A Benchmark Data Set and Evaluation of Deep Learning Architectures for Ball Detection in the RoboCup SPL

Simon O'Keeffe(✉) and Rudi Villing

Department of Electronic Engineering, Maynooth University, Maynooth, Ireland
simon.okeeffe.2010@mumail.ie, rudi.villing@nuim.ie

Abstract. This paper presents a benchmark data set for evaluating ball detection algorithms in the RoboCup Soccer Standard Platform League. We created a labelled data set of images with and without ball derived from vision log files recorded by multiple NAO robots in various lighting conditions. The data set contains 5209 labelled ball image regions and 10924 non-ball regions. Non-ball image regions all contain features that had been classified as a potential ball candidate by an existing ball detector. The data set was used to train and evaluate 252 different Deep Convolutional Neural Network (CNN) architectures for ball detection. In order to control computational requirements, this evaluation focused on networks with 2–5 layers that could feasibly run in the vision and cognition cycle of a NAO robot using two cameras at full frame rate (2×30 Hz). The results show that the classification performance of the networks is quite insensitive to the details of the network design including input image size, number of layers and number of outputs at each layer. In an effort to reduce the computational requirements of CNNs we evaluated XNOR-Net architectures which quantize the weights and activations of a neural network to binary values. We examined XNOR-Nets corresponding to the real-valued CNNs we had already tested in order to quantify the effect on classification metrics. The results indicate that ball classification performance degrades by 12% on average when changing from real-valued CNN to corresponding XNOR-Net.

Keywords: Convolution neural network · Deep learning · Ball detection
XNOR-Net

1 Introduction

In the RoboCup Soccer Standard Platform League (SPL), ball detection has frequently relied on hand-crafted heuristic approaches that rely on colour with some shape constraints. The Softbank Robotics NAO robots that are used in the SPL have limited computational resources and this is a principal reason why heuristic vision processing approaches have been used to date.

In 2016, rule changes led to a change of the standard ball from an orange street hockey ball to a 10 cm foam ball with the 32 panel black and white pattern typical of a traditional soccer ball. The principal challenge of this new ball is that it does not have

H. Akiyama et al. (Eds.): RoboCup 2017, LNAI 11175, pp. 398–409, 2018.
https://doi.org/10.1007/978-3-030-00308-1_33

a unique colour on the field of play and for that reason colour alone cannot be used to detect it. Furthermore the ball can be difficult to distinguish from parts of other robots or when partly occluding field objects such as lines, goal posts, and robots. In general, heuristic based vision processing approaches need to deal with many different conditions identified through domain expertise and trial and error testing. Our own team's heuristic based ball detection was found to regularly require the addition of extra conditions, tended to produce many false positives, and suffered from a limited ball detection range (when compared to the previous orange ball detector). In 2017, the SPL rules have been changed to permit play in natural and variable light conditions and this further challenges the heuristic based approach.

Given the difficulties associated with heuristic based ball detection, a more sophisticated approach is required that is more robust and less dependent on colour and uniform lighting. Deep Convolutional Neural Networks (CNNs) are recognised as the state of the art for object recognition [1] and we expect that such state of the art approaches should outperform the heuristic based algorithms that we and other teams have used to date. Given a suitable dataset, a Deep Neural Network (of which CNNs are but one possibility) can learn features of the ball that are robust to lighting changes, occulusion and distractor conditions, and movement by the robot or the ball. Therefore the first contribution of this paper is to publish an extensive labelled data set of ball images that may be used for training and subsequent test of Deep Neural Networks and other machine learning techniques.

One of the key factors that has enabled the advancement of Deep Neural Networks (DNNs) has been the use of Graphical Processing Units (GPUs), with speed-ups on the order of 10 to 30-fold in comparison to CPU only processing [2]. However, DNN approaches are much less typical with low power embedded systems that do not have a GPU due to the computational requirements. Therefore our second contribution is an evaluation of multiple Deep CNN architectures that may be feasibly implemented on the NAO robot and similar low power embedded systems. This evaluation focuses on the classification metrics of the networks and the inference time per image.

There are a number of approaches that may be used to reduce the computational requirements of Deep CNNs and these are described under related work. Our third contribution is a specific evaluation of XNOR-Net [3], a particularly promising approach for reducing computation and speeding up inference that quantizes both the network weights and activations to binary values.

The remainder of this paper is organized as follows: Sect. 2 presents some related work and motivates the evaluation of XNOR-Net. Section 3 describes the approach taken to the dataset. Our network design approach is presented in Sect. 4. Section 5 contains our results and discussion. Finally Sect. 6 presents our conclusion and future work.

2 Related Work

Most of the computation performed during training and application of DNNs results from the multiplication of real-valued weights by real-valued activation values. Several

approaches have been proposed to improve the computational efficiency of the network at both training and inference time.

Shallow networks have been used to estimate deep networks. First Reference 4 showed that a large enough hidden layer of sigmoid units can approximate any decision boundary. However for vision and speech processing, shallow networks generally can't compete with deep models [5].

Pre-trained deep networks can be compressed by pruning redundant weights in a trained network to reduce the size of the network at inference time. Early methods for pruning a network included weight decay [6], Optimal Brain Damage [7], and Optimal Brain Surgeon [8]. More recent approached to pruning included Deep Compression [9], which reduces the storage and energy required to run inference on large networks so they can be deployed on mobile devices. Deep compression does this by removing redundant connections and quantizing weights so that multiple connections share the same weight, and then use Huffman coding to compress the weights.

Designing compact blocks that use fewer parameters at each layer of a deep network can help to save memory and computational costs. Replacing the fully connected layer with global average pooling was examined in the Network in Network architecture [10], GoogLenet [11], and Residual-Net [12], which have achieved state-of-the-art results on several benchmarks. The bottleneck structure (which uses 1×1 convolutions) in Residual-Net has been proposed to reduce the number of parameters and improve speed.

High precision parameters are not very important in achieving high performance in deep networks [13] and many approaches have proposed quantizing parameters to reduce the size of the network. The authors in [13] proposed to quantize the weights of fully connected layers in a deep network by vector quantization techniques. They showed that simply thresholding the weight values at zero decreases the top-1 accuracy on ILSVRC2012 by less than 10%. Other work examined using ternary weights with the weights restricted to $+1/0/-1$ [14] and networks that used ternary weights and 3-bits activations [15].

Several researchers have gone a step beyond the above quantization approaches to network binarization. Initially, the performance of highly quantized or binarized networks were believed to be very poor due to the destructive property of binary quantization [16]. However, this was later shown not to be the case. BinaryConnect [17] trains a DNN with binary weights during forward and backward propagations, but retains the precision of the stored weights in which gradients are accumulated. The authors found that BinaryConnect acted as a regularizer and obtained near state-of-the-art results on MNIST, CIFAR-10 and SVHN. BinaryNet [18] was proposed as an extension of BinaryConnect. In BinaryNet both weights and activations are binarized, constrained to either $+1$ or -1. If all operands of the convolutions are binary, then the convolutions can be estimated by XNOR and bit counting operations. This quantization can also applies to the fully connected layers. Again, this approach achieved nearly state-of-the-art results on the MNIST, CIFAR-10 and SVHN datasets. XNOR-Net is another method that binarizes the weights and activations in a network [3]. XNOR-Net differs from BinaryNet in the binarization and the network structure. XNOR-Net was found to outperform BinaryNet on large datasets (e.g. ImageNet).

3 General Approach and Data Set

As a low power embedded processor, the Intel Atom processor of the NAO robot does not have the compute power needed to execute standard DNN techniques applied to the entire full resolution camera image at camera frame rate (usually 30 frames per second). Therefore we assume a general vision pipeline in which a ball candidate region proposal algorithm first scans the image for ball candidates using some unspecified but computationally efficient technique (that may be heuristic based or not). We then assume that one or a subset of proposed ball candidate regions are tested using a DNN to determine which candidate (if any) best represents a ball.

To ensure a data set that is suitable for training and testing the DNN component of this pipeline while maximizing flexibility for future developments the requirements for a benchmark data set are as follows. The data should provide full images with labelled region coordinates that specify ball and non-ball candidate regions (patches). In addition, the data set should contain a wide variety of candidates (with and without ball) that span the space of conditions under which a ball must be detected.

Our final data set comprises 6564 unique 640×480 images and it is available for download at https://www.roboeireann.ie/research/SPLBallDataset.zip. From this set of images, 5209 ball patches (candidate regions which contain a ball) and 10924 non-ball patches (candidate regions which do not contain a ball) are extracted.

The data set is divided into training, validation and test sets such that 70% is used for training, 15% for validation, and the remaining 15% for test. The ball patch data includes candidates that were close (less than 3 m away) and far away from the robot (3–8 m away). It includes candidates that were in free space on the field and candidates that were near, partially occluding or, if appropriate, partially occluded by various distractors (penalty spots, field lines and intersections, and robot parts). The data set includes ball candidates where the robot and ball were both static and where the robot, ball, or both were moving. Finally, the data includes ball candidates on various pitch surfaces, some of which were under artificial light and others under natural light. The non-ball patches include field lines and intersections, robot parts, goal parts, shoes, feet, and hands. A selection of ball and non-ball patches can be seen in Fig. 1.

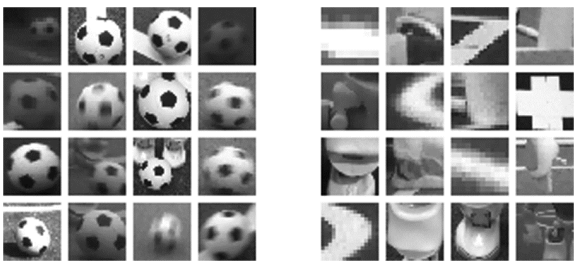

Fig. 1. Example of ball and non-ball patches extracted from full images in the dataset

The dataset was prepared from vision log files collected at RoboCup and in our laboratory. In all, 31 log files were used. All log files were captured from NAO V4 and

NAO V5 robots. The logs were captured from 9 different robots and the logs include a mix of top camera and bottom camera. Bottom camera images were captured natively at 640×480 pixels resolution whereas top camera images were captured at 1280×960 pixels resolution and then decimated to 640×480. Images used YUV format.

The ball pixels were manually labelled in images extracted from five of the log files. The remaining log files were first processed through our existing heuristic based ball detector. Using this approach, each image was always labelled with a non-ball patch location, that is, the location of a candidate considered but ultimately rejected as a ball. In addition the same image was labelled with a ball patch location if our existing ball detector accepted one of the ball candidates it had processed. The patches associated with each image were inspected afterwards and manually re-classified as ball or non-ball as needed. This ensured that the data set was not negatively affected by weaknesses (primarily false positives) in our existing ball detector.

Ball patches in the source images varied from 12×12 pixels (the minimum size we permitted) up to 158×158 pixels. The luminance (Y) channel of each ball and non-ball patch was extracted and resized to a standard size for later training and test of DNNs. We used sizes of 12×12, 20×20, and 32×32 for reasons explained in Sect. 4.1. Resizing was performed using the computationally efficient nearest-neighbor algorithm since that is likely to be used in the vision pipeline on the NAO robot.

Simply extracting all ball and non-ball patches from consecutive image frames in each log file can result in excessively correlated patches in the case that neither the robot nor the ball is moving. To eliminate such correlation we included a ball or non-ball patch from a given log file in the data set only if the mean absolute difference between its pixels and those of the previously included patch exceeded a threshold of 10 luminance points per pixel. This threshold was determined empirically by examining the mean absolute difference of patches from consecutive frames throughout the data set. This process eliminated 43.2% of the ball and non-ball patches due to correlation.

Many ball-patches that had already been included in the data set were based on a bounding box that cropped the ball tightly and excluded extraneous information as a consequence. However, it may not always be possible for the ball candidate proposal algorithm to achieve this. Therefore, we augmented the data set by creating variants of ball and non-ball patches that were more loosely cropped (and where the ball was smaller in the patch as a consequence). To do this, we went back to the patches in the images prior to resizing to standard patch sizes. The original bounding box around each patch was first scaled by value between 1.1 and 1.5 chosen at random. The bounding box was then translated by a random value between -0.33 to 0.33 times its new edge length in the horizontal direction and similarly translated by a random value in the vertical direction. If the original bounding box for a patch was at the border of the image it was excluded from augmentation. The data set after augmentation contains 89% more patches, consisting of 5209 ball and 10924 non-ball patches.

4 Deep CNN Evaluation Design

The evaluation was designed to evaluate the performance of a large number of Deep CNN networks that could be expected to execute quickly enough on the NAO robot. If images from both cameras in a NAO robot are processed at their maximum rate then there is a time budget of approximately 16.7 ms available to process each image and perform any necessary perception and cognition activities. Therefore, a Deep CNN that will be used in the vision pipeline can consume only a portion of that time budget. The shorter the inference time, the more likely it is that the network can be applied to multiple candidate patches rather than just one, so this makes it attractive to identify network architectures which can make inference as quickly as possible while maintaining accuracy.

We used the Caffe framework to develop and test our network architecture [19]. Caffe is a deep learning framework that facilitates rapid testing of different network architectures because the network architectures are specified by configuration files. In addition, Caffe can switch between using CPU and GPU depending on the host platform which allows for fast training on a machine with a GPU with subsequent deployment to another system having only a CPU, such as the NAO robot, for inference testing.

4.1 Network Design

There are a number of parameters that can be used to specify a network. One of the most fundamental of these is the size of the input image patch. Our existing heuristic based ball detector performs worst with balls that are more than 3 m from the robot. With a 640×480 pixel image the ball diameter at 3 m from the robot is approximately 20 pixels. This decreases to 12 pixels between 5 m and 6 m and to 7–8 pixels at 8 m from the robot. This suggested that patch sizes between 8 and 20 pixels square could be appropriate. We are aware of two other RoboCup SPL teams that have considered Deep CNNs for ball detection. Nao-Team HTWK reported a network that uses 20×20 pixel input patches [20] while UT Austin Villa's code release 2016 [21] used somewhat larger 32×32 pixel patches.

These input patch sizes are similar in size to those of the well-studied LeNet architecture [22] which used 32×32 pixel patches. LeNet was one of the first convolutional networks and operated on the MNIST dataset of hand-written digits. The authors presented many different variants of LeNet with the most successful consisting of 2 convolutional layers followed by 2 fully connected layers, with 20 outputs in the first layer, 50 outputs in the second layer, 500 in the third layer and 10 outputs for the final layer for each of the 10 digits. It used 5×5 convolution kernels.

More recent work on CNNs such as VGGNet [23] and GoogLeNet [11] has introduced smaller kernel sizes. Smaller kernels have the advantage of capturing more detail yet they can be stacked up to capture wider receptive fields (e.g. two 3×3 kernels in different layers together have a receptive field of 5×5). For this reason we evaluated designs with various kernel sizes. Network in Network [10] introduced the idea of 1×1 convolution kernels. Such a kernel can be used to reduce the number of parameters in the network and may be used as a convolutional layer in the network, where it is known

as a bottleneck, or to replace the fully connected layers that are often placed at the end of a CNN (coupled with average pooling). Our evaluation included network designs that replaced fully connected layers with 1×1 convolutional layers.

Batch Normalization [24] layers normalize the input batch by its mean and variance. This technique was introduced to overcome internal covariate shift where the distribution of each layer's inputs changes during training as the parameters of the previous layers change. The authors found that Batch Normalization speeds up training time, achieving the same accuracy with 14 times fewer training steps as well being more robust to high learning rates and parameter initialization. Rectified Linear Units (ReLUs) [25] are now commonplace in many state-of-the-art deep neural networks. ReLUs are used over the sigmoid function as they have a reduced likelihood of vanishing gradient. The constant gradient of ReLUs results in faster learning.

In our Deep CNN designs, a convolutional block consists of convolution, Batch Normalisation, ReLU activation and max pooling in that order. The last three of these operations are optional and we tested networks both with and without these operations.

The Nao-Team HTWK network is similar to LeNet but with fewer outputs at each layer. It comprises two convolutional layers using 5×5 kernels and max pooling followed by two fully connected layers. We included four variations of the HTWK network as the authors did not specify whether or not Batch Normalization or ReLU activations were used. We also included the UT Austin Villa network which is a shallower network featuring just one convolution layer using 7×7 kernels and one fully connected layer. In total we evaluated these 252 designs based on the parameter options in Table 1.

Table 1. Deep CNN design parameters.

Design parameter	Values tested
Layers	2 layer networks: 1 conv layer and 1 FC or 1×1 conv layer 4 layer networks: 2 conv layers and 2 FC or 1×1 conv layers; 3 conv layers and 1 FC or 1×1 conv layers 5 layer networks: 3 conv layers and 2 FC or 1×1 conv layers
Convolutional layers	
Kernel size	$1 \times 1, 3 \times 3, 5 \times 5, 7 \times 7$ (7×7 only applied to 32×32 input patch)
Kernel dilation	1 or 2
Stride	1, 2, or 4
Output channels (kernels)	6, 8, 10 or 12
Pooling	None, Max pooling, or Average Pooling (Average pooling used only for the final 1×1 convolution layer)
Activation	None or ReLU
Batch normalization	Yes or no
Fully connected layers	
Layer outputs	16, 32, or 48
Activation	ReLU

4.2 XNOR-Net

The XNOR-Net architecture binarizes activations and kernel weights within a network so that the multiplications and additions in a convolution may be replaced by XNOR and bit counting (pop count) operations. A key element required to successfully train an XNOR-Net is the block structure of a convolutional layer which is different to block structure in a typical real-valued CNN in order to reduce the loss of information [3]. The modified convolution block for XNOR-Net therefore consists of the following blocks in the order specified: Batch Normalization, Binary Activation, Binary Convolution, and finally pooling.

The authors of XNOR-Net claim a dramatic 58× speedup when using XNOR based convolution in comparison to a normal real-valued convolution. This number does depend on the number of input channels and the kernel size and for our networks the number would be smaller (e.g. for 12 channels and 3×3 kernels the theoretically predicted speedup would be 40×). Achieving this speedup in practice is challenging but the method is attractive and for this reason we evaluate the impact of the XNOR-Net quantization on classification metrics for a subset of networks.

5 Results and Discussion

5.1 Real Valued Network Precision and Recall

Figure 2 summarizes the precision and recall classification metrics for all real-valued networks tested on the test set. It is clear that the recall performance is relatively insensitive to the network design parameters in the networks under test ($M = 97.2\%$, $SD = 1.5\%$). The precision performance is somewhat more variable. In a RoboCup setting, false positive ball detections are often more harmful than false negatives since they may lead to poor autonomous behavior decisions. Among the real-valued CNNs under test, thirty-two had a precision greater than 99%. A common feature of the networks with precision less than 90% was that none used ReLU activation or batch normalization. (This is relevant to XNOR-Net designs as binarized networks are inherently incompatible with ReLU activation.)

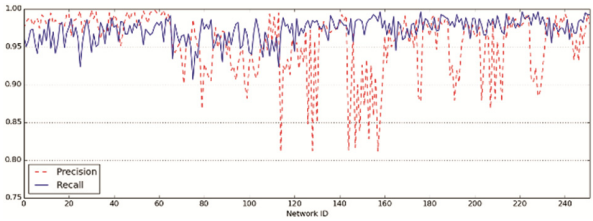

Fig. 2. Precision and recall for all real valued CNNs. Networks 0–65 use 12×12 input images, networks 66–241 use 20×20 input images, and the remainder use 32×32 input images. For each input dimension, the networks are sorted in ascending order by number of multiplications.

The network with the best overall classification performance measured by F_1 score (number 222) obtained 98.9% recall and 99.4% precision using a 5 layer network. With this data set, the HTWK network with ReLU and normalization (number 206) demonstrated 96.1% precision and 98.6% recall, the HTWK network without ReLU and normalization (number 204) scored a lower 92.4% precision with 96.6% recall, and UT Austin Villa's network (number 242) achieved 96.7% precision and 95.9% recall.

Inference times for the same networks when executed on the NAO robot are presented in Fig. 3. These were only measured for real-valued networks since the unoptimized XNOR-Net implementation used in this work performed floating point multiplications internally and provided no speed up. There is very little correlation between classification performance in Fig. 2 and inference time in Fig. 3 ($\rho = 0.13$). This suggests that, for ball detection, choosing a more complex network with a longer inference time is unlikely to be of much benefit. The networks for each input dimension are presented in order of the number of multiplications. The spikes in inference time correspond to networks with a larger number of weighted layers and smaller convolution kernels. The convolutions in the Caffe framework are performed using BLAS matrix multiplication, as such a large number of multiplications can be combined into one matrix multiplication. Therefore more BLAS calls with fewer multiplications per call will be slower.

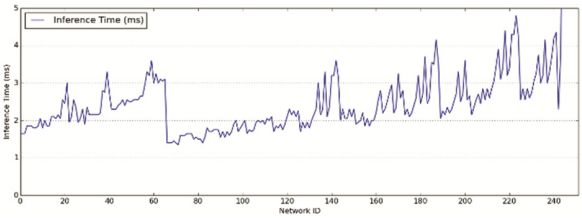

Fig. 3. Inference time on the NAO for all real valued networks evaluated (0–65 use 12×12 input images, 66–241 use 20×20 input images, and the remainder use 32×32 input images). Inference times larger than 5 ms are not shown in the figure.

On the other hand, although network 66 produced the fastest inference time of 1.4 ms, its balance of F_1 score performance and inference time was in the bottom 18% of all networks tested. In contrast, the inference time of network 222, which had the best overall classification performance, was rather long at 4.8 ms.

The best balance of overall performance was obtained for network 16 whose inference time was 2.05 ms and whose precision and recall were 98.1% and 98.0% respectively. The design of this network features a 12×12 input patch size and 2 convolutional layers with twelve 3×3 kernels each. Each convolutional layer also included ReLU activation, batch normalization and 2×2 max pooling. The convolutional layers were followed by 2 fully connected layers having 32 and 2 outputs respectively.

For comparison, the inference times of the HTWK network with ReLU and normalization (number 206), HTWK network without ReLU and normalization (number 204), and UT Austin Villa network (number 242) were 2.7 ms, 2.2 ms, and 2.3 ms respectively.

5.2 XNOR-Net Performance

The classification statistics of a small number of XNOR-Net designs corresponding to real-valued networks already tested were also evaluated. In general XNOR-Net designs exhibited greater sensitivity to the training parameters chosen and often failed to converge or had poor performance when training parameters derived from the equivalent real-valued networks were used.

Figure 4 indicates that XNOR-Nets have degraded classification performance compared to equivalent real-valued CNNs, as expected, and attain average scores that are almost 12% lower. In general more complex networks with more weights in hidden layers were more robust to the destructive effect of binary quantization. XNOR-Nets use binary activation rather than ReLU activation and it is possible that this is a contributor to the poor performance as the lack of ReLU activation was associated with the worst precision statistics for real-valued networks.

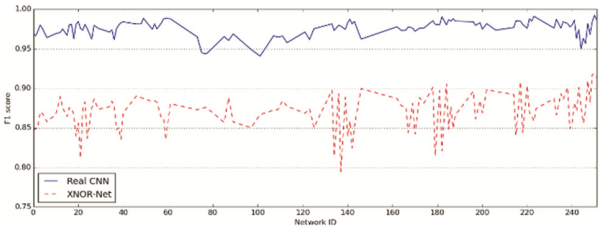

Fig. 4. Comparison of F_1 score for real-valued networks and corresponding XNOR-Net designs.

5.3 Recall Performance for Different Ball Detection Scenarios

We examined the recall performance of all networks in more detail by examining the recall for different subsets of test images that were grouped by ball detection scenario. The scenarios examined were ball far away (more than 3 m), ball in free space, ball moving, ball in natural light, ball on or near a line, and ball occluding, occluded by, or near a robot.

Figure 5 summarizes the results and shows that performance was quite consistent across the scenarios. Nevertheless, moving balls or balls in natural light or far away provide the biggest detection challenges to the networks. Somewhat surprisingly, the scenarios that provide the greatest challenge to our existing heuristic based ball detection, namely ball on line and ball on robot, are handled very well by the majority of networks.

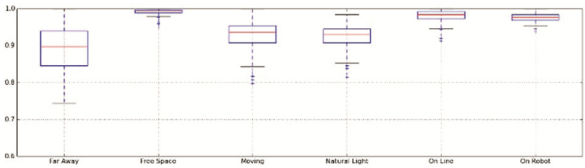

Fig. 5. Recall performance of all real-valued networks across different ball detection scenarios.

6 Conclusion and Future Work

This work presented a data set for benchmarking ball detection in RoboCup soccer. Full images with labelled ball and non-ball regions have been published so that the entire vision pipeline may be tested, but in this work we focused on one particular aspect of that pipeline, namely, classification of candidate regions as ball or non-ball using Deep CNNs.

We trained a range of networks spanning a parameter space that varied the number of weighted layers, the kernel sizes, and the numbers of outputs at each layer among other parameters. We found that deeper networks with more channels in the hidden layers do not necessarily lead to better accuracy but does increase inference time. We conclude that the network classification performance is relatively insensitive to the network design for this ball detection problem.

This work focused on analyzing the classification performance of XNOR-Net and did not use an optimized implementation that could benefit from the binary weights and activations. We found that XNOR-Net architectures had an F_1 score that was 12% lower than the corresponding real valued network on average. The theoretically predicted speed up (by replacing real multiplications with binary XNOR) for our CNN layers is between 29× and 40×. This speed up could allow more image patches to be evaluated within the available time budget on the robot or to enable substantially more complex networks to be feasibly executed. If more image patches can be evaluated during each cycle, then this work could extend to classifying additional field objects such as robots and goal posts to the architecture. For this reason we intend to examine the feasibility of a sufficiently optimized implementation on the Intel Atom processor as part of our future work.

Acknowledgements. The authors would like to acknowledge the valuable inclusion of labelled images in the data set from the final year project work of Robert McCraith. The authors would like to gratefully acknowledge funding provided by the Irish Research Council under their Government of Ireland Postgraduate Scholarship 2013.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)

2. Raina, R., Madhavan, A., Ng, A.Y.: Large-scale deep unsupervised learning using graphics processors. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 873–880 (2009)
3. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. arXiv preprint arXiv:1603.05279 (2016)
4. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control., Signals, Syst. (MCSS) **2**, 303–314 (1989)
5. Seide, F., Li, G., Yu, D.: Conversational speech transcription using context-dependent deep neural networks. In: Interspeech, pp. 437–440 (2011)
6. Hanson, S.J., Pratt, L.: Comparing biases for minimal network construction with back-propagation. Adv. Neural. Inf. Process. Syst. **1**, 177–185 (1989)
7. LeCun, Y., Denker, J.S., Solla, S.A., Howard, R.E., Jackel, L.D.: Optimal brain damage. In: NIPs, pp. 598–605 (1989)
8. Hassibi, B., Stork, D.G.: Others: second order derivatives for network pruning: optimal brain surgeon. In: Advances in Neural Information Processing Systems, pp. 164–164 (1993)
9. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
10. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)
11. Szegedy, C., et al.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)
13. Gong, Y., Liu, L., Yang, M., Bourdev, L.: Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115 (2014)
14. Arora, S., Bhaskara, A., Ge, R., Ma, T.: Provable Bounds for Learning Some Deep Representations. ICML. pp. 584–592 (2014)
15. Hwang, K., Sung, W.: Fixed-point feedforward deep neural network design using weights $+1, 0$, and -1. In: Signal Processing Systems (SiPS), IEEE Workshop on 2014, pp. 1–6 (2014)
16. Courbariaux, M., Bengio, Y., David, J.-P.: Training deep neural networks with low precision multiplications. arXiv preprint arXiv:1412.7024 (2014)
17. Courbariaux, M., Bengio, Y., David, J.-P.: Binaryconnect: training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems, pp. 3123–3131 (2015)
18. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: training deep neural networks with weights and activations constrained to $+1$ or -1. arXiv preprint arXiv:1602.02830 (2016)
19. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM International Conference on Multimedia, pp. 675–678 (2014)
20. HTWK, N.-T.: Team Research Report. http://robocup.imn.htwk-leipzig.de/documents/TRR_2016.pdf?lang=en. (2016)
21. UT Austin Villa Code Release. https://github.com/LARG/spl-release (2016)
22. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**, 2278–2324 (1998)
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
24. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
25. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of 27th International Conference on Machine Learning (2010)