Designing Convolutional Neural Networks Using a Genetic Approach for Ball Detection

Georg Christian Felbinger(✉), Patrick Göttsch, Pascal Loth, Lasse Peters, and Felix Wege

RobotING@TUHH e.V., Hamburg University of Technology, Hamburg, Germany
hulks@tuhh.de, https://hulks.de

Abstract. At RoboCup 2017, the HULKs reached the Standard Platform League's quarter finals and won the mixed team competition together with our fellow team B-Human. This paper describes the design of a convolutional neural network used for the detection of the black and white ball - one of the key contributions that led to the team's success. We present a genetic design approach that optimizes network hyperparameters for a cost effective inference on the NAO, with limited amount of training data. Experimental results demonstrate that the genetic algorithm is able to optimize the hyperparameters of convolutional neural networks. We show that the resulting network is able to run in real-time on the robot with a very precise classification in generalization test.

1 Introduction

In 2016, a black and white patched ball was introduced into the Standard Platform League (SPL). While in previous years color based approaches [2,5] were sufficient to achieve a acceptable detection and classification performance, the new ball requires more sophisticated techniques. Requirements for a detection algorithm comprise a robust detection and classification in dynamically changing environments, as well as a cost-effective real-time computation on the NAO.

Approaches based on convolutional neural networks (CNN) for object detection led to promising results in RoboCup SPL [7,8]. However, hyperparameters for the structural setup of such networks need to be chosen carefully. Genetic approaches as described in [4] and [9] can be used to determine an optimized network topology. Stanley and Miikkulainen described the evolution of fully connected network topologies [10]. The idea can easily be applied to other model components, e.g. convolutional layers. A similar genetic approach was used by Sun, Xue, and Zhang to automatically discover good architectures of CNNs [11].

This paper presents a genetic framework to design CNNs for real-time applications on computationally weak hardware by simultaneously optimizing the classification performance and inference complexity. Our approach considers a bounded capability to collect large amounts of training data and allows the user to prioritize true negative rate and true positive rate suitable for a specific task.

D. Holz et al. (Eds.): RoboCup 2018, LNAI 11374, pp. 150–161, 2019.
https://doi.org/10.1007/978-3-030-27544-0_12

The detection of a black and white ball on the NAO robot is used to demonstrate the performance of the framework.

Section 2 outlines the general idea of a genetic algorithm as well as the components used to assemble a CNN. Section 3 describes the resulting search space and the fitness function used for the genetic optimization. Section 4 presents the process of data acquisition. Herein the techniques used to generate and setup training data are described. Finally, in Sect. 5 the conducted experiments are presented and the evaluation results are discussed.

2 Prerequisites

2.1 Genetic Algorithm

The method used in this work follows a genetic algorithm pattern [4]. The basic elements are chromosomes or individuals $c \in S$, a possible solution in given k-dimensional search space S. The algorithm works in an iterative manner with a fixed number of generations N. A set of n chromosomes used during iteration j is called population $P_j = \{c_1, ..., c_n\} \subset S$. The initial population P_0 is generated randomly. In each generation $j \in [1, N]$ the population of the previous iteration is evaluated using a fitness function $f(c) : \mathbb{C}^k \mapsto \mathbb{R}$. Given the individuals fitnesses of the previous population a set S_j is selected from P_{j-1} as parents. In the next step a mutation function will be applied to every element in S_j. Finally, mutated parent elements are recombined yielding the next generation P_j.

2.1.1 Selection

The selection is done using the following steps. According to a given clipping parameter $c \in [0, 1]$ individuals in the lower cth percentile is dropped. The minimal fitness within the population is given by $\min_{k \in [1,n]}(f(c_k))$. Given the other m individuals the probability of survival is calculated by Eq. (1).

$$p(c_i) = \frac{f(c_i) - \text{minscore}}{\sum_{j=1}^{m} (f(c_j) - \text{minscore})} \tag{1}$$

Hence, the individual with the lowest fitness value is assigned to the survival probability zero. According to this distribution, n elements are sampled for mutation and reproduction.

2.1.2 Mutation

For every value within chromosome c a new value will be sampled based on a given mutation probability p_m. If a gene is to be replaced a new random value is chosen.

2.1.3 Reproduction

In the reproduction phase the selected and mutated chromosomes are pairwise randomly sampled. Each pair yields two new children. For every value within the chromosome of a child, the corresponding parent value is chosen randomly.

2.2 Convolutional Neural Networks

In our recent work [7] convolutional neural networks showed promising results in the field of object detection. The following briefly describes the basic components used for our CNN structure.

2.2.1 Convolutional Layer

In this work multiple two-dimensional convolutions are used, i.e. an input image with q channels is mapped to an output image with k channels. Equation (2) shows the computation of a convolutional layer.

$$\begin{gathered} y_{i,j,k} = \sum_{di,dj,q} x_{i+di,j+dj,q} \cdot m_{di,dj,q,k} \\ x \in \mathbb{R}^{i \times j \times q}, m \in \mathbb{R}^{di \times dj \times q \times k} \end{gathered} \tag{2}$$

2.2.2 Pooling Layer

Pooling layers reduce every dimension of each image channel by applying a function to neighboring pixels using a 2×2 mask. In this paper $\max(a, b, c, d)$ (maximum value of arguments) and $\text{avg}(a, b, c, d)$ (arithmetic mean of arguments) are used.

2.2.3 Normalization Layer

Batch normalization layers are used to increase the learning rates and to reduce the sensitivity to the initialization of the weights [6]. During training normalization is calculated batch-wise. For input vectors $[1, m] \in \mathbb{N}$ it is calculated by Eq. (3).

$$\text{BN}_{\gamma,\beta}(x_i) = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B + \epsilon}} + \beta \tag{3}$$

The scale γ and offset β are trainable parameters which get optimized due to the training problem. The batch mean is element-wise computed by $\mu_B = \frac{1}{m} \sum_{j=1}^{m} x_j$. The batch variance is also element-wise computed by $\sigma_B = \frac{1}{m} \sum_{j=1}^{m} (x_j - \mu_B)$.

For the inference mean and variance are approximated by a moving average approach during training [6, pp. 4]. Mean μ_n and variance σ_n after nth batch can be computed recursively using a moving average.

3 Genetic Design of Convolutional Neural Networks

To find an optimal topology of the CNN the genetic algorithm mentioned in Sect. 2.1 is used. Each topology set corresponds to a single individual within the search space. CNN structure and search space are specified in this chapter.

3.1 Network Structure

The uniform structure used for all evaluated individuals is given in Fig. 1. The input is a YCbCr candidate image of arbitrary quadratic size. It is resized to a fixed quadratic size using nearest neighbor interpolation. Then, multiple convolutional layers are applied. The next step is a batch normalization layer. Finally, multiple fully connected layers are applied. The output is a vector representing the class scores.

Each individual specifies the remaining hyperparameters within this structure. These are the input size, number of convolutional and fully connected layers as well as their internal configuration. Each convolutional layer is parameterized with a mask size, pooling type and activation function. Likewise, the parameters of a fully connected layer consists of the size and activation function.

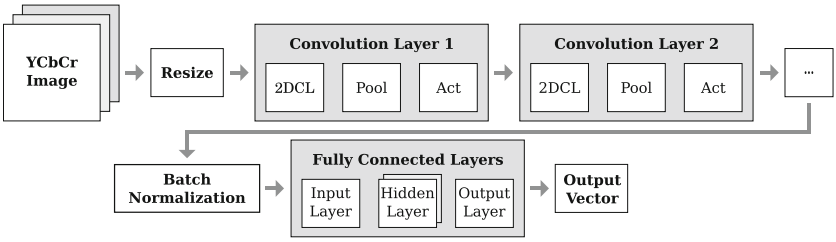


Fig. 1. General structure of a CNN. Each convolutional layer consists of a two dimensional convolution mask (2DCL) followed by a pooling layer (Pool) and an activation function (Act). The convolved and normalized image is fed into multiple fully connected layers yielding the final output vector.

3.2 Search Space

The search space for the genetic algorithm consists of parameters described in Sect. 3.1 with the value ranges described in the following. The size of the quadratic input image is sampled within the range $[8, 16] \in \mathbb{N}$. The amount of convolutional layers is limited to two. For each convolutional layer the number of kernels is chosen from $[1, 5] \in \mathbb{N}$. The kernel size is equal within each convolutional layer and is either two or three in both dimensions. Either no pooling, max-pooling or avg-pooling is used in the pooling layer. The activation function for the CNN and the fully connected layer is either tanh or rectified linear unit (ReLU). There are four fully connected layers at maximum each with a number of neurons within $[2, 20] \in \mathbb{N}$.

3.3 Fitness Function

We optimize classification performance and inference complexity at the same time. In the fitness function classification performance is represented by the true negative and true positive rate. Inference complexity is approximated asymptotically.

3.3.1 Classification Performance

For each network a k-fold cross validation was performed which yielded k values for true negative rate TNR_{nk} and true positive rate TPR_{nk}. In order to approximate a lower bound of these performance metrics the difference of mean and variance were used in the fitness function. The TPR_n and TNR_n for a network n was computed by:

$$TPR_n = \text{Avg}(TPR_{n1}, ..., TPR_{nk}) - \text{Var}(TPR_{n1}, ..., TPR_{nk}) \tag{4}$$

$$TNR_n = \text{Avg}(TNR_{n1}, ..., TNR_{nk}) - \text{Var}(TNR_{n1}, ..., TNR_{nk}) \tag{5}$$

where Avg is the arithmetic mean and Var is the variance.

3.3.2 Inference Complexity

The complexity of a network was asymptotically approximated and linearly scaled. The complexity cc of a convolutional layer i is approximated by Eq. (6).

$$cc_i = \frac{I_x \cdot I_y \cdot I_c \cdot m_x \cdot m_y \cdot m_c}{\hat{I}_x \cdot \hat{I}_y \cdot I_c \cdot \hat{m}_x \cdot \hat{m}_y \cdot \hat{m}_c} = \frac{I_x \cdot I_y \cdot m_x \cdot m_y \cdot m_c}{\hat{I}_x \cdot \hat{I}_y \cdot \hat{m}_x \cdot \hat{m}_y \cdot \hat{m}_c} \tag{6}$$

Symbols I_x, I_y, I_c correspond to layer input size and depth, m_x, m_y, m_c to amount and size of the convolution masks in this layer. While $\hat{I}_x, \hat{I}_y, \hat{m}_x, \hat{m}_y, \hat{m}_c$ represent maximum values as defined by the Sect. 3.2.

The complexity cf of the fully connected part is approximated by Eq. (7).

$$cf = \frac{\sum_{i=1}^{k} s_i \cdot s_{i-1}}{\sum_{i=1}^{k} \hat{s}_i \cdot \hat{s}_{i-1}} \tag{7}$$

The number of hidden layers is denoted by k and the size of layer i by s_i. The input vector size is s_0.

Hence, the final complexity of a network topology with j convolutional layers is

$$c_n = 1 - \frac{\sum_{i=1}^{j} cc_i + cf}{j+1}. \tag{8}$$

3.3.3 Resulting Fitness Function

Given the approximation of classification performance and inference complexity the resulting fitness function is chosen as follows:

$$f_n = 0.7 \cdot TNR_n^2 + 0.25 \cdot TPR_n^2 + 0.05 \cdot c_n. \tag{9}$$

In our case for the desired behavior of the ball detection, the TNR is much more important than the TPR. Hence, this component is assigned the largest weight. The search space is already limited to topologies which are feasible for inference on the target system. Therefore, the inference complexity is weighted with a very low weight in the fitness function. If networks have similar classification

performance the smaller network is preferred. These weights were chosen based on empirical considerations.

For networks with a good classification performance it is disproportionally difficult to further increase the *TNR* and *TPR*. Thus, the *TNR* and *TPR* are squared in the fitness function.

4 Data Acquisition

4.1 Data Setup

The data used for training and evaluation of the classifier were collected during various events (RoboCup 2017, Iran Open 2017, German Open 2017, weekly test games). It consists of 16880 positive examples (candidate images containing a ball) and 23876 negative examples (candidate images not containing a ball). During training negative examples are subsampled randomly to ensure that in every cross-validation set the same amount of positive and negative examples are present.

4.2 Candidate Generation

The training data is collected directly on the robot to ensure equally sampled data during training and inference. This allows an iterative process consisting of the following steps.

1. Training a network with the collected data.
2. Running the newly trained model in a test environment while collecting data not yet seen by the network.
3. Labeling newly collected data and evaluating classification performance based on the test set.

4.2.1 Generating Seeds

In the first step of the candidate generation the algorithm determines seeds for possible candidates. The image is segmented using vertical scan lines on every second column of the image. The two dimensional gradient is computed along the scan lines. Whenever this gradient exceeds a preconfigured threshold a new segment along this scan line is created. The median of five pixels equally distributed over the segment determines the color of the segment. A seed is the central pixel of a segment which passes a series of checks.

1. The luminance of the corresponding region must be lower than 100 which naturally corresponds to the black patches of the ball.
2. The corresponding ball radius in pixels r_p at the seed's position is determined by projecting the assumed ball onto the image. If the ratio $r_s = \frac{l_s}{r_p}$ of the segment's length l_s to the pixel radius is not within the range $[0.1, 0.7]$ the seed is dropped.

3. Neighboring areas of the black patch are checked. Therefore the luminance in eight directions around the seed with a distance of $\frac{r_p}{2.5}$ are sampled. All of those sampled values must be slightly higher than the luminance of the seed. Also, five of those values must have a significant difference in luminance.

If all conditions match, the seed is used for the candidate generation.

4.2.2 Merging Seeds To Candidates

Seeds are merged into a single candidate if they are close to each other. First, an empty set of candidates is initialized. For every seed in the image it is checked if there is a nearby candidate with a maximum distance of a ball diameter in pixels. If a candidate is found the current seed will be merged into that candidate by taking the mean of the position and radius. Otherwise a new one with position and radius of the current seed is added. Afterwards candidates are filtered such that only candidates based on at least two seeds remain.

4.2.3 Reprojection of Found Balls

Our existing software framework features a filter to estimate the ball state. The physical model of the ball is used to predict the ball position in the current image which yields another candidate.

(a) (b) (c)

Fig. 2. Visualization of ball candidates [3, pp. 17–18]. (a) Seed is corresponding to the center of the black patches on the ball. (b) Merged seeds and projection of the corresponding ball radius. (c) Reprojected ball from result of the ball filter (green circle bounded black rectangle). (Color figure online)

5 Experiments and Evaluation

The computational power of the NAO is severely limited. Therefore, designing the CNN using the genetic algorithm is done offline on a more powerful machine. The resulting network is transferred to the robot and is evaluated in a final generalization test. The classification performance is measured based on labeled data collected by the candidate generation.

5.1 Setup

In every experiment 15 generations with 50 networks in each generation were evaluated. The worst 10% in each generation were excluded from reproduction. The mutation probability was set to $\frac{1}{16}$ according to the maximum number of degrees of freedom of the given search space.

The algorithm should be able to design a CNN as a solution to the ball detection problem considering a limited amount of training data. To show that this can be achieved three experiments were conducted, sampling 25%, 50% and 100% of the available training data.

5.2 Results

For the experiment with 25% of the data the best networks in the last generation reached a *TNR* of 0.91 to 0.95 with a *TPR* of about 50% to 75%. The resulting network has a very small sample size of 8×8, one convolutional layer of four masks and only three hidden layers in the fully connected part.

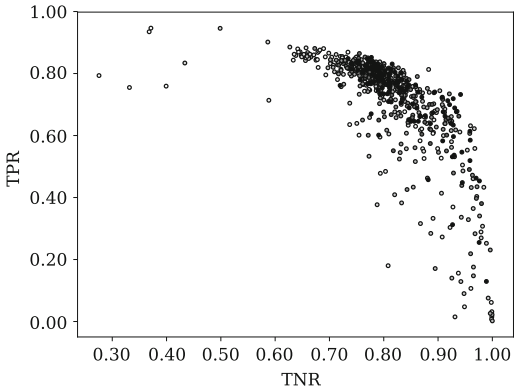
With 50% of data the best networks in the last generation reached a *TNR* of about 0.93 with a *TPR* above 0.85. Networks with one large convolutional layer and mainly three hidden layers in the fully connected part dominated this experiment.

With the full amount of data the best networks in the last generation reached a *TNR* of about 0.95 with a *TPR* above 0.90. While the sample size and convolutional layers remained similar to those in the second experiment, the fully connected part converged to four hidden layers instead of three.

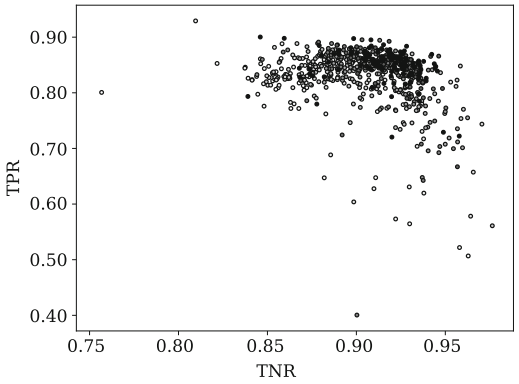
5.3 Evaluation

In early generations of the first experiment networks with a very high *TNR* also had a very low *TPR* as Fig. 3a illustrates. These individuals were eliminated due to the weights of the fitness function. Networks of the last generation were highly biased to reject input which yields a high *TNR* while having a poor *TPR*. Classification performance in the second experiment was significantly higher. Therefore, not only individuals with a bad *TNR* were eliminated but also the *TPR* converged over time. Figure 3b shows that there were no outliers with a *TPR* below 0.70 in later generations. Figure 3c shows that later generations of the final experiment also formed a very dense cluster. Hence, the algorithm could not really find a much better solution throughout generations but was able to select better networks and remove outliers.

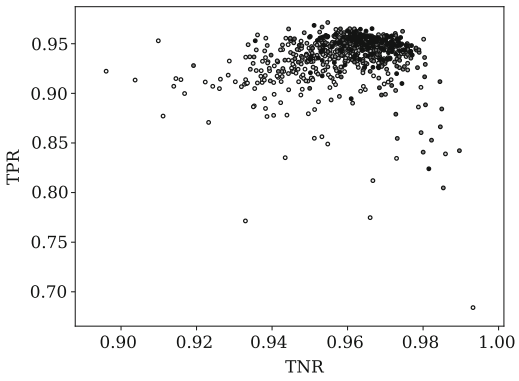
The network having the highest score in the last generation is considered to be the resulting network. Table 1 shows a summary of those networks in each experiment. Networks became more complex while increasing the amount of data resulting in better classification performance.



(a) 25% training data



(b) 50% training data



(c) 100% training data

Fig. 3. Evolution of classification performance. Note that scaling differs between experiments as the overall results got better. Results of the first generation are plotted with white filled circles. Results of the following generations are plotted in increasingly darker shades of gray.

Table 1. Overview of the resulting networks of experiments. Column *Exp* lists the number of the experiment. The second column *Data* corresponds to the amount of data used. Columns *TNR*, *TPR* and *Comp* show the components of the fitness function. Note that the *TNR* and *TPR* parts are the lower bound approximations described in Eqs. (4) and (5). Column *Score* corresponds to the resulting score.

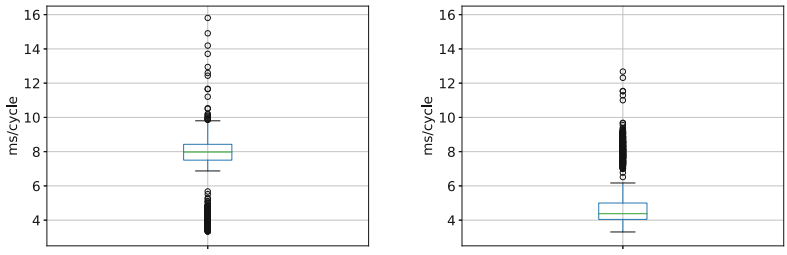
Exp	Data	*TNR*	*TPR*	Comp	Score
1	25%	0.921	0.700	0.732	0.856
2	50%	0.932	0.853	0.663	0.899
3	100%	0.972	0.958	0.638	0.922

5.4 Generalization Test

The best network of the last generation that was trained with all of the training data is subjected to a final generalization test. This final network is evaluated with data collected in another environment which can be considered to be a proper generalization test because no data from these testing conditions was used during training. The classifier predicted 4989 of 5687 positives and 12680 of 12730 negatives correctly resulting in a $TNR = 0.99$ and a $TPR = 0.87$.

5.5 Runtime Analysis on the NAO Robot

The whole ball detection including the resulting network running on the NAO was evaluated. For this test we fixed the number of generated candidates per image to the average amount five to get stable measurement results. Figure 4 shows the result of those measurements. With an average runtime of about 8 ms on the top and 4 ms on the bottom camera we reached our real-time criteria which is 30 ms for a vision cycle.



(a) Top Camera (b) Bottom Camera

Fig. 4. Runtime of the ball detection including the resulting network on the NAO. The green line indicates the mean of the runtime. The interquartile range is shown by the blue box. The upper black bar illustrates the 0.75-quantile, respectively the lower black bar the 0.25-quantile. The circles correspond to outliers. (Color figure online)

6 Conclusion

The goal of this paper was to optimize the topology of neural networks with respect to classification performance and inference complexity simultaneously. We presented a genetic framework that was successfully applied to the problem of black and white ball detection using little computational power. Our experiments showed that a genetic approach is able to identify a small yet efficient network suitable for a specific classification task. The presented optimization strategy obtains suitable hyperparameters even with limited amount of training data. However, the optimization of the architecture needs a lot of computation time as the algorithm has to train and evaluate plenty of CNNs, in our case 2250 networks. Thus, applying the presented optimization strategy to classification problems that require significantly more complex networks may be infeasible.

In order to enhance convergence speed future work should focus on evaluating different variants of genetic algorithms such as elitism [1]. Additionally, we would like to apply the approach to multiclass problems using a modified fitness function.

References

1. Baluja, S., Caruana, R.: Removing the genetics from the standard genetic algorithm. Technical report CMU-CS-95-141. Pittsburgh, PA: Carnegie Mellon University, May 1995
2. Budden, D., Fenn, S., Walker, J., Mendes, A.: A novel approach to ball detection for humanoid robot soccer. In: Thielscher, M., Zhang, D. (eds.) AI 2012. LNCS (LNAI), vol. 7691, pp. 827–838. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35101-3_70
3. Felbinger, G.C.: A genetic approach to design convolutional neural networks for the purpose of a ball detection on the NAO robotic system. Project Work, October 2017. https://www.hulks.de/_files/PA_Georg-Felbinger.pdf
4. Harbich, S.: Einführung genetischer Algorithmen mit Anwendungsbeispiel. Universität Magdeburg, December 2007
5. Härtl, A., Visser, U., Röfer, T.: Robust and efficient object recognition for a humanoid soccer robot. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013. LNCS (LNAI), vol. 8371, pp. 396–407. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44468-9_35
6. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456, March 2015
7. Kahlefendt, C.: A Comparison and Evaluation of Neural Network-based Classification Approaches for the Purpose of a Robot Detection on the Nao Robotic System. Project Work. April 2017. http://www.hulks.de/_files/PA_Chris-Kahlefendt.pdf
8. Menashe, J., et al.: Fast and precise black and white ball detection for RoboCup soccer. In: Akiyama, H., Obst, O., Sammut, C., Tonidandel, F. (eds.) RoboCup 2017. LNCS (LNAI), vol. 11175, pp. 45–58. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00308-1_4
9. Mitchel, M.: An Introduction to Genetic Algorithms A Bradford Book. The MIT Press, Cambridge (1999). ISBN 0-262-63185-7

10. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002). http://nn.cs.utexas.edu/?stanley:ec02
11. Sun, Y., Xue, B., Zhang, M.: Evolving deep convolutional neural networks for image classification. In: CoRR abs/1710.10741 (2017). arXiv: 1710.10741. url: http://arxiv.org/abs/1710.10741