Learning and Adaptivity Report: Convolutional Neural Networks and Sparse Coding for Image Classification of RoboCup@Home and RoboCup@Work Objects

Octavio Arriaga, Nitish Koripalli, Maximilian Schoebel *BRS University of Applied Sciences*
email:
octavio.arriaga@smail.inf.h-brs.de
maximilian.schoebel@smail.inf.h-brs.de
nitish.koripalli@smail.inf.h-brs.de

June 13, 2016

Object recognition is an actively researched field in machine learning. Two underlying approaches that can be used for robust object recognition algorithms are hierarchical and sparse feature extraction. In this work we explore convolutional neural networks and sparse coding which embody these properties respectively. We test their object recognition performance in a structured and industrial environment: RoboCup@Work and Robocup@Home.

1 Introduction

Convolutional Neural Networks

Deep learning techniques yield state-of-the-art results for object detection and recognition; specifically Convolutional Neural Networks (CNNs) have proven to be extremely useful for image classification. Therefore, we introduce our own deep architecture model, and train it to classify objects from the RoboCup@Home competition.

In order to understand the CNN used for our training example, first we would have to define the discrete convolution operator for 2-Dimensions.

$$S(i,j) = (I*k)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{1}$$

This equations can be thought of as performing an discrete-operation to all possible values of an input I. This can also be seen as the application of a filter to an image-Matrix I, see Figure 1.

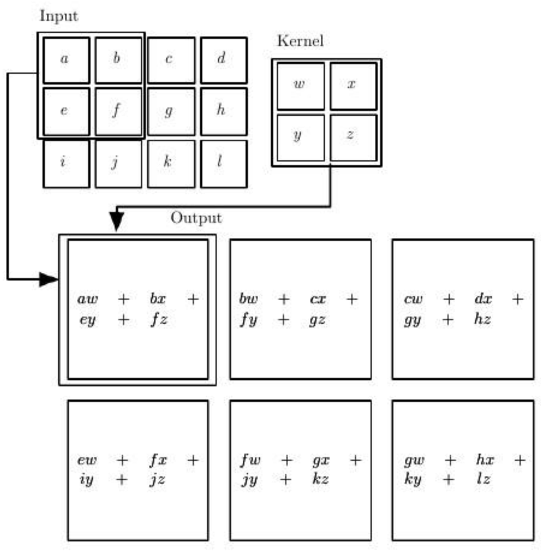

Figure 1: *Discrete convolution applied to an input-matrix*

The input-matrix can represent an image, and the output is a matrix-like multiplication to a section of the input image which is the size of the kernel. This operation is repeated for all possible input values of the image.

In the typical CNN network terminology, the first argument is often refered to as the *input*, the second argument is the *kernel*, and the output by applying the convolution, is the *feature map*.

As can be seen in Figure 1, each output pixel results from applying the kernel to a small patch of the input image. These patches do overlap however and do can be bigger then in Figure 1. Several kernels are trained and applied to the input image and thereby form the first convolutional layer.

The general Convolutional Network architecture consists of several convolutional layers stacked over each other, each of them followed by a pooling layer. Pooling layers perform, depending on their kind, a combination of a small output patch of the previous convolutional layer. Following figure 2 shows a simple example of max-pooling, where only the maximum value of the convolutional layer's output is passed on as the output of the pooling patch.

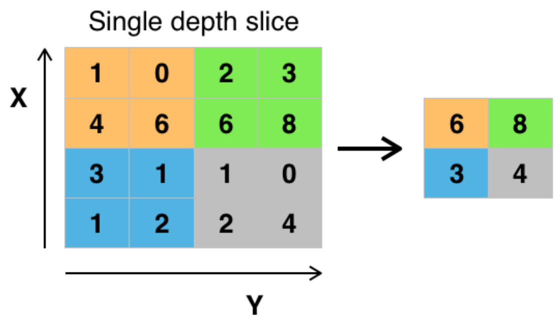

Figure 2: *Max-pooling*

The outputs of the pooling layers form a filtered, lower-dimensional representation of the previous convolutional layer. By stacking pairs of convolutional- and pooling layers, the representations get more abstract with each layer and since each convolution and pooling operation reduces the dimensionality, the upper layers operate on a bigger scope of the input image.

The use of pooling-layers also makes the learned lower-level representations, i.e. features, invariant to small shifts in position. The reduction of dimensionality in depth with each layer is also called subsampling.

The following figure 3 shows the typical architecture of a Convolutional Neural Network.

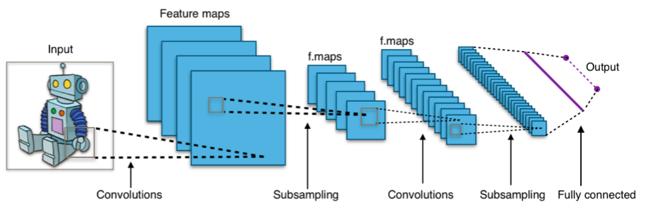

Figure 3: *Typical CNN architecture*

After training several layers of convolution and subsampling (e.g. pooling) learned a fixed size and abstract representation of the input image. The last subsampling layer is then followed by several fully connected layers which act as a classifier to distinguish the image-classes present in the dataset. The complete architecture is then trained end-to-end (including the fully connected layers).

Intermediate CNN Results and Visualization

Before performing any training on our data, we wanted to test the capabilities of the used framework *Keras* on a well tested dataset. Therefore we trained a smaller model with 2D convolutional layers and max-pooling using the MNIST dataset (see Figure 4). In **??** we show the results after the first convolutional layer. This model was trained using the Keras framework on top of Theano.

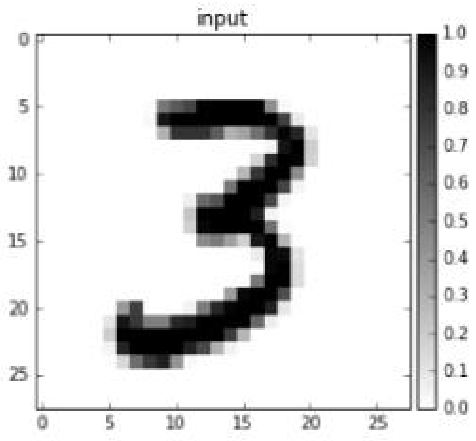

Figure 4: *Image taken from MNIST dataset. Every row and column represents one pixel size.*

The learned 32 kernels after the convolutional layer can be seen in the following image.

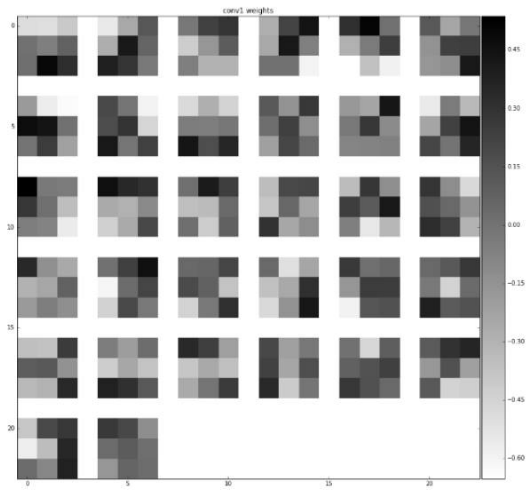

Figure 5: *All 32 kernels learned from the first convolutional layer.*

And the application of these filters or learned kernels to the image presented in 4 is shown below.

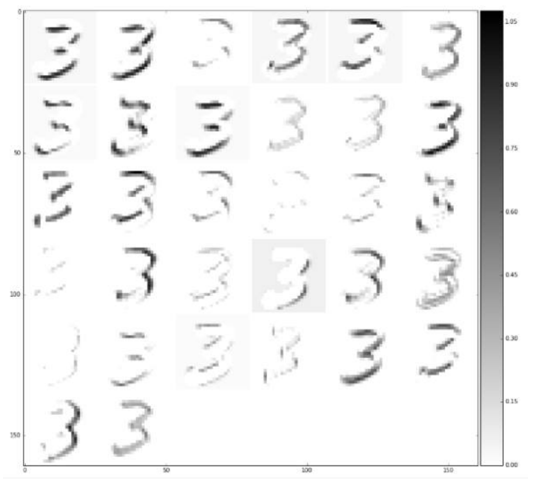

Figure 6: *All 32 Kernels applied to an example of the datase.*

We can observe that just after the first convolution layer, the model is able to distinguish the shape of the contours of the data.

Sparse Coding

Sparse coding is a class of unsupervised methods for learning sets of over-complete bases to represent data efficiently. The aim of sparse coding is to find a set of basis vectors ϕ_i such that we can represent an input vector $\mathbf{x}$ as a linear combination of these basis vectors. [1]

$$\mathbf{x} = \sum_{i=1}^{k} \alpha_i \phi_i$$

While techniques such as Principal Component Analysis (PCA) allow us to learn a complete set of basis vectors efficiently, we wish to learn an over-complete set of basis vectors to represent input vectors $\mathbf{x} \in \mathbb{R}^n$ (i.e. such that k > n). The advantage of having an over-complete basis is that our basis vectors are better able to capture structures and patterns inherent in the input data. However, with an over-complete basis, the coefficients α_i are no longer uniquely determined by the input vector $\mathbf{x}$. Therefore, in sparse coding, we introduce the additional criterion of **sparsity** to resolve the degeneracy introduced by over-completeness.

$$\text{minimize}_{a_i^{(j)},\phi_i} \sum_{j=1}^{m} \left\| \mathbf{x}^{(j)} - \sum_{i=1}^{k} a_i^{(j)} \phi_i \right\|^2 + \lambda \sum_{i=1}^{k} S(a_i^{(j)})$$

Figure 7: *The cost function used for finding the basis functions and the coefficients α.*

S(.) is a sparsity cost function which penalizes α_i for being far from zero. We can interpret the first term of the sparse coding objective as a reconstruction term which tries to force the algorithm to provide a good representation of $\mathbf{x}$ and the second term as a sparsity penalty which forces our representation of $\mathbf{x}$ to be sparse. The λ is a scaling constant to determine the relative importance of these two contributions. The intuition for sparse coding can be seen as the following [3] :

cost_function = reconstruction_error + sparseness

The typical sparseness constraint is L1 regularization also known as *lasso*. The equation minimizes the cost function in an iterative manner. For one iteration, first the encoding (the extraction of coefficients α) is kept constant and a dictionary ϕ that minimizes the cost function is found and then keeping the dictionary contant the encoding (the extraction of coefficients α) that minimizes the cost function is found.

Intermediate Sparse Coding Results and Visualization

We wanted to test the framework for sparse coding which is *sparsex* [2]. We trained a 400 dimension dictionary on 40000 images of Yann LeCun's MNIST database [4]. Each image is 28x28 pixels. We obtained a result of 89.9 ± 0.5 % accuracy. The visualizations of the data preprocessing, feature extraction and results are shown:

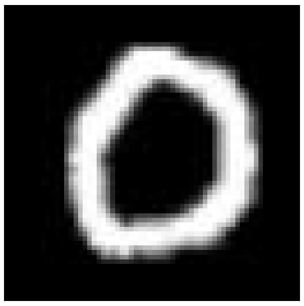

Figure 8: *Image taken from MNIST dataset. Every row and column represents one pixel size. The image size is 28x28 pixels.*

A part of preprocessing the data is patch extraction, where we extract small 8x8 pixel patches from the original 28x28 image covering the entire image. This results in extraction of $(28-8+1)^2$ patches. The motivation behind extraction of patches is to obtain small localized features.

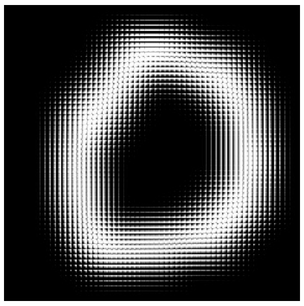

Figure 9: *A montage (i.e. arranging patches in the same location they were extracted from) of the extracted patches. The montage consists of 441 patches extracted from a 28x28 image where each patch is 8x8 pixels.*

Once the patches are extracted for one image, they are preprocessed further by applying normalization and whitening. All patches from one image are whitened together to decorrelate (remove redundancies) from the patches. This makes feature extraction and classification converge faster.

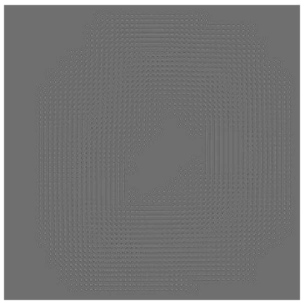

Figure 10: *A montage of whitened patches which is a result of applying whitening or sphering to the patches obtained in figure 9.*

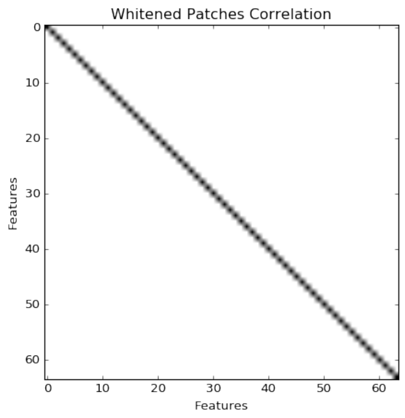

Figure 11: *A correlation plot of the whitened patches which shows decorrelation (reduced redundancy) among the extracted patches. This is expected and ideal behaviour.*

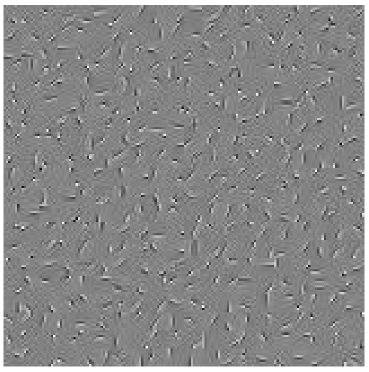

Figure 12: *A montage of the trained dictionary which has 400 elements (20x20) where each element has a dimension of 8x8 pixels. The dictionary elements are similar to the gabor filters.*

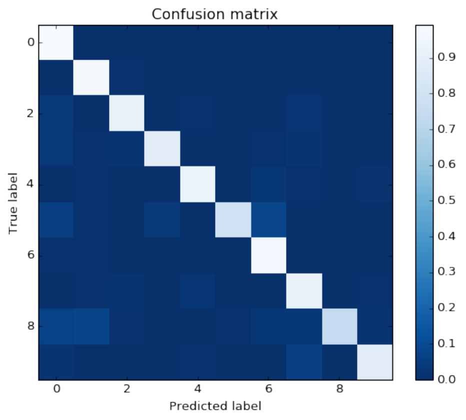

Figure 13: *A normalized confusion matrix of the results of the classification. Diagonal elements show the ratio of samples that were correctly classified, and non diagonal elements show the ratio of samples that were misclassified.*

2 Approach

CNN-Architecture

Our final model consists of 4 convolutional layers, each followed by a max-pooling layer. The first layer takes 32x32 pixels for each color channel (RGB) as inputs, resulting in 3*32*32 input units. After the final max-pooling layer a two layered fully connected network is added as classifier. The final layer uses softmax-activation function, each previous layer uses rectified linear units.

CNN-Training

The model was trained using stochastic gradient descent with a learning-rate of 0.01, weight-decay of 10^{-6} and momentum of 0.9 for 100 epochs. Dropout is used in every layer to reduce overfitting.

The training dataset was splitted into a training-set (80%) and a testing-set (20%).

Sparse Coding

The general approach we used for sparse coding involves the following steps. Each step has its respective citation that describes the implementation and motivation behind that step.

1. Splitting data into training and test set
   a) 5-fold cross validation [11]
2. Preprocessing
   a) Image resizing [5]
   b) Patch extraction [5]
   c) Patch normalization [5]
   d) Patch whitening [5]
3. Feature extraction
   a) Dictionary learning [6, 7]
   b) Sparse encoding [6, 7]
   c) Sparse encoding sign-split
   d) Max-pooling [9, 10]
4. Classification
   a) Standardization of features [12]
   b) Training linear SVM classifier using training data [8]
   c) Classifying the test data [8]

First we split the dataset into training and test sets using 5 fold cross validation. For one experiment we resized the images to 67x67 pixels and extracted 8x8 pixel patches from the images with a stride of 1. We then applied preprocessing techniques like normalization or feature scaling to make the pixel values of the patches be in an appropriate range and applied whitening to decorrelate the patches. We then use the training set patches to learn a dictionary of 625 elements. We use a subsampled set of patches for learning the dictionary because it is a time consuming process and also since many patches repeat themselves in the dataset we do not have a lot of loss of information. Using the learnt dictionary we encode the patches into a sparse weighted sum of basis functions (dictionary elements), which is the feature extraction phase. We then reduce the dimensionality of the data using max pooling. Finally we concatenate the pooled patch encodings of one image into one long feature vector for that image. This feature vector, depending on the pool filter size may have a dimensionality between 5000-32000. We then pass this feature vector as input to the SVM linear classifier for training. We repeat the process above for the test data and obtain predictions from the classifier.

3 Datasets

@Home Objects

For training the final model we collected a datset consisting of a total of 424 Images of the following RoboCup@Home-Objects:

1. Pack of Coffee
2. Paper coffee-cup
3. Juice-box
4. Big ketchup bottle
5. Small ketchup bottle
6. Mr Muscle cleaner
7. Pepper
8. Pringles can
9. Water bottle

@Work Objects

For training the final model we collected a datset consisting of a total of 360 Images of the following RoboCup@Work-Objects [13]:

1. F20 20 B
2. F20 20 G
3. S40 40 B
4. S40 40 G
5. M20 100
6. M20
7. M30
8. R20
9. Bearing Box
10. Bearing
11. Axis
12. Distance Tube
13. Motor

The pictures were taken with a low resolution camera (640x480 pixel) equivalently to the camera mounted on Jenny-COB, from a fixed angle and on a fixed background (white table). The Images were then cropped and scaled to fit the network input size of 32x32 pixels for @Home objects and 67x67 pixels for @Work objects.

4 Results

CNN

After 100 epochs of training, which took approximately ??, the model achieved an accuracy 98.55%.

We took a cluttered picture by accident which has correctly been classified.

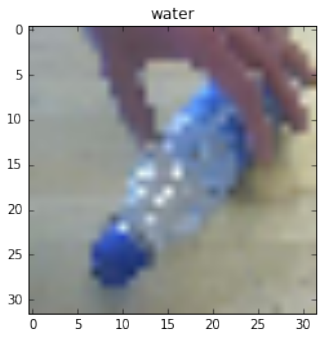

Figure 14: *Cluttered image of the class "Water bottle" with hand of experimenter. The network was able to classify the image correctly.*

When plotting the activations of convolutional layers, the abstraction and reduction in dimensionality that occurs can be made visible.

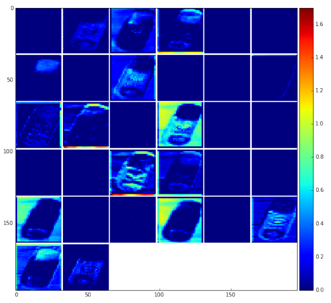

Figure 15: *Output after the first convolutional layer of our network, when presented with an input image.*

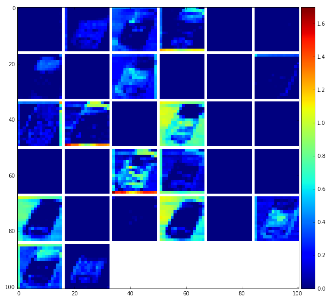

Figure 16: *Output after the second convolutional layer of our network, when presented with an input image.*

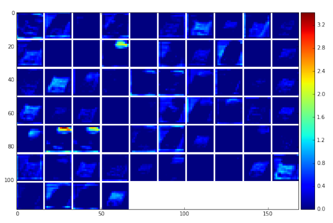
Figure 17: *Output after the third convolutional layer of our network, when presented with an input image.*

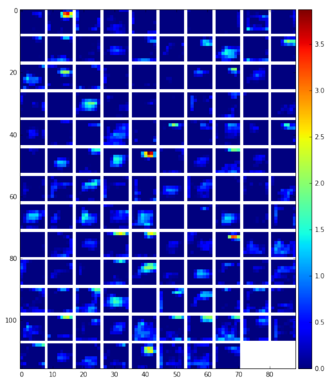
Figure 18: *Output after the fourth convolutional layer of our network, when presented with an input image.*

Our datasets, as well as the implementation of the CNN is available here: `https://github.com/oarriaga/machine_learning_algorithms/tree/master/algorithms/convolutional_neural_network`

Sparse Coding

After tests on 5-fold cross validation, we were able to achieve 71.63 % accuracy on the Robocup@Work dataset. Following are some of the visualizations from the experiments:

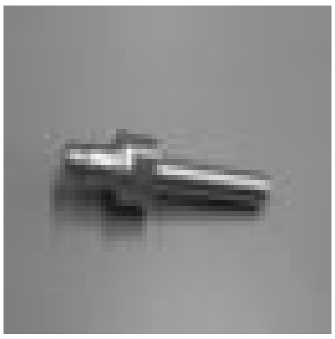
Figure 19: *Image taken from @Work dataset. Every row and column represents one pixel size. The image size is 67x67 pixels.*

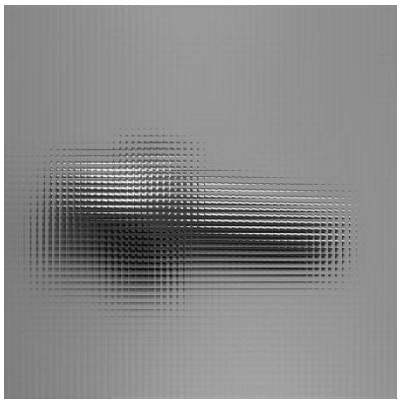
Figure 20: *A montage of 8x8 pixel patches extracted from the original 67x67 image. A total of 3600 patches are extracted per image.*

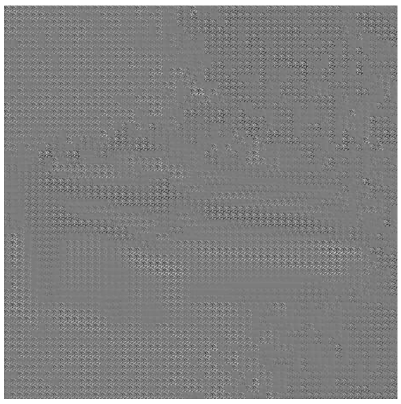
Figure 21: *A montage of whitened patches which is a result of applying whitening or sphering to the patches obtained in the previous figure.*

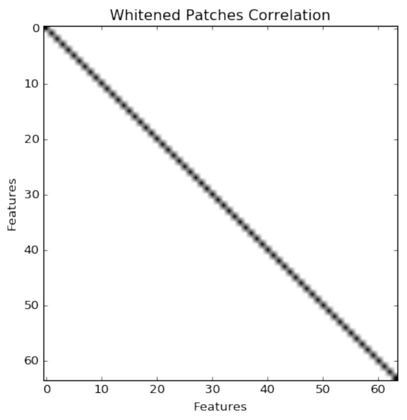

Figure 22: *A correlation plot of the whitened patches which shows decorrelation (reduced redundancy) within the data since the features are correlated only with themselves. This is a correct and expected behaviour.*

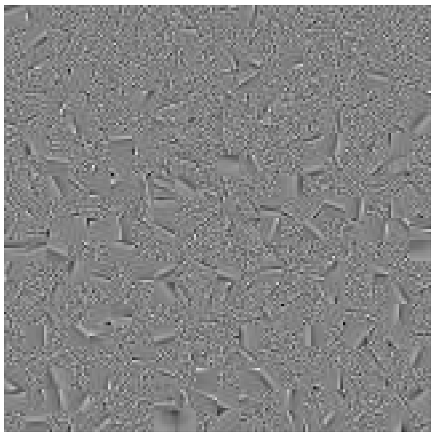

Figure 23: *A montage of the trained dictionary which has 400 elements (20x20) where each element has a dimension of 8x8 pixels. The filters that have been learnt here are not ideal and have a lot of noise as compared to the filters shown for the MNIST dataset in a previous figure.*

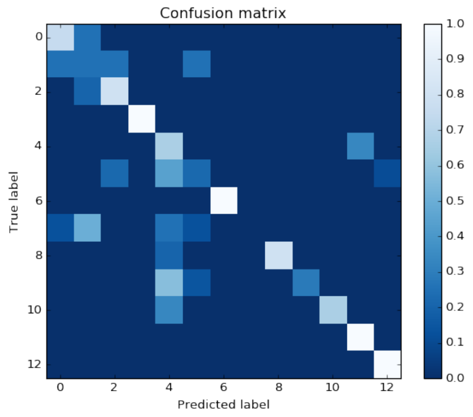

Figure 24: *A confusion matrix of the test vs predicted results which shows no results for label 7 which is object R20. Also we can see a lot of confusion among class labels 0, 1, 2, 4 and 5.*

The implementation of the Sparse Coding is available here: `https://github.com/nitred/sparsex`

5 Conclusion

CNN

We implemented and trained a CNN on a dataset that we created from RoboCup@Home and RoboCup@Work objects. The results were very good, a real system could easily function with a classification accuracy of 98.55%.

In the future several issues should be addressed however. Since the dataset we created was relatively small, overfitting might be an issue. A second dataset could be created with the same objects but diffrent conditions as lighting, background, occlusions and viewing angle. Artificial methods for enlarging the dataset, like applying small transformations could also be used.

Sparse Coding

We implemented and trained a feature extraction using Sparse Coding and classification using SVMs on a dataset that we created from RoboCup@Work objects. The results were insufficient to be used for the Robocup competitions with 71.83 % accuracy.

For the future there are several problems that can be solved. Firstly, the cropping of the images was done manually which caused scaling errors which may have made some objects look like others. Secondly, the *sparsex* implementation currently only supports grayscale images and should support RGB images which would add additional features. Thirdly, the dictionary that was learnt on the training data had a lot of noise in it, therefore in the future the right parameters for dictionary learning must be explored. Lastly, we need to collect a larger dataset so as to extract more meaningful features and have enough data to properly train the classifier.

References

[1] http://ufldl.stanford.edu/wiki/index.php/Sparse_Coding

[2] https://github.com/nitred/sparsex

[3] Olshausen, Bruno A., and David J. Field. *"Sparse coding with an overcomplete basis set: A strategy employed by V1?."* Vision research 37.23 (1997): 3311-3325.

[4] http://yann.lecun.com/exdb/mnist/

[5] Coates, Adam, Andrew Y. Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning." International conference on artificial intelligence and statistics. 2011.

[6] Mairal, Julien, et al. "Online dictionary learning for sparse coding." Proceedings of the 26th annual international conference on machine learning. ACM, 2009.

[7] http://spams-devel.gforge.inria.fr/

[8] http://svmlight.joachims.org/

[9] Yang, Jianchao, et al. "Linear spatial pyramid matching using sparse coding for image classification." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.

[10] Scherer, Dominik, Andreas Mller, and Sven Behnke. "Evaluation of pooling operations in convolutional architectures for object recognition." Artificial Neural NetworksICANN 2010. Springer Berlin Heidelberg, 2010. 92-101.

[11] http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedKFold.html

[12] https://en.wikipedia.org/wiki/Feature_scaling

[13] http://www.robocupatwork.org/rules.html

Appendix B: Results