airplanes* and *helicopters*. The best recognized classes were the *cellphones* and the *scissors*, which show the highest recall rate.

When looking at the numbers for the improved neural-SIFT descriptor the largest improvement regarding recall was achieved with the *umbrella* class, showing an increase from 70 to 82 images correctly classified. The top two correctly classified classes were still the *cellphones* and the *scissors*.

Table 4.9: Confusion matrices of the Caltech-101 dataset for the 300 clusters setting, summed over 10 folds. The labels are represented by abstract characters, where **A**: airplanes; **B**: cameras; **C**: car sides; **D**: cellphones; **E**: cups; **F**: helicopters; **G**: motorbikes; **H**: scissors; **I**: umbrellas; and **J**: watches.

Original neural-SIFT

		Predicted class											
		A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**Prec.**	**Recall**
Actual class	**A**	111	0	15	0	0	18	2	0	4	0	0.730	0.740
	B	4	83	4	1	25	7	4	3	5	14	0.686	0.553
	C	4	0	127	0	0	16	2	0	1	0	0.713	0.847
	D	0	0	1	136	4	0	0	1	1	7	0.895	0.907
	E	2	18	2	6	59	20	12	6	12	13	0.457	0.393
	F	15	1	10	1	6	104	3	0	6	4	0.500	0.693
	G	6	4	14	0	5	15	101	2	1	2	0.802	0.673
	H	0	2	0	0	5	0	0	132	2	9	0.835	0.880
	I	8	4	4	6	13	19	2	10	70	14	0.648	0.467
	J	2	9	1	2	12	9	0	4	6	105	0.625	0.700

Improved neural-SIFT

		Predicted class											
		A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**Prec.**	**Recall**
Actual class	**A**	109	1	17	0	0	12	4	0	7	0	0.637	0.727
	B	7	89	1	0	26	5	7	1	9	5	0.701	0.593
	C	10	0	132	0	0	5	2	0	1	0	0.725	0.880
	D	3	0	0	135	3	0	1	0	2	6	0.906	0.900
	E	3	13	2	7	68	9	11	9	15	13	0.466	0.453
	F	22	1	15	1	6	88	3	0	8	6	0.607	0.587
	G	5	6	9	0	6	6	111	0	1	6	0.782	0.740
	H	0	0	0	2	4	0	0	140	1	3	0.864	0.933
	I	10	5	6	3	15	11	1	10	82	7	0.626	0.547
	J	2	12	0	1	18	9	2	2	5	99	0.683	0.660

The confusion matrices for the Corel-1k dataset are shown in Table 4.10. An almost perfect score was achieved for the *dinosaurs* class, having both a precision and recall of 0.995. Also the *buses* and the *flowers* classes showed good performance with the original neural-SIFT descriptor. The lowest performance was encountered with the *African people*, *beaches*, and *mountains* categories. The biggest confusion seemed to be between the *beaches* and the *mountains* classes, which share some semantical features. Interestingly, *mountains* were also often classified as being *food*.

Table 4.10: Confusion matrices of the Corel-1k dataset for the 300 clusters setting, summed over 10 folds. The labels are represented by abstract characters, where **A**: African people; **B**: beaches; **C**: buses; **D**: dinosaurs; **E**: elephants; **F**: flowers; **G**: food; **H**: horses; **I**: monuments; and **J**: mountains.

Original neural-SIFT

		Predicted class											
		A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**Prec.**	**Recall**
Actual class	**A**	154	3	5	1	6	5	8	6	9	3	0.762	0.770
	B	5	151	3	0	3	0	2	10	10	16	0.812	0.755
	C	1	1	187	0	0	0	1	1	9	0	0.926	0.935
	D	0	0	0	199	0	0	0	0	0	1	0.995	0.995
	E	11	1	0	0	170	0	1	8	3	6	0.821	0.850
	F	2	0	0	0	1	193	1	0	1	2	0.946	0.965
	G	13	5	0	0	6	2	166	0	2	6	0.826	0.830
	H	3	5	2	0	10	0	2	172	3	3	0.843	0.860
	I	10	6	4	0	5	0	4	5	160	6	0.796	0.800
	J	3	14	1	0	6	4	16	2	4	150	0.777	0.750

Improved neural-SIFT

		Predicted class											
		A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**Prec.**	**Recall**
Actual class	**A**	160	3	3	2	11	2	10	4	3	2	0.804	0.800
	B	3	159	2	0	3	1	1	9	11	11	0.832	0.795
	C	0	2	185	0	1	0	1	0	11	0	0.939	0.925
	D	0	0	0	200	0	0	0	0	0	0	0.985	1.000
	E	11	1	0	0	174	0	1	8	1	4	0.817	0.870
	F	2	1	0	0	0	192	1	1	1	2	0.928	0.960
	G	11	3	1	1	5	3	168	0	4	4	0.848	0.840
	H	4	2	1	0	12	2	0	176	3	0	0.863	0.880
	I	5	8	4	0	5	0	5	5	162	6	0.818	0.810
	J	3	12	1	0	2	7	11	1	2	161	0.847	0.805

With the improved neural-SIFT descriptor the class *dinosaurs* achieved a perfect score: 200/200 images correct. The precision did go down a bit to 0.985, but it still shows that this class is well separated from the others. All classes, except for the *buses* and the *flowers*, showed an increase in performance. The largest performance differences are seen with the classes that previously showed the poorest accuracy (*African people*, *beaches*, and *mountains*). The highest confusion is seen with *horses* being classified as *elephants* and *mountains* being classified as *beaches*.

Table 4.11: Average classification accuracy of the test set over 10 folds for using either the neural-SIFT network, the improved neural-SIFT network, or the randomly initialized neural-RANDOM network as local image descriptor. Numbers in parentheses show the difference in performance when compared to the original neural-SIFT descriptor.

	100 clusters		
	Neural-SIFT		**Neural-RANDOM**
	Original	**Improved**	**Original**
Caltech-101	62.47%	66.13% (+3.66%)	50.33% (−12.14%)
Corel-1k	81.65%	86.15% (+4.50%)	78.10% (−3.55%)
	300 clusters		
	Neural-SIFT		**Neural-RANDOM**
	Original	**Improved**	**Original**
Caltech-101	68.53%	70.20% (+1.67%)	54.59% (−13.94%)
Corel-1k	85.10%	86.85% (+1.75%)	76.90% (−8.20%)

4.4 Random descriptor initialization

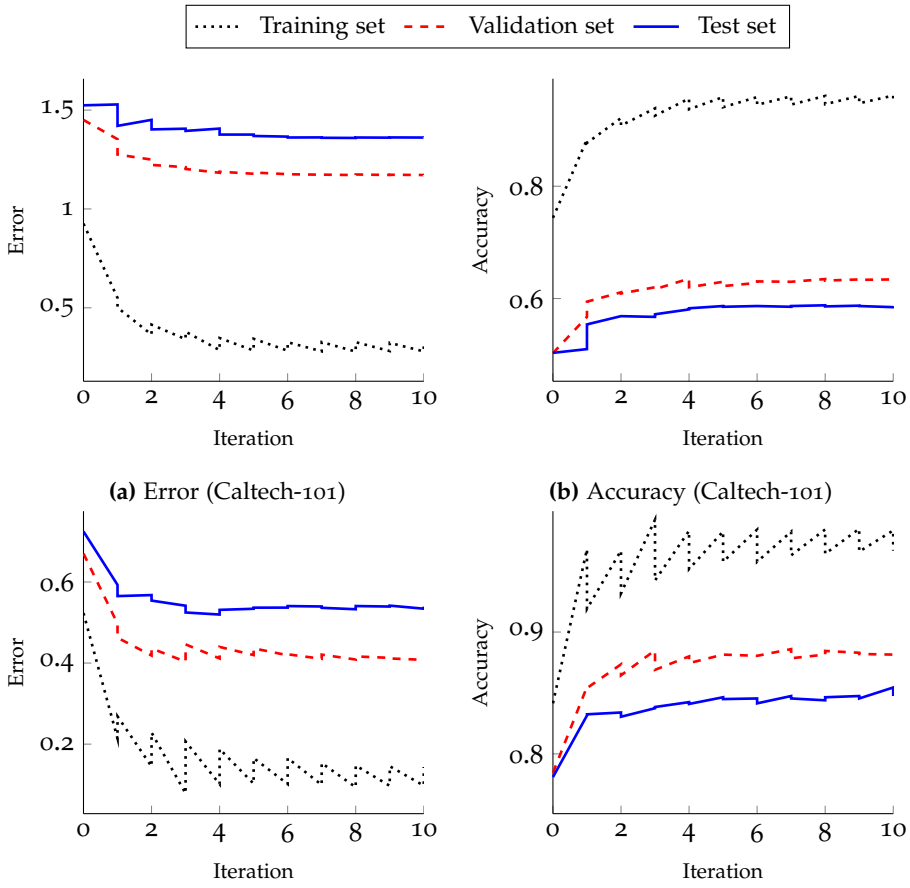
During the previous experiments, full backpropagation was applied to a system where the neural-SIFT network was initialized on SIFT descriptor output. In the following experiment, a randomly initialized neural-SIFT network is used instead, termed the neural-RANDOM network, to see how much influence initialization has on performance. Can the system learn to extract good features without initializing on an existing feature descriptor when using full backpropagation training, or is it a crucial step in the process?

4.4.1 System settings

Almost all settings used for this system are identical to the neural-SIFT based system. The only aspect that changed is the initialization of the neural-SIFT network. The neural-SIFT network was pretrained on SIFT descriptor output. However, in this case, the weights will only be initialized with random values. When the weights are initialized in a small range (e.g., $[-0.1, +0.1]$), regularization will have too much influence when applying full backpropagation. The weights will be pushed too hard towards zero when λ is set to the same value used in previous settings. Using the Nguyen-Widrow initialization method, it was observed that the system had a hard time obtaining a reasonable level of performance when applying full backpropagation. Therefore, weights were randomly initialized in a much broader range, namely $[-0.5, +0.5]$.

4.4.2 Results without full backpropagation

Performance of both the 100 and 300 clusters setting without full backpropagation are shown in Table 4.11. In each setting the randomly initialized neural network performs less well than the network initialized on SIFT descriptor output. This is not surprising as the SIFT descriptor has shown to be a well performing local image descriptor [24] and the random neural network has no theoretical basis whatsoever.



(a) Error (Caltech-101)

(b) Accuracy (Caltech-101)

(c) Error (Corel-1k)

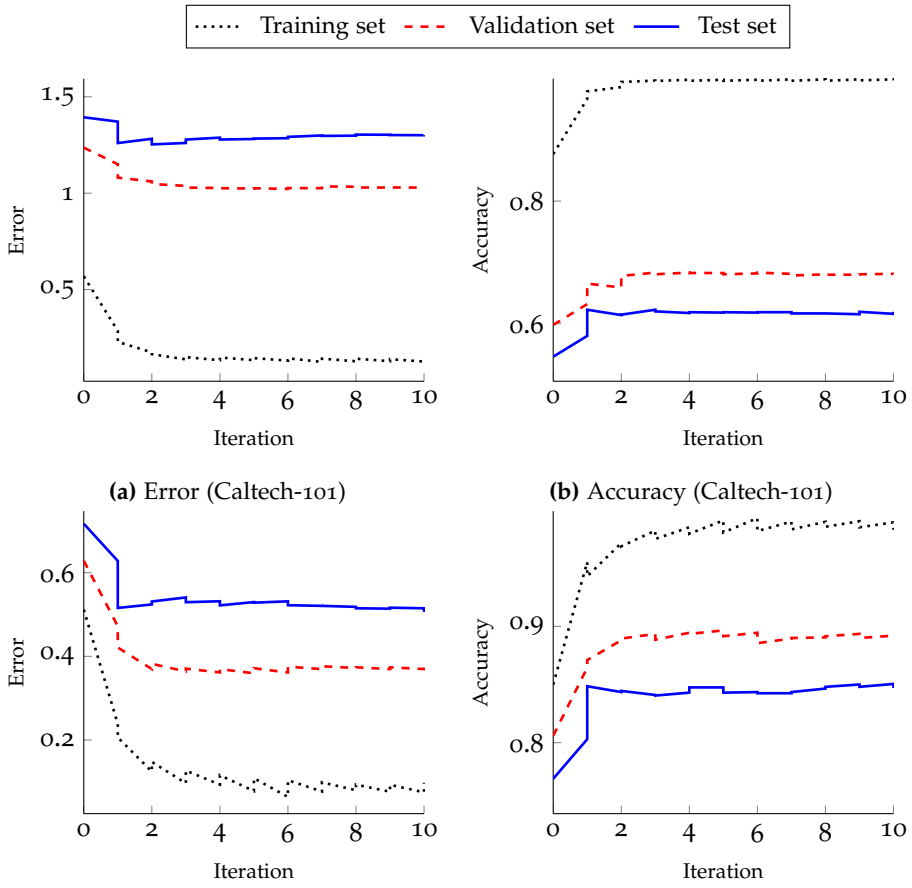
(d) Accuracy (Corel-1k)

Figure 4.9: Full backpropagation progress with a random initialized local image descriptor network using k = 100 clusters. Error and accuracy are averaged over 10 folds. **(a)** and **(b)** show the progress for the Caltech-101 dataset, the progress for the Corel-1k dataset is shown in **(c)** and **(d)**. Vertical updates indicate the error and accuracy after retraining the clusters and the neural classifier.

4.4.3 Full backpropagation training

Full backpropagation is applied to the system and results for the 100 clusters setting are shown in Figure 4.9. For the Caltech-101 dataset the validation error stabilized after around iteration 4. After that, the error went down just a little, but no big decrease was observed further on. The lowest error was observed at iteration 10. The lowest validation error for the Corel-1k dataset was observed at iteration 3, before retraining the clusters and the neural classifier. At this point, the training set error and accuracy showed some fluctuating behavior, but did not affect the other image sets that much. The highest decrease in error and increase in accuracy was observed at iteration 1.

The training results for the k = 300 clusters setting are shown in Figure 4.10. Similar to the full backpropagation progress of the 100 clusters setting, the progress for the 300 clusters setting for the Caltech-101 dataset shows a quick decrease in error for the first iteration. After that, it improves just a little bit up to iteration 6 after which the validation error increased just a little bit. The validation error of the Corel-1k dataset reached its lowest point at iteration 5, before retraining the clustering and the neural classifier.



(a) Error (Caltech-101)

(b) Accuracy (Caltech-101)

(c) Error (Corel-1k)

(d) Accuracy (Corel-1k)

Figure 4.10: Full backpropagation progress with a random initialized local image descriptor network using $k = 300$ clusters. Error and accuracy are averaged over 10 folds. **(a)** and **(b)** show the progress for the Caltech-101 dataset, the progress for the Corel-1k dataset is shown in **(c)** and **(d)**. Vertical updates indicate the error and accuracy after retraining the clusters and the neural classifier.

4.4.4 Improved neural-RANDOM versus improved neural-SIFT

A summary of the results thus far is shown in Table 4.12. After applying full backpropagation training, only one of the settings, the Caltech-101 dataset with 100 clusters setting, outperformed the SIFT descriptor. However, in each setting the neural-RANDOM network loses from its neural-SIFT network counterpart performance-wise. The difference is larger for the Caltech-101 dataset than for the Corel-1k dataset. Adding more training images seems to have a positive effect on the learning capabilities of the system through full backpropagation. Although a properly initialized local image descriptor neural network gives a head start performance-wise, if there are enough train images available the system can recover quite well using full backpropagation. In other words, it can come up with its own representation of what a good image feature looks like. For the Corel-1k dataset, for example, it even outperformed the SIFT descriptor without any prior knowledge of how to describe a local part of an image.

Table 4.12: Average classification accuracy of the test set over 10 folds for using either the SIFT descriptor, the improved neural-SIFT network, the randomly initialized neural-RANDOM network, or the improved neural-RANDOM network as local image descriptor. Numbers in parentheses show the difference in performance when compared to the SIFT descriptor.

	100 clusters			
	SIFT	**Neural-SIFT**	**Neural-RANDOM**	
	descriptor	**Improved**	**Original**	**Improved**
Caltech-101	62.80%	66.13%	50.33%	58.73%
		(+3.33%)	(−12.47%)	(−4.07%)
Corel-1k	83.15%	86.15%	78.10%	83.60%
		(+3.00%)	(−5.05%)	(+0.45%)
	300 clusters			
	SIFT	**Neural-SIFT**	**Neural-RANDOM**	
	descriptor	**Improved**	**Original**	**Improved**
Caltech-101	65.33%	70.20%	54.59%	61.80%
		(+4.87%)	(−10.74%)	(−3.53%)
Corel-1k	86.25%	86.85%	76.90%	84.70%
		(+0.60%)	(−9.35%)	(−1.50%)

4.5 Generalizability

In order to test the generalizability of the improved neural-SIFT descriptor, performance is measured with a different classifier. This classifier is trained using the original neural-SIFT network and the corresponding clusters in the first setting (i.e., without full backpropagation applied). In the second setting it uses the improved neural-SIFT network and the corresponding clusters (i.e., after applying full backpropagation). This experiment is applied for both $k = 100$ and $k = 300$ clusters.

4.5.1 SVM classifier

The Support Vector Machine (SVM) [9] is used as the new classifier (see Section 2.5). The SVM classifier has been applied in numerous studies and is known for its good performance with high dimensional input data. As this system deals with high dimensional input for the classifier, the SVM classifier is well suited for the task at hand.

The radial basis function (RBF) is used as kernel function. This function involves tuning two important parameters: C and γ (see Section 2.5.2 for a brief explanation of the two). A grid-search algorithm is applied to fine-tune the two parameters. First, a coarse grid search is used with C-values of $2^{-5}, 2^{-4}, ..., 2^{15}$ and γ-values of $2^{-15}, 2^{-14}, ..., 2^{3}$. The best performing parameters, denoted C^* and γ^* for C and γ respectively, are used as the starting point for a fine grid-search. In this search, C ranges from $0.5C^*$ to $2C^*$ and γ ranges from $0.5\gamma^*$ to $2\gamma^*$, where each range is divided in 20 equal steps. The best parameters of the fine grid-search are used to test performance of the system.

Table 4.13: Average classification accuracy of the test set over 5 folds for using either the neural classifier or the SVM classifier with the original or improved neural-SIFT descriptor. Numbers in parentheses show the difference in performance when compared to the original neural-SIFT descriptor for the same classifier.

	100 clusters			
	Neural classifier		**SVM classifier**	
	Original	**Improved**	**Original**	**Improved**
Caltech-101	62.47%	66.13% (+3.66%)	75.90%	77.08% (+1.18%)
Corel-1k	81.65%	86.15% (+4.50%)	90.10%	90.55% (+0.45%)
	300 clusters			
	Neural classifier		**SVM classifier**	
	Original	**Improved**	**Original**	**Improved**
Caltech-101	68.53%	70.20% (+1.67%)	79.17%	79.72% (+0.55%)
Corel-1k	85.10%	86.85% (+1.75%)	90.25%	91.40% (+1.15%)

4.5.2 Results

The results for this experiment are shown in Table 4.13. In the second and third column the results are again presented for the neural classifier. The fourth column shows the results while using the SVM classifier with the original neural-SIFT descriptor. For this setting, the SVM performed better than the neural classifier with the original neural-SIFT network for each number of clusters and dataset. Especially for the Caltech-101 dataset the SVM performed a lot better: +13.43% for the 100 clusters setting and +10.64% for using 300 clusters.

When training the SVM with the improved neural-SIFT network instead, the performance for each setting improved. The improved neural-SIFT descriptor not only led to an improvement for the neural classifier by which it was trained, but it also led to an improvement when using a completely different classifier. A classifier which has little or no structural resemblance to a neural network. Although the performance increases are small, this shows that the full backpropagation system has the capability to improve a feature descriptor that can then be used in other settings than the one in which it was trained.

5 Conclusion & further work

In Chapter 4, experiments were set up to answer the research questions compiled in Section 1.2. Section 5.1 reflects on the obtained results by answering the research questions one by one. In Section 5.2, possible areas of explorations and improvements are suggested for the presented framework and Section 5.3 concludes this thesis.

5.1 Research questions

The research questions are answered in the same order as they were compiled, starting with the full backpropagation training scheme in general. Next, initialization of the system will be examined, and lastly, generalizability will be the subject of discussion.

5.1.1 Full backpropagation

Can training the feature descriptor based on the classification error be able to improve recognition results?

A training scheme was proposed and constructed, capable of propagating the classification error all the way back to the feature extracting neural-SIFT network. This neural-SIFT network was initialized on SIFT descriptor output and was capable of achieving a good level of performance when compared to using a randomly initialized neural network. However, the performance when using the neural-SIFT network was worse in three out of four settings than when using the SIFT descriptor itself.

When full backpropagation was added to the pipeline, the performance of the whole system improved for all settings. Compared to the original neural-SIFT network improvements were obtained of 3.66% and 1.67% for the Caltech-101 dataset and of 4.50% and 1.75% for the Corel-1k dataset, using either 100 or 300 clusters for the visual vocabulary, respectively. That the improvements for the Corel-1k were higher than for the Caltech-101 dataset is not surprising, as the number of training samples for the former are a lot higher (60 versus 15 training images per class).

Multiple iterations of full backpropagation showed to be able to boost the system further than using just a single iteration. In most cases, the first iteration of backpropagation together with the retraining of the clustering and the neural classifier showed the highest performance increase. After about 10 iterations of training, either the validation error stabilized or showed an ever increasing trend, at which point training was halted. At these points, the training errors were already really low which could lead to overfitting on the train data more easily.

In the end, the improved neural-SIFT descriptor clearly outperformed the SIFT descriptor on which the system was initially trained. For the Caltech-

101 dataset the improved neural-SIFT showed a significant difference in performance of +3.33% and +4.78%. For the Corel-1k this was +3.00% and +0.60%. For both settings concerning the Corel-1k dataset, the difference in performance was shown to be almost significant at the $p = 0.050$ level ($p = 0.054$).

From the results mentioned above we conclude that using the classification error the system is capable of improving the feature descriptor and as a consequence improve recognition accuracy. Not even was the system capable of improving the original neural-SIFT feature descriptor, it also achieved higher performance than the original feature descriptor on which it was trained in two out of four settings.

5.1.2 Initialization

Can the system come up with a good feature descriptor without initializing on an existing one?

The experiments involved in answering the first research question all used a neural network feature descriptor initialized on the widely used SIFT descriptor, with the task of improving it. The question that was asked subsequently was: What would happen if this initialization process would not take place? Would the system be able to learn to extract good features on its own, or does the system initially need to be pushed in some predefined direction?

To answer these questions, the neural network used as feature descriptor (i.e., the neural-SIFT network) was not trained on SIFT descriptor output anymore. The only procedure applied was random weight initialization in a certain range. The resulting network was termed the neural-RANDOM network, in a similar way as the network initialized on SIFT descriptor output was termed the neural-SIFT network. The experiments conducted were identical to the ones using the neural-SIFT network.

Without applying full backpropagation training, the system's performance was clearly worse than that of using either the SIFT or the original neural-SIFT descriptor. For using a vocabulary size of 100 (i.e., 100 clusters) the achieved accuracy was 12.14% and 3.55% lower than that of the original neural-SIFT network for the Caltech-101 and Corel-1k dataset, respectively. Looking at a vocabulary size of 300 the performance difference was even higher: 13.94% for the Caltech-101 dataset and 8.20% for the Corel-1k dataset. It seems that when using more clusters a random initialized network causes a higher relative decrease in performance.

When full backpropagation training was added, the performance increase of the neural-RANDOM network was considerable. For the 100 clusters setting the system showed an increase of +8.40% and +5.50% for the Caltech-101 and Corel-1k dataset, respectively. An increase of +7.21% and +7.85% was observed for the 300 clusters setting. In most settings, the first iteration with the retraining of the clusters and the neural classifier resulted in the largest decrease in error and the largest increase in accuracy, as was the case with training the neural-SIFT network. In all cases, the performance stabilized already quite soon after about three or four iterations.

Comparing the results of the improved neural-RANDOM network with the improved neural-SIFT network, the difference in performance is still in favor of the neural-SIFT network, although the gap did decrease. Notably, the difference is much higher for the Caltech-101 dataset (-7.40% and -8.40%) than for the Corel-1k dataset (-2.55% and -2.15%). It seems that when the system has

a lot of training images at its disposal the quality of the feature descriptor can increase to almost the same heights as when using the neural-SIFT network. Although the difference was small, for one setting the system even achieved a higher performance than when using the SIFT descriptor (+0.45%).

Given the availability of a high number of training images, we conclude that the system is able to achieve high performance using a random initialized feature descriptor. This means that the system does not need to have any prior knowledge of what a good feature descriptor looks like, it only needs the classifier to tell what is present in each image. When dealing with a low number of training images the system does benefit greatly from initialization on an existing feature descriptor. For using a high number of training images this is the case as well, as it gives a large head start performance-wise.

5.1.3 Generalizability

Is the improved feature descriptor generalizable to other recognition systems?

The improved neural-SIFT descriptor showed not only higher performance compared to the original neural-SIFT descriptor, but also showed to be superior to the SIFT descriptor itself in the context in which it was improved. The question that arose was: Is the improved neural-SIFT descriptor, besides being superior in the current context, also superior in other contexts? To answer this question, a small experiment was set up that tested the performance of both descriptors while using a different classifier than the neural classifier, in this case an SVM.

Using an SVM with the original neural-SIFT descriptor resulted in a higher performance than with using the neural classifier in each single setting. For the Caltech-101 dataset the difference in performance was a lot higher (+13.43% and +10.64% for the 100 and 300 clusters setting, respectively) than for the Corel-1k dataset (similarly, +8.45% and +5.15% increase). Remarkably, in each setting the SVM with the original neural-SIFT descriptor scored higher than the neural classifier with the improved neural-SIFT descriptor. When the improved neural-SIFT descriptor was used with the SVM, the performance increased for each setting. The differences compared to the SVM with the original neural-SIFT descriptor were small, ranging between +0.45% and +1.18%.

The results obtained in this experiment indicate that the neural-SIFT descriptor, which was improved using a neural classification system, does generalize to other systems, in this case the SVM. The improvements made, however, were small and were only shown with one other type of classifier. Consequently, these results should be interpreted with care. It is, therefore, hard to draw firm conclusions in terms of generalizability to other systems, but the improvements do show potential.

5.2 Future work

Research is a never ending process, which applies to constructing the framework presented in this thesis as well. The proposed training scheme was a first step in improving feature descriptors that are currently available using classifier feedback, and many aspects exist which have not yet been touched upon. Additionally, many adjustments to the system can be made that can possibly lead to improved performance.

In Section 5.2.1, possible areas of exploration are suggested for the system in its current form. Section 5.2.2 provides modifications to the system that are worthy of consideration.

5.2.1 Exploration

The system comes with many tunable parameters. For a multilayer neural network there are already many, including: the number of hidden layers and units in those hidden layers; the type of activation function and the steepness for that function to use in each layer; the connectivity between layers; the learning algorithm; and many others. It's nearly impossible to go through all configurations and test which one performs best.

The goal of this thesis was not to attain the highest performance possible, but rather to obtain higher performance than with using the original feature descriptor. Therefore, the effects of a lot of parameters were explored in little detail, or not at all. For example, the parameters for the fixed partitioning approach. The fixed partitioning was implemented using a sliding window approach, where the number of pixels to shift can be changed. By decreasing this value a lot more different image patches are fed to the system. Also the size of the image patches can be changed to capture more or less pixels.

An important parameter is the number of hidden units to use in the neural-SIFT network. Both while training the network on SIFT descriptor output as while applying full backpropagation training, overfitting can occur. Overfitting is more likely to occur when there are more units and connections, so setting it to a low value is recommended. The downside is that this decreases the degree of fit with the SIFT descriptor. A more thorough parameter sweep can be applied to get a better understanding of the influence of this parameter.

To further test the full backpropagation pipeline, different local image descriptors could be used. Now, only the SIFT descriptor was shown to be improved by it, other feature descriptors (e.g., SURF and others) can be cast into this framework as well. Virtually any feature descriptor that takes a window of image pixels as input can be used and tested.

For creating the image histogram only two different visual vocabulary sizes were considered, being 100 and 300 visual words (clusters). In each setting, being it training the neural-SIFT network, the neural-RANDOM network, or using an SVM as classifier, the 300 clusters setting showed higher performance than the 100 clusters setting. The number of clusters to use thus has a clear influence on accuracy. Optimizing the number of clusters to use seems like an evident step in improving recognition accuracy for this system.

Lastly, in this thesis, generalizability of the improved neural-SIFT descriptor has only been tested using the SVM classifier using the same bag of visual words approach. To really get a grip on the generalizability of the improved feature descriptor, the descriptor could be tested with additional classifiers and using different pipelines than the currently used bag of visual words approach.

5.2.2 Modifications

Like the suggested areas of exploration, there are numerous ways of changing parts of the system that can possibly lead to an improved performance. A few modifications to the system are presented here.

For the neural-SIFT network, a simple feedforward multilayer perceptron was used. Other types of networks can be used instead, for example, convolutional neural networks [25, 27]. These networks are biologically inspired by the visual cortex, where the cells in the cortex are only sensitive to specific regions of the visual field. Convolutional networks use the same concept and consist of multiple layers and use different types of layers. The most noteworthy type of layer is the convolutional layer which consists of units taking a rectangular subsection of the previous layer as input. The weights for each unit in such a layer are shared. Convolutional neural networks have received a lot of attention lately in the academic world and are shown to give very promising results (e.g., in [25]). Implementing them in the presented framework could give a possible boost in performance.

To construct the visual vocabulary, the k-means algorithm was applied as clustering technique. Usually, k-means uses the hard assignment method where a sample instance only belongs to one single cluster. To make this method differentiable the distances to each cluster were transformed to similarity values. Instead of using this approach other techniques exist which fit the distribution of data instances using probability density functions. When comparing a test sample with such a distribution the relative likelihood of the distribution generating this sample is returned. These values can be interpreted as how similar an instance is to a cluster, and therefore these techniques are interesting to apply in this system. An example of such a technique is the Gaussian mixture model (GMM) where the Gaussian distribution function is used as density function (e.g., used in [17, 37]). This technique was first attempted for the proposed system, but turned out to be too computational expensive in the current setting when training. A GMM uses a $d \times d$ covariance matrix for each centroid to describe the distribution, where d is the dimensionality. d in this system equals the size of the SIFT descriptor, which is 128. The algorithm involved in computing these big covariance matrices involve a lot of computation and, as a consequence of the high dimensionality, introduced computational under- and overflow problems. However, when a solution can be found to the under- and overflow problems, for example, when using descriptors with a lower dimensionality, GMMs can certainly be applied.

Abdullah et al. provided an overview of multiple methods for constructing image histograms [2]. One of the simplest being the hard bag-of-features approach where the hard assignment method for clusters is applied and the number of cluster occurrences is counted. Another novel method, termed the MIN approach, was proposed which involves computing a minimum distance map for each visual word in the vocabulary [2]. This approach showed to be able to outperform other histogram methods. Exploring the different approaches in this system could possible lead to improved performance as well.

Each of these histogram construction methods, however, ignore any spatial information present in the image. The fact that specific extracted features are close to each other in the image can be very relevant for a specific object and can help the classifier in the decision process. To get some form of spatial information back in the system, the spatial pyramids approach was introduced [18, 26]. For this approach, a sample image is divided in multiple regions, possibly at different resolutions. For each of these regions a histogram is constructed and these are combined in a later stage. Applying the spatial pyramid approach has shown to outperform the single histogram approach in many studies (e.g., in [2, 7, 26]) and can be applied in the system proposed in this thesis as well. Full backpropagation will need to be adapted to distribute the error over the

available region histograms, which is a similar approach as distributing the error over the multiple histograms for each image patch, already incorporated.

The last stage in the proposed system is classification. As with the neural-SIFT network, a feedforward multilayer perceptron was used here as well. In the generalizability experiment an SVM classifier was used for testing. In each setting the SVM outperformed the neural classifier. This shows that the choice of classifier to use is an important one in terms of performance. The full back-propagation approach can be applied to any type of classifier as long as the classification error can be propagated back to the inputs.

5.3 Conclusion

In this thesis, a deep neural network training framework was introduced which propagates the classification error through the entire pipeline all the way back to the feature extraction stage to learn to extract 'better' features than the initial local image descriptor. A neural network was trained on the SIFT descriptor (termed neural-SIFT) and by using the proposed full backpropagation training scheme the system was able to achieve higher recognition accuracy in each single setting. Performance increases between 0.60% and 4.87% were realized when compared to the SIFT descriptor.

The influence of initializing the feature extracting neural network on an existing local image descriptor was examined by using a randomly initialized network instead. The system showed a much lower performance initially, but was able to narrow the performance-gap quite a bit after applying full back-propagation training. This gap was a lot smaller for the Corel-1k dataset where a lot more training images were available than for the subset of classes used in the Caltech-101 dataset. This shows that, when a system has many training samples available, it can figure out by itself, without any prior knowledge on the subject, how a distinctive image descriptor should look like.

The image descriptor was improved in one specific context, with the bag of visual words approach and a neural network classifier. To see if the improved image descriptor not only led to an increased performance in the context in which it was trained, both the original and the improved descriptor were tested in another context as well. An SVM classifier was used for this purpose and in each setting the performance increased when using the improved neural-SIFT descriptor instead of the original. However, these improvements were small and were only achieved in one other context. Other contexts should be tested to get a better view on the generalizability of the improved descriptor.

The proposed system was a first step in improving feature descriptors based on classifier feedback. The system can be modified and improved upon in numerous ways and the influence of a large portion of parameters were examined in little detail, or not at all. Other feature descriptors, or combinations of them, can be applied, different image histogram creation methods can be used, and any type of classifier which allows the error to be propagated back to the input can be applied. The possibilities are endless.

Bibliography

[1] A. Abdullah, R. C. Veltkamp, and M. A. Wiering. Spatial pyramids and two-layer stacking svm classifiers for image categorization: A comparative study. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 5–12. IEEE, 2009.

[2] A. Abdullah, R. Veltkamp, and M. Wiering. Ensembles of novel visual keywords descriptors for image categorization. In *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference*, pages 1206–1211. IEEE, 2010.

[3] A. Abdullah, R. Veltkamp, and M. Wiering. Fixed partitioning and salient points with MPEG-7 cluster correlograms for image categorization. *Pattern Recognition*, 43(3):650–662, 2010.

[4] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[5] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.

[6] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision*, pages 404–417. Springer, 2006.

[7] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. In *Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 401–408. ACM, 2007.

[8] T. Botterill, S. Mills, and R. Green. Speeded-up bag-of-words algorithm for robot localisation through scene recognition. In *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*, pages 1–6. IEEE, 2008.

[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

[10] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Proceedings of the European Conference on Computer Vision, Workshop on statistical learning in computer vision*, volume 1, page 22, 2004.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[12] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1403–1410. IEEE, 2003.

[13] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[14] J. S. Denker, R. E. Howard, L. D. Jackel, and Y. LeCun. Hierarchical constrained automatic learning neural network for character recognition, Sept. 10 1991. US Patent 5,067,164.

[15] M. Egmont-Petersen, D. de Ridder, and H. Handels. Image processing with neural networks a review. *Pattern recognition*, 35(10):2279–2301, 2002.

[16] C. Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the Internation Conference on Machine Learning*, volume 3, pages 147–153, 2003.

[17] J. Farquhar, S. Szedmak, H. Meng, and J. Shawe-Taylor. Improving "bag-of-keypoints" image categorisation: Generative models and pdf-kernels. Technical report, University of Southampton, 2005.

[18] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1458–1465. IEEE, 2005.

[19] E. Hadjidemetriou, M. D. Grossberg, and S. K. Nayar. Spatial information in multiresolution histograms. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–702. IEEE, 2001.

[20] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[21] C. Igel and M. Hüsken. Improving the Rprop learning algorithm. In *Proceedings of the Second International Symposium on Neural Computation, International Conference on Semantic Computing*, volume 2000, pages 115–121. ICSC Academic Press, 2000.

[22] Y.-G. Jiang, C.-W. Ngo, and J. Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 494–501. ACM, 2007.

[23] T. Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. Technical report, DTIC Document, 1996.

[24] L. Juan and O. Gwun. A comparison of SIFT, PCA-SIFT and SURF. *International Journal of Image Processing (IJIP)*, 3(4):143–152, 2009.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[26] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference*, volume 2, pages 2169–2178. IEEE, 2006.

[27] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995.

[28] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[29] D. D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Proceedings of the European Conference on Machine Learning*, pages 4–15. Springer, 1998.

[30] D. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[31] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. California, USA, 1967.

[32] J. L. McClelland, D. E. Rumelhart, P. R. Group, et al. Parallel distributed processing. *Explorations in the microstructure of cognition*, 2, 1986.

[33] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 27(10): 1615–1630, 2005.

[34] F. Monay, P. Quelhas, D. Gatica-Perez, and J. Odobez. Constructing visual models with a latent space approach. In *Subspace, Latent Structure and Feature Selection*, pages 115–126. Springer, 2006.

[35] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 21–26. IEEE, 1990.

[36] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[37] F. Perronnin, C. Dance, G. Csurka, and M. Bressan. Adapted vocabularies for generic visual categorization. In *Proceedings of the European Conference on Computer Vision*, pages 464–475. Springer, 2006.

[38] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[39] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[40] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1470–1477. IEEE, 2003.

[41] V. Srinivasan, P. Bhatia, and S. Ong. Edge detection using a neural network. *Pattern Recognition*, 27(12):1653–1662, 1994.

[42] M. Swain and D. Ballard. Color indexing. *International journal of computer vision*, 7(1):11–32, 1991.

[43] D.-M. Tsai. Boundary-based corner detection using neural networks. *Pattern Recognition*, 30(1):85–97, 1997.

[44] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends® in Computer Graphics and Vision*, 3(3):177–280, 2008.

[45] J. C. van Gemert, C. J. Veenman, A. W. Smeulders, and J.-M. Geusebroek. Visual word ambiguity. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(7):1271–1283, 2010.

[46] D. Yu and L. Deng. Deep learning and its applications to signal and information processing. *Signal Processing Magazine, IEEE*, 28(1):145–154, 2011.

[47] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *International journal of computer vision*, 73(2):213–238, 2007.