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

Octavio Arriaga, Nitish Reddy, 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 10, 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.

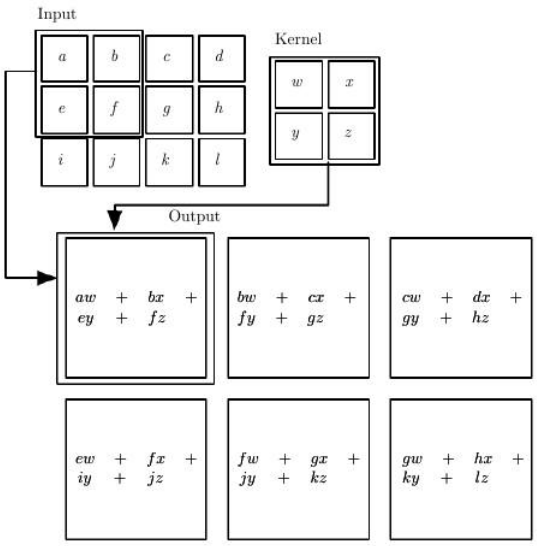

Figure 1: *Discrete convolution applied to an input-matrix*

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.

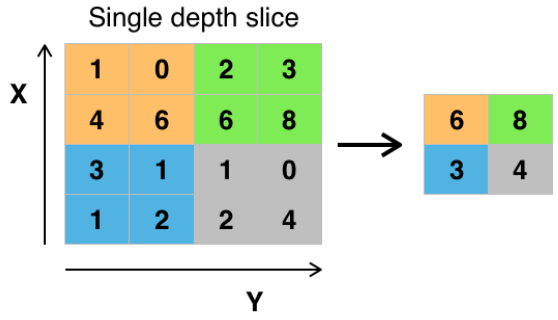

Figure 2: *Max-pooling*

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.

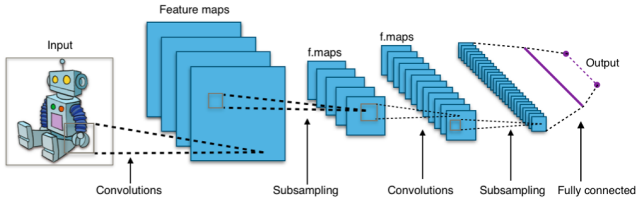

Figure 3: *Typical CNN architecture*

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.

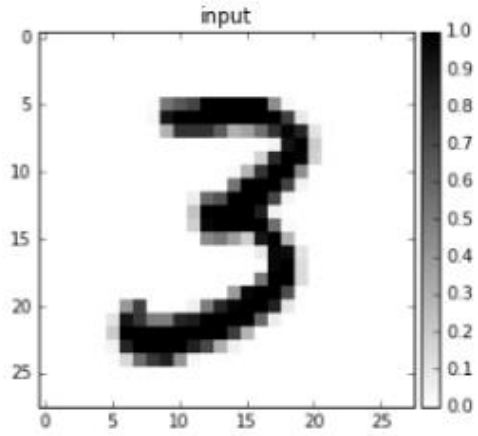

Figure 4: *Image taken from MNIST dataset. Every row and column represents one pixel size.*

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

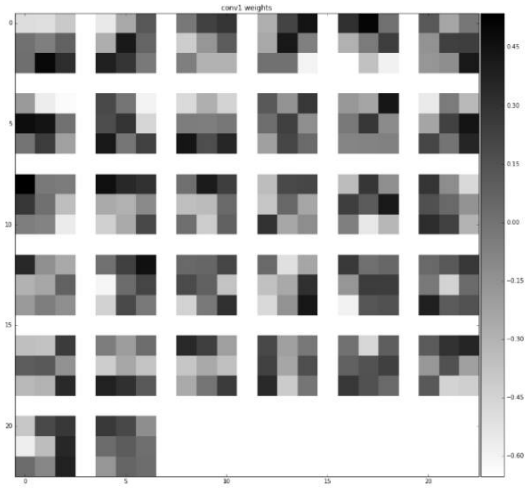

Figure 5: *All 32 kernels learned from the first convolutional layer.*

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

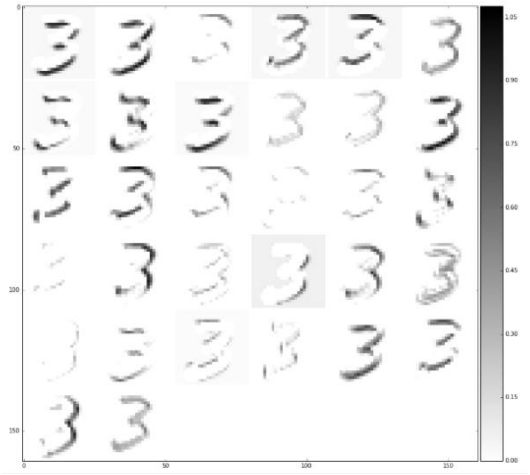

Figure 6: *All 32 Kernels applied to an example of the datase.*

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

—— Nitish's text here. ——

2 Approach

Datasets

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

- Pack of Coffee
- Paper coffee-cup
- Juice-box
- Big ketchup bottle
- Small ketchup bottle
- Mr Muscle cleaner
- Pepper
- Pringles can
- Water bottle

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 pixel.

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

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.

2.1 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

—— Nitish's text here. ——

3 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.

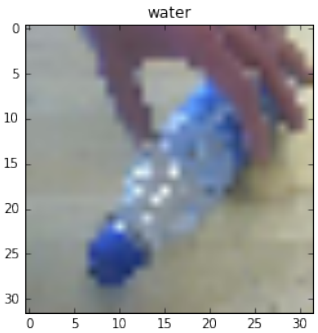

Figure 7: *Cluttered image of the class "Water bottle" with hand of experimenter. The network was able to classify the image correctly.*

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

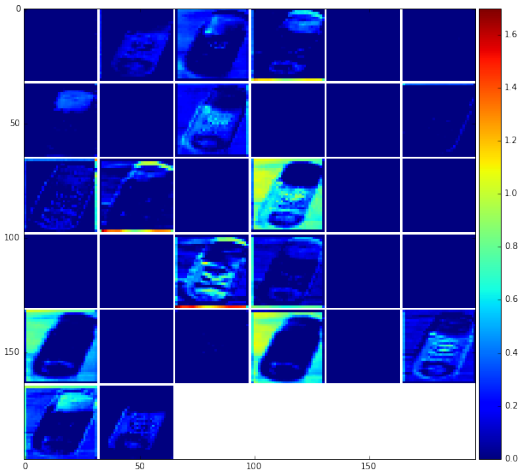

Figure 8: *Output after the first convolutional layer of our network, when presented with an input image.*

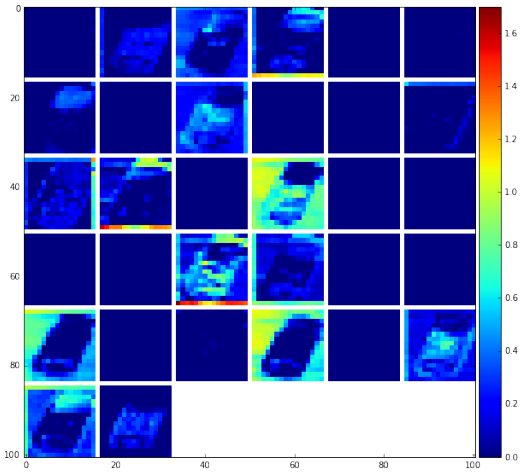

Figure 9: *Output after the second convolutional layer of our network, when presented with an input image.*

Sparse Coding

—— Nitish's text here. ——

4 Conclusion

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%.

Sources

Appendix B: Results