In *RoboCup-2017: Robot Soccer World Cup XXI*, (RoboCup 2017), Nagoya, Japan, July 2017

Fast and Precise Black and White Ball Detection for RoboCup Soccer

Jacob Menashe, Josh Kelle, Katie Genter, Josiah Hanna, Elad Liebman, Sanmit Narvekar, Ruohan Zhang, and Peter Stone

{jmenashe,jkelle,katie,jphanna,eladlieb,sanmit,zharucs,pstone}@cs.utexas.edu

The University of Texas at Austin

Abstract. In 2016, UT Austin Villa claimed the Standard Platform League's second place position at the RoboCup International Robot Soccer Competition in Leipzig, Germany as well as first place at both the RoboCup US Open in Brunswick, USA and the World RoboCup Conference in Beijing, China. This paper describes some of the key contributions that led to the team's victories with a primary focus on our techniques for identifying and tracking black and white soccer balls. UT Austin Villa's ball detection system was overhauled in order to transition from the league's bright orange ball, used every year of the competition prior to 2016, to the truncated icosahedral pattern commonly associated with soccer balls.

We evaluated and applied a series of heuristic region-of-interest identification techniques and supervised machine learning methods to produce a ball detector capable of reliably detecting the ball's position with no prior knowledge of the ball's position. In 2016, UT Austin Villa suffered only a single loss which occurred after regulation time during a penalty kick shootout. We attribute much of UT Austin Villa's success in 2016 to our robots' effectiveness at quickly and consistently localizing the ball.

In this work we discuss the specifics of UT Austin Villa's ball detector implementation which are applicable to the specific problem of ball detection in RoboCup, as well as to the more general problem of fast and precise object detection in computationally constrained domains. Furthermore we provide empirical analyses of our approach to support the conclusion that modern deep learning techniques can enhance visual recognition tasks even in the face of these computational constraints.

Keywords: Machine Learning, Deep Learning, Classification, Ball Detection

1 Introduction

RoboCup, or the International Robot Soccer World Cup, is an international robotics competition introduced in 1997 as a research initiative to advance the fields of robotics and artificial intelligence using soccer as a motivating challenge domain [5].

UT Austin Villa participates in the Standard Platform League (SPL) of the RoboCup competition. In 2016, the SPL league moved from a bright orange soccer ball, which had been used all previous years, to a classic truncated icosahedral black and white soccer ball.

This shift presented a significant challenge to the SPL teams; black and white are prominent colors throughout the field, whereas the previous orange ball could be easily distinguished from the surrounding environment using an RGB thresholding approach. With an accurate threshold it was a relatively simple matter to identify all of the "orange" pixels in the camera frame, and due to their sparseness, thoroughly evaluate each orange cluster to determine whether it was a ball. Since black, white, and gray are prominent colors on the soccer field, identifying a black and white soccer ball is a far more challenging task. The image is generally filled with clusters of black and white and thus it is too computationally expensive to scrutinize every cluster of black or white pixels. As described in Section 3, detecting a pattern is an involved process that is computationally expensive and thus great care must be taken to preserve computational capacity and maximize the payoff of investigating any particular ball candidate. This transition from orange to black and white thus rendered the problem of consistent ball detection significantly more difficult than in previous years. Ultimately, however, our new ball detection algorithm proved to be as good as or better than that of other teams at the competition and allowed UT Austin Villa to place second at the international RoboCup competition in Germany and first at the RoboCup US Open competition in Maine.

In this work we discuss UT Austin Villa's approach to solving the challenge of black and white ball detection with detailed analyses of the various components of our detection system. After we cover the necessary background work in Section 2 we proceed to Section 3 where we cover the main ball detection loop.

In Section 4 we provide empirical analyses of our ball detector and then summarize and conclude in Section 5.

2 Background

In this section, we first introduce RoboCup and the SPL, as well as the various challenges teams in the SPL have undertaken in recent years in order to stay competitive. Later in this section, we introduce heuristic approaches to this year's main challenge and briefly discuss Machine Learning as a possible complement to heuristics.

2.1 RoboCup Soccer Challenges

RoboCup has many leagues including multiple simulation leagues and various physical robot soccer leagues. Almost all of the leagues involve robot agents that must act autonomously in their environments.

UT Austin Villa has participated in a physical robot soccer league called the Standard Platform League since 2008. The SPL is different from other RoboCup leagues in that all teams must use the same robot platform, namely the Softbank

NAO Robot. The NAO uses a single-core 1.6 GHz Intel x86 processor and a pair of 1280x960 non-stereoscopic cameras running at 30Hz.

At RoboCup 2016, teams competed in 5 vs. 5 soccer games on a 9 meter by 6 meter soccer field. Each game consists of two 10-minute halves. Teams must play completely autonomously — no human input is allowed during games outside of game state signals sent by an official to communicate to the robots when a goal has been scored, when they have been penalized, etc. The robots on each team are additionally allowed to communicate with each other over a wireless network.

Historically, there have been many vision-related challenges since the SPL began using the NAO robots in 2008. In 2008, games were played with an orange street hockey ball on a 6 meter by 4 meter field. This field had a blue goal and a yellow goal to allow teams to easily differentiate sides. Over time, the playing environment has become less and less color-coded. After a few changes in field size that eventually resulted in a 6 meter by 4 meter playing surface, in 2012 both goals became yellow. This change required teams to find ways other than goal color to differentiate the two sides of the field. The field size was increased again in 2013 — to the current 9 meters by 6 meters — and then in 2015 white goals were introduced.

In 2016, the league removed the final color-coded landmark when they switched from playing with an orange street hockey ball to a black and white printed soccer ball. This paper discusses the research problems this change presented, as well as how we successfully handled these problems to finish 2nd at RoboCup 2016.

Due to the hardware constraints of the NAO and the characteristics of the SPL, the problem of visual object detection rests on two primary criteria:

1. Efficiency: Detectors process 1280x960 images faster than 30Hz.
2. Precision: Detectors must minimize false positives.

The difficulty of completing the full sensing and action loop at 30Hz is compounded by the fact that processing takes place sequentially on a single CPU core. UT Austin Villa's codebase in particular allocates approximately 10ms for the entire ball detection pipeline. Even performing a full image read takes 14ms, so care must be taken to minimize reads and isolate computations to the areas of the image most likely to yield results.

However, computational efficiency cannot come at the expense of precision. While a false negative may delay action for a short period of time, a single false ball detection can lead robots astray and cause the team to pursue a ball that doesn't exist. UT Austin Villa's detection system therefore aggressively prunes out false detections at the possible cost throwing out correct detections on occasion.

2.2 Heuristic Approaches

When deciding whether an object in the visual frame belongs to one class or another there are generally two possible approaches to take: hand-designed heuristic methods, and machine learning models.

Heuristic methods tend to be ad-hoc in nature, as finding the right combination of features involves a great deal of trial and error. For example, one might identify whether an object is a soccer ball by considering the patterns identified on the object and testing how close they are to the expected patterns of the ball. This general technique can be easy (i.e. computationally efficient) to compute and is common in the RoboCup domain.

Even though machine learning techniques have been very successful in object detection and recognition tasks [9], heuristic methods are still relevent because of their computational efficiency. Rather than apply a traditional "sliding window" approach, our algorithm incorporates machine learned models by first applying a high level heuristic search that quickly identifies ball candidates for further analysis, and progressively applies more rigorous filters to throw out candidates that are likely to be false positives. Finally the algorithm applies a more computationally expensive machine-learned ball classifier to verify that the selected candidate is indeed a soccer ball.

2.3 Machine Learning Approaches

Machine Learning (ML) is a more principled approach that is supported by a wide variety of algorithms in the computer vision literature. Examples of popular models are Support Vector Machines (SVMs) and Deep Neural Networks (DNNs) which map vectors of image pixels to discrete classifications. Many ML algorithms require anywhere from tens of thousands to millions of labeled samples for successful training. While these techniques are generally more accurate and robust than heuristic approaches, they must be provided with properly cropped input images that derive from either a sliding window or a region-of-interest (ROI) detection system. The latter of these has much greater potential for computational efficiency and, as we will describe below, is the most effective method of incorporating ML into RoboCup vision tasks such as soccer ball detection. Rather than using ML as a complete alternative to heuristics, in this work we show that ML can be combined with heuristics to strike an effective balance between generality, accuracy, and computational efficiency.

3 Ball Detection

In this section we describe the complete ball detection algorithm. Within the constraints of available space, we describe our approach with the goal of enabling full reimplementation. To complement our description, we also provide the complete source code from our implementation[1].

3.1 Ball Candidates

As shown in Algorithm 1, the ball detection algorithm can be broken down into six subroutines that are aimed at progressively refining an estimate of the ball's

[1] Source code is hosted at: `https://github.com/LARG/spl-release`

position. The first phase, seen on line 2, consumes a raw image and produces a set of regions with exceptionally high contrast. Line 3 of the algorithm iterates over these regions to identify dark blobs which intuitively correspond to the black pentagons on the soccer ball. This produces one set of blobs for each region. Line 4 organizes each set of blobs into one triplet per set; this triplet corresponds to a triangle formed by 3 pentagons. Line 5 of the algorithm selects the triangle whose characteristics most closely fit those of the soccer ball. Line 6 of the algorithm applies a Hough transform to estimate the ball's center and radius.

Finally a machine-learned classifier is used to filter out false positives in line 7. The remainder of this section describes these phases in further detail.

Algorithm 1 The ball detection algorithm described throughout Section 3.

```
1: function DetectSoccerBall(RawImage i)
2:     R_HC ← DetectHighContrastROIs(i)
3:     P ← DetectBlackPentagons(R_HC)
4:     T ← ConstructTriangles(P)
5:     t ← SelectBestTriangle(T)
6:     b ← HoughCorrection(t)
7:     return b if Classify(b) else return null
8: end function
```

Line 2 [DetectHighContrastROIs]: The motivation behind computing regions of interest is speed. We cannot devote expensive image processing time to the whole image, so the algorithm first quickly narrows its focus to smaller subsets of the image. Compute time is then allocated to these smaller ROIs. Our ROI detector algorithm is based on the fact that the ball will produce regions of high contrast because it has black spots on a white surface. We identify areas of high contrast by using adaptive thresholding [1]. Adaptive thresholding requires two parameters - window size (measured in pixels) and a threshold value. Optimal window size depends on the size (in pixels) of the ball in the image, which in turn depends on the ball's position relative to the camera and the robot's orientation. A ball that is far away from the robot will appear smaller in the image and thus requires a smaller window size. To address this issue, the image is split into 3 parts - top, middle, and bottom, corresponding to far, medium, and close balls. Adaptive thresholding is applied to each part independently with different window sizes and thresholds. Window sizes are selected to be approximately 1.6 times the projected ball size in that portion of the image.

Adaptive thresholding produces a set of response pixels - pixels that exceed the adaptive threshold. The algorithm ignores response pixels that are classified as green in the original color image because they are unlikely to have been captured from the ball. The algorithm clusters the non-green response pixels by Euclidean distance to one another. Any two pixels which have Euclidean distance at most 0.6 times the expected ball size are put into the same cluster. Each cluster is then converted to an ROI bounding box whose center is set to the cluster's centroid, and whose width and height is set to the expected ball

diameter plus padding. ROIs whose projected world coordinates are greater than 20 meters away are not returned.

Line 3 [DetectBlackPentagons]: In this phase, each ROI is evaluated independently with more computation to determine if a ball might be present. The first step is to identify the black pentagons of the ball. Blobs are computed by partitioning each row of pixels into either dark or light scanlines, and then merging scanlines into blobs based on the union find algorithm [2].

Scanlines are computed on the subsampled grayscale image, moving from left to right along a row of pixels. The first and last scanlines of the row are assumed to be white because the ROIs were padded to contain the entire ball. Because of imperfect lighting conditions, the black spots on the ball don't have a consistant grayscale value. Thus, the algorithm looks for pixel segments that are dark relative to their neighboring pixels. A color change is triggered when the percent difference between two adjacent pixels is more than 25% of their average color. We refer to this percentage as the *Blob Threshold* and examine alternative values in Section 4.1. A value of 25% enables the algorithm to be somewhat robust to varying lighting conditions. After every row of the ROI has been segmented into light and dark scanlines, they are then merged into blobs.

It is often the case that non-ball objects in the image will create blobs. We attempt to filter out these erroneous blobs according to the following heuristics:

1. The blob must cover at least some percentage of the area of its bounding box.
2. The blob cannot be too big or too small, relative to the expected ball size.
3. The blob's bounding box aspect ratio cannot be too narrow or too wide.
4. The blob cannot have too many green pixels.
5. The average intensity of the blob's constituent pixels must be below some threshold.

Line 4 [ConstructTriangles]: We make the assumption that at least 3 of the ball's black spots will be visible regardless of ball orientation, and that these 3 of the black spots will form a nearly-equilateral triangle. We group 3 blobs together into a "BlobTriangle" and apply the following series of heuristic tests to filter out BlobTriangles that probably aren't the result of a true ball. Thresholds and specific criteria can be found in our accompanying source code release.

1. Relative Size Test: The ratio between the areas of the smallest blob and largest blob cannot exceed some threshold.
2. Angle Test: The largest angle of the triangle formed by the three blob centroids cannot exceed some threshold. This angle is recorded as the BlobTriangle's score. A smaller score is considered better because smaller angels are more similar to the ideal equilateral triangle. We notice in practice that many false BlobTriangles are farther from equilateral than true BlobTriangles.
3. Float Test: The projected height of the ball must not be too far above or below the ground.
4. Intersecting Blob Test: No other blobs are allowed to be inside the triangle.

If the BlobTriangle passes all of these tests, then it proceeds to the next round of tests. If there are more than 3 blobs in the ROI, every combination of 3 is evaluated and the BlobTriangle with the largest score is chosen for its enclosing ROI.

Line 5 [SelectBestTriangle]: Next, the algorithm identifies the best ROI by computing the following real-valued heuristic features from the ROI's best BlobTriangle. No single threshold is applied to any of these values because they are filtered in aggregate [8].

1. The percentage of green pixels in an imaginary box below the BlobTriangle. True balls usually have a large amount of green field directly below them, so a larger value here indicates higher likelihood.
2. Percentage of green pixels in the BlobTriangle's bounding box. True balls usually have very few green pixels on them, so a smaller number here indicates higher likelihood.
3. Projected Ball Height: Similar to the Float Test, but this time with a soft threshold.
4. Distance from the field. The center of the BlobTriangle in image coordinates is projected into world coordinates. The farther this projected position is from the field, the less likely it is to be a true ball.
5. Ball velocity, as measured by the difference in world distance coordinates between this BlobTraingle and the previously observed ball.
6. ***KW*** *Discrepancy*: The discrepancy between **K**inematics-based and **W**idth-based distance computations (see Menashe et al. [8]).

These values are fed through a multidimensional Gaussian estimator which has hard-coded means and standard deviations for each of the 6 features [8]. This gives a likelihood estimate for the BlobTriangle being the result of a true ball. If this likelihood estimate is below a certain threshold, we throw it out and no longer consider it to be a ball candidate. If multiple ROIs pass this test, then only the ROI with best BlobTriangle score will proceed to the next round. We refer to the precise threshold for the likelihood estimate as the *Gaussian Likelihood Threshold* and examine different settings of this variable in Section 4.1.

Line 6 [HoughCorrection]: Before applying the final test, the algorithm uses a Hough transform to correct the estimated position and radius of the ball represented by the best triangle. This operation is only applied to the single best candidate due to being computationally expensive.

We want the resulting circle to conform to the ball's contour. However, the Hough transform sometimes "snaps" to the contours of the black spots on the ball due to their high contrast and similarity with normal edges. This problem is partially alleviated by overwriting the pixel values of these blobs to a ligher gray color; in this way the algorithm "erases" the dark blobs from the ROI, and is thus less likely to produce unwanted edges during the Hough transform. Figure 1 shows an example of this.

Line 7 [Classify]: In this final test, the ROI image is classified using a machine learned classifier with low false negative rate. If the classifier's prediction is positive, then we consider this ROI/BlobTriangle pair to be a ball, and we signal a ball observation to the rest of the system. Section 3.2 describes the classification step in further detail.

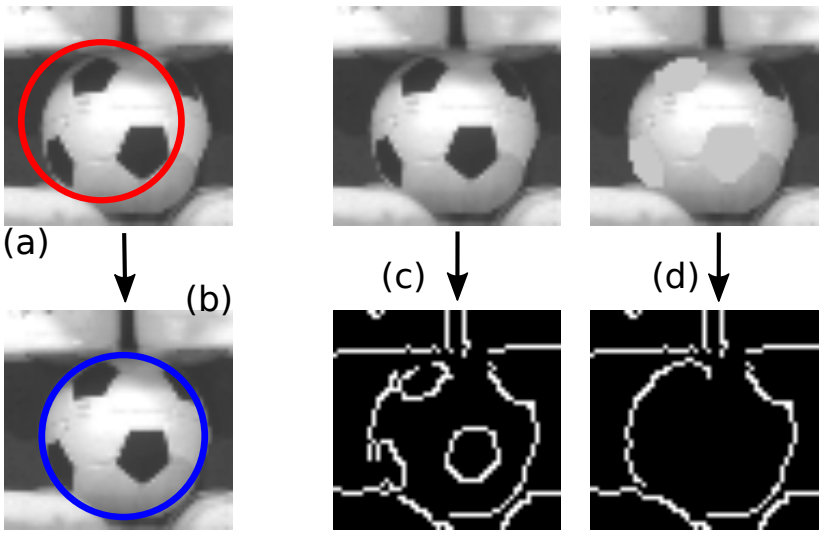

Fig. 1: (a) Ball localization based on BlobTriangle location. (b) Ball localization after Hough Transform. (c) Edge image without erasing dark blobs. (d) Edge image after erasing dark blobs.

3.2 Classification

The last step of the filtering process described in Section 3.1 uses a general purpose machine learning technique to perform a final test on a potential ball detection. The algorithm uses a machine-learned classifier (MLC) - either a Deep Neural Network (DNN) or a Support Vector Machine (SVM). In Section 4.2 we evaluate the relative effectiveness of these two approaches.

In this work, MLCs are trained with binary labels: 0 (or "negative") indicating that the input image does not contain a ball, and 1 (or "positive") indicating that the input image does contain a ball. Input images are taken from the ROI detection system described in Section 3.1, converted to the MLC's expected input format, and then processed by the MLC. Rather than evaluate every ROI, the computational complexity of our MLCs only allows for a single validation per image frame. Thus, we simply take the highest-scoring candidate and apply our MLC validation step to the candidate as a way to further reduce our false positive rate.

Support Vector Machines One of the two MLCs we evaluate in this work is the Support Vector Machine. In order to train the SVM we collected our training dataset in two phases. In the first phase we used our candidate identification system to collect positive sample images of the ball, and manually removed false positives from the dataset. We were able to gather around 1,000 positive samples in this manner.

We collected negative samples by capturing images from the NAO's camera at various points on the field, taking care to ensure that none of the images had balls in their field of view. We then randomly divided these images into ROIs and supplied these ROIs as negative samples. We produced approximately 15,000 negative samples through this process.

We used OpenCV's Nu-SVM implementation with a linear kernel. Images were resized to 32-by-32 pixel grayscale and then transformed into a vector of 8-by-8 HoG descriptors. Training time generally ranged from 15 to 30 minutes on a modern laptop with an Intel i7 processor. Training time increased exponentially as more negative samples were added so in practice we were not able to go beyond the 15,000 noted above.

Deep Neural Networks We also applied Deep Neural Networks to the binary ball classification problem using similar data collection techniques as with SVMs (Section 3.2), We used the Caffe Deep Neural Network implementation [4] on raw pixels with GPU training enabled. With this approach we were able to train the ball detector within about 2 hours on a typical laptop with GPU hardware.

In order to enable fast test-time processing on the NAO's hardware we used a simple DNN consisting of a single Gaussian convolutional layer. Even such a simple network required 3ms to test a single image, which was comparable with the test time for our SVM implementation. At the RoboCup competition we found that the SVM showed a lower false negative rate than the DNN, which was perhaps due to the simplicity of our DNN structure. Section 4.2 provides a complete comparison based on rigorous experiments.

4 Experiments

In this section we provide empirical analysis of different variable settings and MLC techniques to evaluate the options available when implementing the ball detection algorithm. In Section 4.1 we first look at the effect of varying one of the primary thresholds used in filtering out ball candidates. In Section 4.2 we examine the accuracy of both SVMs and DNNs to show the preferred technique to be used given one's computing and dataset constraints.

4.1 Ball Candidate Detection

Figure 2 shows the effect of altering the Gaussian likelihood threshold described in line 5 of Algorithm 1. This likelihood estimate threshold is described in detail by Menashe et al. [8] where it is also used for ball detection. The figure shows that recall is consistent except for very high threshold values, indicating that true positives consistently exhibit a high likelihood estimate and are seldom filtered by this step of the algorithm.

4.2 SVM and Deep Classification

Deep Neural Network as Ball Classifier Deep neural networks, and convolutional neural networks in particular [6, 7], have demonstrated great success

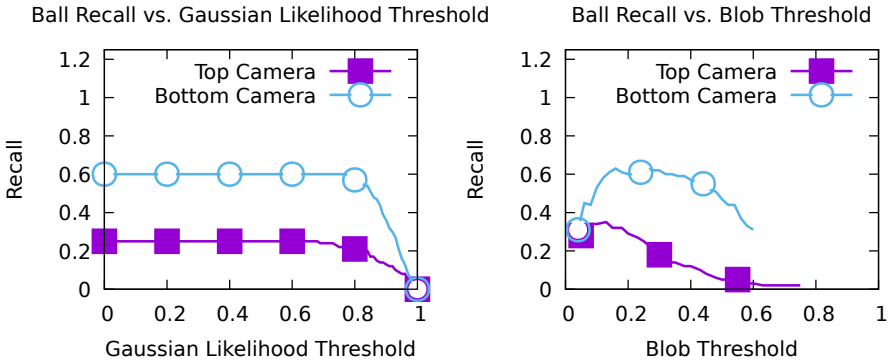

Fig. 2: A plot of Recall vs Gaussian Likelihood and Blob thresholds for ball candidates used in Algorithm 1.

in visual recognition tasks. Due to the limited computational capacity of the robot, we design four simple neural network structures to perform ball classification. The first two networks are convolutional neural networks. They each have one convolution layer, a max-pooling layer, and a fully connected layer. The first network, which we will refer as Conv-1, learns 4 9x9 convolution filters with stride length 5. While the second one, Conv-2, has 32 7x7 convolution filters with stride length 4. The third network has a single fully connected layer. The fourth network has two fully connected layers, in which the hidden layer has 512 neurons. We will refer these two networks as Fc-1 and Fc-2. The dataset we use for evaluation was collected at two RoboCup competitions and has a total of 34,684 annotated images with 7,666 positives and 30,018 negatives. Table 1 shows the training times, model complexity, and efficacy results for each network.

	Time	#Params	Precision	Recall	Accuracy
Conv-1	320s	**1106**	.9797	.9746	.9907
Conv-2	213s	5314	**.9948**	**.9941**	**.9977**
Fc-1	**12s**	6146	.9251	.9341	.9712
Fc-2	116s	1574402	.9914	.9772	.9936

Table 1: Classification results of neural network classifiers.

We also test the transferability of the networks across different environments, since in RoboCup games could in different locations. We train the networks on a dataset collected in one environment and 1) test on some other data collected in the same environment; and 2) test on a dataset collected in a different envi-

ronment, without any additional fine-tuning. The results are shown in Table 2a and 2b. Overall, the performance of all networks degrades considerably due to overfitting to a particular environment, especially the convolutional networks. This implies it is desirable to have deep networks fine-tuned or retrained when environment changes, which is practical given the training time in Table 1.

	USopen16 →		USopen16
	Precision	Recall	Accuracy
Conv-1	.9972	**1.000**	.9994
Conv-2	**.9991**	.9991	**.9996**
Fc-1	.9346	.9468	.9746
Fc-2	.9944	.9944	.9976

	USopen16 →		RoboCup16
	Precision	Recall	Accuracy
Conv-1	.1226	.1440	.6726
Conv-2	.4163	.8748	.7654
Fc-1	**.7349**	.8596	**.9218**
Fc-2	.7214	**.8923**	.9215

(a) Transferability results of our DNN classifiers, using USopen 2016 dataset as source task and RoboCup 2016 dataset as target task.

	RoboCup16 →		RoboCup16
	Precision	Recall	Accuracy
Conv-1	.9820	.9776	.9928
Conv-2	**1.000**	**.9910**	**.9984**
Fc-1	.9623	.9731	.9884
Fc-2	.9977	.9843	.9968

	RoboCup16 →		USopen16
	Precision	Recall	Accuracy
Conv-1	.6754	.2576	.8109
Conv-2	**.9890**	.3925	.8664
Fc-1	.8402	.5926	.8865
Fc-2	.9199	**.6064**	**.9026**

(b) Transferability results of our DNN classifiers, using RoboCup 2016 dataset as source task and USopen 2016 dataset as target task.

Support Vector Machine as Ball Classifier Since their inception in the early 1990s, SVMs have proven to be a robust and powerful family of classifiers[3]. In these experiments we use the datasets from the DNN experiments to compare SVMs with three different kernels: linear, polynomial (degree 3), and radial basis function (RBF). The results are presented in Table 2a.

As we have done for deep nets, we also wish to measure the transferability, or generalization, of learning across domains. In these experiments we train the SVM models on a dataset collected in one environment and test on a dataset collected in a different one. These results are presented in Table 2b. Overall, the performance degrades dramatically, more so than in the neural networks case, due to overfitting to a particular environment, without the flexibility that the deep architectures offer. Again this implies that one must adjust or retrain the existing SVM models when presented with a new environment in order to maintain acceptable performance.

4.3 SVM versus DNN

There are two major lessons to take away from the SVM and DNN experiments of Section 4.2. First, transferred models perform poorly relative to models that are trained on datasets similar to their test sets. In the context of RoboCup, this means that it is highly beneficial to train models on-site at the competitions in order to train on data that will closely resemble what is seen in the official games.

Second, although complex multi-layer DNNs are too computationally expensive for constrained domains such as RoboCup, simpler networks can still provide

SVM Kernel	Accuracy	AUC	Precision	Recall
SVM kernel	Accuracy	AUC	Precision	Recall
Linear	0.883	0.869	0.833	0.543
Polynomial	**0.970**	**0.990**	**0.972**	**0.881**
RBF	0.961	0.989	0.989	0.824

(a) Classification results of SVM classifiers.

SVM Kernel	USOpen16				USOpen16→RoboCup16			
	Accuracy	AUC	Precision	Recall	Accuracy	AUC	Precision	Recall
Linear	0.893	0.923	0.807	0.655	**0.595**	**0.420**	**0.163**	**0.325**
Polynomial	**0.985**	0.997	0.962	**0.968**	0.363	0.140	0.098	0.027
RBF	0.982	**0.998**	**0.972**	0.944	0.421	0.134	0	0
	RoboCup16				RoboCup16→USOpen16			
Linear	0.992	0.997	0.995	0.960	0.781	0.387	0.200	0.001
Polynomial	0.998	0.999	**1.00**	0.987	**0.782**	**0.520**	**1.00**	**0.001**
RBF	**0.998**	**0.999**	0.998	**0.991**	0.782	0.337	1.00	0.001

(b) Generalization results of our SVM classifiers. In the first stage we train a model on 80% of the source dataset and then test on the remaining 20%. We then test the trained model on the target (transfer) dataset.

an advantage over SVMs. In particular we see that the DNN recall observed in Table 1 is signifcantly higher than the SVM recall in Table 2a, while both techniques exhibit high accuracy.

To summarize, we find that a well-trained DNN can outperform a well-trained SVM on object classification tasks, even when the DNN's architecture is simplified to enable fast computation.

5 Conclusion and Future Work

In this work we have described UT Austin Villa's black and white soccer ball detection algorithm and discussed in detail the heuristic techniques applied as well as the machine-learned classification algorithms we incorporated for optimizing detector accuracy. In addition to our proven success at gameplay we provided empirical results indicating the benefits of applying a Deep Neural Network to the task of detecting false positives. We showed that a modest DNN architecture along with a heuristic ROI filtering pipeline can be combined to create a fast, precise object detection system that is suitable for computationally constrained environments such as RoboCup soccer.

As the RoboCup competition (and the SPL in particular) progresses toward more challenging and realistic requirements, lighting invariance will take on a greater role in vision algorithms in the coming years. Our work is designed to be robust to changes in lighting conditions, particularly in the area of geometric checks. However, improvements may still be possible with respect to adaptive thresholding and green detection in our color table. Improving these components will be the subject of future work.

Bibliography

[1] John Bernsen. Dynamic thresholding of grey-level images. In *International conference on pattern recognition*, volume 2, pages 1251–1255, 1986.

[2] James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2061–2066. IEEE, 2000.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. ISSN 1573-0565. doi: 10.1007/BF00994018. URL `http://dx.doi.org/10.1007/BF00994018`.

[4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[5] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, AGENTS '97, pages 340–347, New York, NY, USA, 1997. ACM.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[8] Jacob Menashe, Samuel Barrett, Katie Genter, and Peter Stone. Ut austin villa 2013: Advances in vision, kinematics, and strategy. In *The Eighth Workshop on Humanoid Soccer Robots at Humanoids 2013*, 2013.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.