Fast Multi-Scale fHOG Feature Extraction Using Histogram Downsampling

Authors removed for blind review

Abstract

We describe an adaptive fHOG feature pyramid construction scheme based on histogram downsampling. Varying the pyramid level to which the scheme is applied gives control over the trade-off between precision or speed. We evaluate the scheme on a modern computer and on a NAO humanoid robot in the context of the RoboCup competition, *i.e.* robot and soccer ball detection, in which we obtain 1.57x and 1.68x increase on PC and robot respectively in pyramid construction speed without any loss in detection performance relative to the baseline that does not use feature approximation. The scheme can be adapted to increase speed while trading off precision until it reaches the conditions of a state of the art feature approximation method.

Object detection has seen tremendous progress in the past years, which stemmed both from hand-crafted features and the relatively recent convolutional neural networks. As CPUs become more and more powerful and with the advent of cheaper GPU processing, some subfields witnessed super-human image recognition accuracy.

Our focus is however on conditions where the system is required to function with low computational resources. Such conditions can be found in the Standard Platform League (SPL) of the RoboCup soccer competition. Here, teams must use the commercially available SoftBank NAO humanoid robot without hardware modifications. The resources available in this setup are an Intel Atom 1.6GHz processor with one thread and 1GB of RAM. Although sufficient for vision, these resources must be shared by several processes to perform locomotion and strategic behavior necessary for the soccer match, hence in reality, vision can only account for roughly 5% CPU when the robot is in motion.

The main tasks for computer vision in this competition are the detection of lines, goal posts, the ball and other robots (teammates and opponents). For humanoid detection we turn to the subfield of pedestrian detection, where Histograms of Oriented Gradients (HOG) continue to play an important role since their invention in 2005 (Dalal and Triggs 2005). HOG consists in dividing an image into cells, computing the gradient of each cell, binning gradients into a histogram of main orientations and finally normalizing over blocks of cells to produce features (originally of 36-dimensions) that are used for training a Support Vector Machine (SVM) classifier over a sliding window. Later, (Felzenszwalb et al. 2010) introduced a refined version of HOG features (commonly referred to as Felzenswalb HOG or fHOG) which proved more robust for pedestrian but also generic object detection. Despite its high popularity however, HOG-like (HOG, fHOG, or other variants) feature extraction is known to be slow.

While many other flavors of object detectors have been proposed, most combine HOG-like features with other types to obtain better results under different conditions. Our main focus is to accelerate fHOG feature extraction in order for it to run on a robotic platform with low computational resources, while maximizing detection precision.

We first discuss existing work on feature approximation and how our work differs from the state of the art methods. We then present a series of evaluations on a modern computer (Intel Core i7, 2.60GHz 6MB) to observe how our choices have an impact on detection quality and execution speed. All results are obtained using a single execution thread and averaged over multiple runs to ensure validity. From these evaluations we make several observations that lead to an adaptive scheme that gives control over the trade-off between average precision and execution speed. Finally we validate the results on the chosen robotic platform, where we obtain higher execution time compared with the modern computer as expected, due to the lower quality processor, but still observe significant improvement relative to the baseline.

Related work

Our work falls in the concept of feature approximation, notably used by (Dollár, Belongie, and Perona 2010) to significantly increase the speed of the feature pyramid construction for non scale invariant features like HOG. The main intuition is that instead of downscaling the image and extracting features at each level of the pyramid, intermediary levels could be *approximated* using nearby feature maps.

Successive image down-sampling and calculating features at each pyramid level is computationally expensive. Taking advantage of the generally fractal structure of natural images, it is possible to obtain similar performance by only downsampling and computing the features for each halved image; *i.e.* at each octave (Dollár et al. 2014). In-between,

the features are approximated (upsampled and downsampled) from the ones directly calculated at each octave.

Regarding the context of application, object detection in the RoboCup Soccer competition (standard platform league) has been achieved with (generally manually tuned) detectors based on color segmentation (Khandelwal et al. 2010) or color histograms (Metzler, Nieuwenhuisen, and Behnke 2011), statistical modeling (Brandao, Veloso, and Costeira 2011), line detection, rough shapes (Budden et al. 2012), or simply as non-green patches which differentiate themselves from the green football field (Gudi et al. 2013).

In our experiments, we use the object detector provided by Dlib (King 2009) which uses the well known 31-dimensional feature extraction method described by (Felzenszwalb et al. 2010) (fHOG) together with Max-Margin Object Detection (MMOD) (King 2015) which improves training efficiency. Final detections are obtained by applying non-maximum suppression on the ensemble of overlapping detection boxes.

Image and Histogram Downsampling

The first important observation is that the main bottleneck in computing fHOG features resides in calculating the gradient histogram which happens before computing the final feature map. Most modern processors provide Single Instruction, Multiple Data (SIMD) instructions, which have the same execution time and electricity consumption as their scalar counterparts, but handle 16 bytes of data simultaneously which, for our purposes, enables 4 floating point operations instead of one. This is also the case for the Intel Atom 1.6GHz processor of the NAO robot used in our research. However, computing histograms involves the decision of which bin is associated with each data point, and therefore cannot be fully vectorized. This observation led us to consider avoiding successive histogram computations, similar to how other authors avoid direct feature extraction.

Another important aspect to consider is that the bulk of computational expense rests in the first few levels of the pyramid that use large scale images. For example, for a 10-level pyramid that uses $4/5$ downscaling factor on a 640x480 image, computing the final features for the first level costs $\sim 36.7\%$ of the entire pyramid computation time (see Figure 1).

We can identify three aspects of the fHOG feature extraction algorithm that can be accelerated:

- The level to which the scheme is applied
- Image downsampling strategy
- Histogram downsampling strategy

Here we describe each of these aspects, and then put them to the test in the following section, in comparison with related state of the art schemes.

All tests are performed on a dataset that contains images from the publicly available SPQR dataset (Albani et al. 2016) and also includes new frames coming from robots during test matches in different lighting conditions and of lower resolution. Lower resolution images were upscaled to 640x480 which is the chosen resolution of our evaluation,

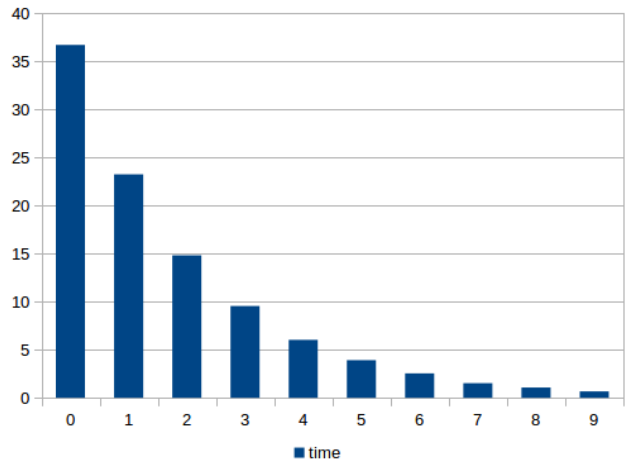

Figure 1: Average execution time (in percentage) at each level of a 10-level pyramid with $4/5$ downscaling factor on a 640x480 image.

as this is the real image size that is usable from the NAO robot camera; in fact, the output of the camera is 960p/30fps (1280x960) and is provided in YUYV format (also known as Y'UV422), but processing the full sized image exhausts much of the resources available on the robot (Genter et al. 2016). For training and testing, we only consider the Y value which can very efficiently be read directly from the raw camera output. The images in the enhanced dataset have been randomized and divided into 100 training images (50 with horizontal flip added) and 98 test images, amounting to 190 and 185 positive examples respectively. The reason for the relatively small training set is that the chosen algorithm seems to perform and generalize better with a smaller amount of training data.

Skipping Levels

As seen in Figure 1, it is most important to approximate lower levels of the pyramid, as they have the highest cost in terms of computation time. Throughout this work, we evaluate different downsampling strategies by applying them up to a certain level of the pyramid. After doing a hyperparameter search on our baseline fHOG object detector, we chose to use a $4/5$ downscale factor which in our implementation leads to a pyramid with 10 levels (0-9). In the following, we report the precision and execution time of each approach by applying the scheme up to a given level exclusively, while higher levels are computed in the same way as the baseline; this way, results at level 1 are equivalent for all schemes and baseline since the scheme is not applied to any level. Starting from level 2 and up to 9, the charts illustrate the effect of the chosen scheme on performance and speed. We also include level 10, which means that the entire pyramid (levels 0-9) is approximated, using the chosen scheme.

Image Downsampling

Constructing smaller scale images from which features are extracted is originally performed (baseline) at each pyramid level (dubbed *slow* method in the following). Related work proposes to only subsample images (dubbed *fast* method) at each octave (ratio of $1/2$) while approximating the intermediary levels. While significantly faster, one may argue that this method can lead to important information loss for higher levels (smaller image scales), due to the fact that entire pixels are ignored in the downsampled image.

We put this intuition to the test, and evaluate quality when skipping several levels when downsampling image. We note that using the *fast* approach, we obtain "pixelated" parts of the image, where gradient information is lost (see Figure 2), but this only becomes clear when the gap between downsampled images increases, in our case, further than 4 or 5 levels. We therefore expect that having at least some intermediary images, such as in the case of the power-law approach (Dollár et al. 2014) which subsamples images at each octave, should improve results.

Figure 2: Downsampled images at level 5 of the pyramid using *slow* (top left) and *fast* (top right) methods. Bottom row zooms in on the robot to highlight differences between *slow* (bottom right) and *fast* downsampling approaches. NAO's pixelated shoulder is clearly visible, showing gradient information loss in *fast* method.

To study the impact of the two image downsampling strategies, we measure speed and performance at each level in the pyramid. To visualize the progression in function of level, we plot the average execution time (Figure 3) obtained by a single image pyramid, measured for 10 configurations. Each configuration i consists in an fHOG detector that performs feature downscaling using the respective image downsampling strategy (*slow* / *fast*) until level $i-1$ and then, from level i onwards it performs the default feature extraction (which corresponds to downsampling the image and extracting features at each step).

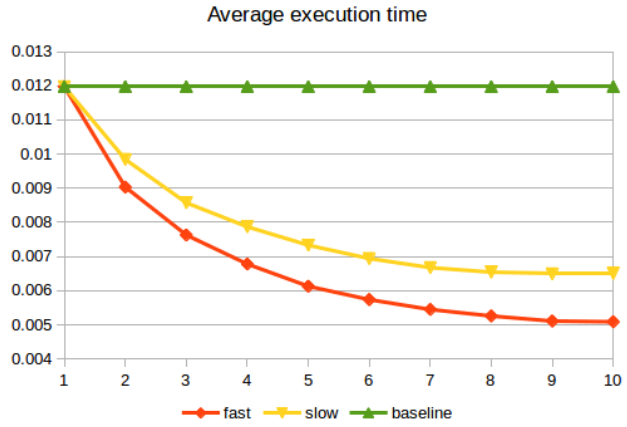

Figure 3: Average execution time (in seconds) on modern computer of *slow* vs. *fast* image downsampling when applying the scheme up to each level of the pyramid.

As expected, the *fast* approach is more desirable in terms of execution time. However, we find that the performance of extracted fHOG features depends on the quality of the image at higher levels, but remain robust to drastic downsampling at lower levels. Figure 4 shows that, with hyperparameter tuning for each configuration, *slow* downsampling outperforms the *fast* method overall. However, this also implies a significant loss in execution speed. Nevertheless, we note that for the first few levels, the performance difference is not as pronounced.

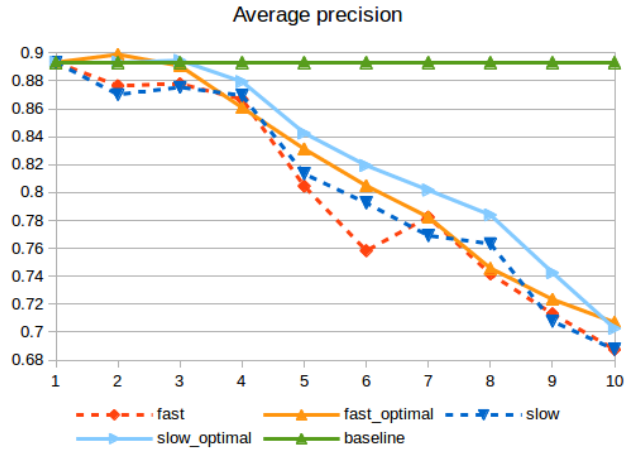

Figure 4: Average precision of *slow* vs. *fast* image downsampling when applying the scheme up to each level of the pyramid. Dotted lines show the performance of the detector in each case using the same hyperparameters initially found for the baseline. With hyperparameter search at each level, we obtain higher scores and a smoother transition.

Therefore, if the approximated gap between downsampled images is small enough, the *fast* downsampling strategy

should retain enough information to minimize performance loss while offering good execution speed gains.

Histogram Downsampling

In this work, we refer to "histogram" as the frequency of gradients binned into each of the 18 orientations described by (Felzenszwalb et al. 2010) that is computed *before* calculating the final 31-dimensional fHOG features, while (Dollár et al. 2014) describe feature downsampling on *final* features. Downsampling final features would seem much faster, because recomputing and normalizing them is directly avoided, however it turns out that the time lost on this process is regained in our approach because the downsampling is done on 18 dimensions instead of 31. This leads to very similar runtime for both approaches, but we observe higher performance loss in scaling final features. This loss could be alleviated by smoothing the feature maps as (Dollár et al. 2014) propose, but this would inevitably lead to slower runtime only to reach detection performance similar to our approach. In all experiments we use downsampling on 18-dimensional histograms and then compute and normalize the 31-dimensional fHOG features.

As with images, histograms can be downscaled using bilinear interpolation of bins between adjacent cells, either by always starting from the first level and obtaining the rest (which we dub *direct* method), or by successively obtaining level $i + 1$ from level i (*progressive* method). Because the algorithm requires histograms for all levels, the *progressive* method yields faster overall computation time as the source histogram is smaller, but leads to additional blur that, contrary to the case of images, decreases detection accuracy.

While blurred histograms may lead to performance loss, there is a chance for the machine learning phase to compensate by adapting to the blurred features, we note that learned models (vignettes) show some variation between the original algorithm and the two approximation methods (see Figure 5).

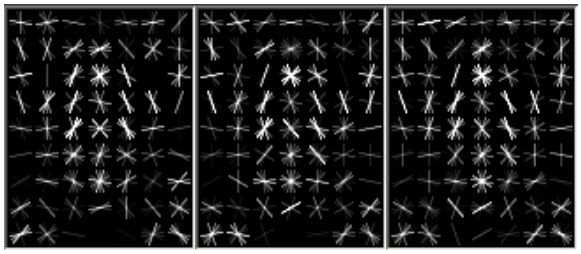

Figure 5: Learned robot detection model (vignette) by original algorithm (left), using *direct* (center) and *progressive* (right) methods. Characteristic differences are not noticeable, as the learned model is able to adapt to some of the blur or information loss in either version.

Figure 6 illustrates the difference between *direct* and *progressive* methods, where the blur introduced becomes visible. In the following we evaluate execution speed and performance of each method, by successively applying it up to a given level ranging from 2 to 10, where 10 is actually a

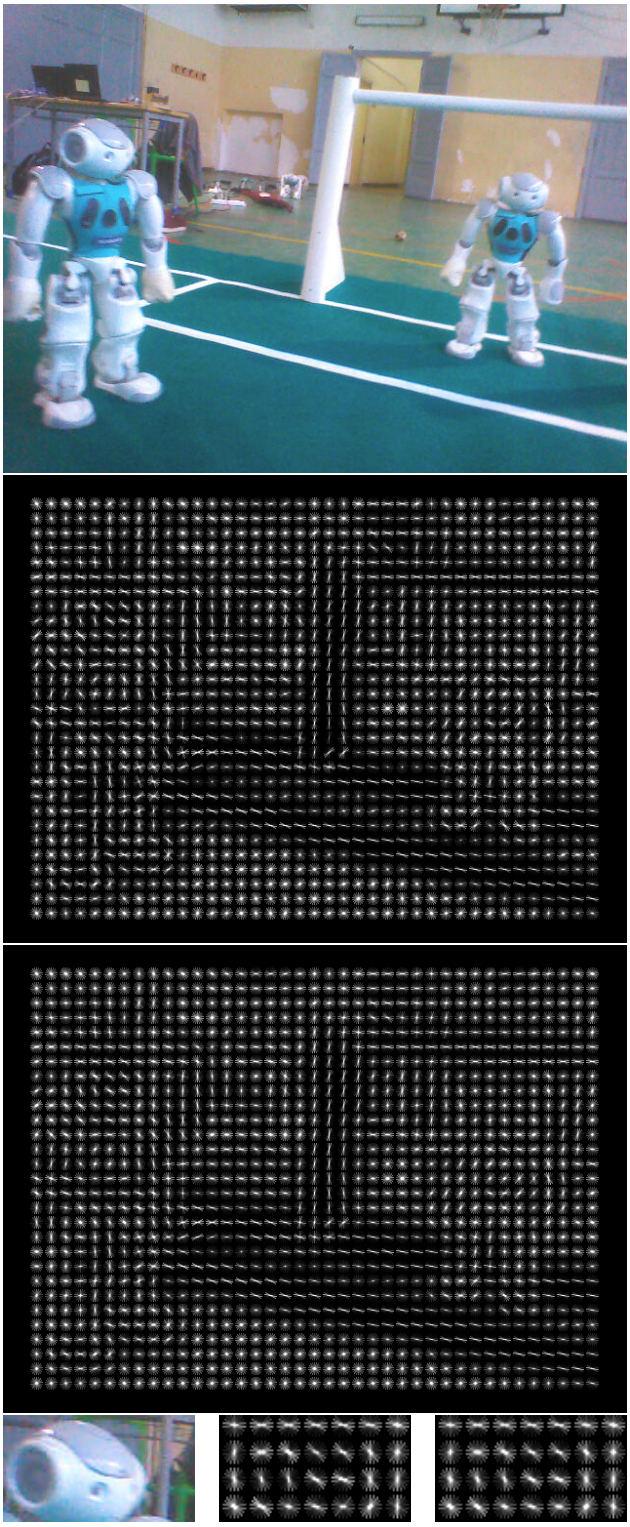

Figure 6: Example image from SPQR dataset (Albani et al. 2016) (top) with extracted fHOG feature maps at level 3 using *direct* (second row) and *progressive* (third row) methods. Bottom row emphasizes information loss in *progressive* method; NAO's ear and head contour are still visible in *direct* gradients but are blurred away in *progressive* method.

completely approximated pyramid, the entire pyramid has 10 levels (0-9) as in the previous results.

From Figure 7 we observe that the speed of both *progressive* and *direct* schemes are very similar, with little loss at higher levels for the *direct* histogram downsampling.

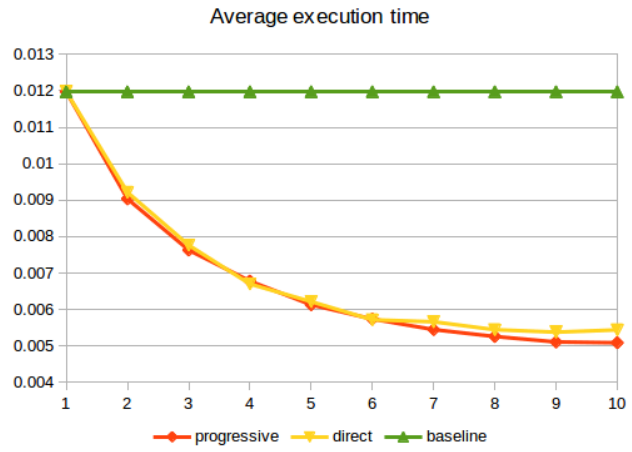

Figure 7: Average execution time (in seconds) on modern computer of *progressive* vs. *direct* histogram downsampling

As for the *slow* and *fast* schemes, we compute the average precision of the detector using *progressive* and *direct* histogram downsampling with hyperparameters of the baseline and with best scores after a parameter search for each level. Results in Figure 8 show that *direct* downsampling outperforms *progressive* by a small but real margin. We must note however that this advantage only appears after a few levels, where the blur introduced by the *progressive* method accumulates.

Adaptive Feature Pyramid Construction

It is clear that a trade-off exists between detection performance and the frame-rate at which the algorithm can run. While it is ideal to obtain accurate detections, in real setups such as the RoboCup competition the robot must also spend computational resources on other tasks, such as maintaining balance while walking. In fact, resource consumption varies throughout the game, depending on the situation. Therefore, it is desirable to have an adaptive control of the trade-off between accuracy and speed, while maximizing detection precision (*i.e.* minimizing false positives).

In the previous section, we evaluated the drop in performance that comes with "skipping" feature computations up to each level of the pyramid. Meanwhile, the power law approach (Dollár et al. 2014) provides a good trade-off: approximately 4% loss in average precision (in our setup, on images of robots) for almost doubling the speed of feature extraction.

Here we evaluate a hybrid[1] between the skipping approach described previously and the power law based

[1]Full code available on anonymous repository: https://github.com/blindpeerreviewgit/fhog

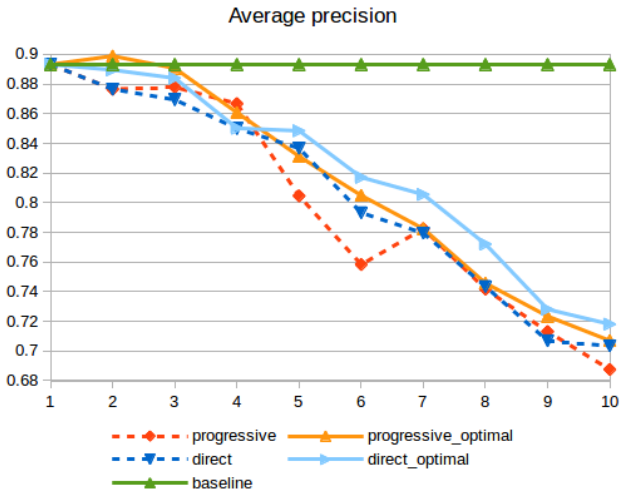

Figure 8: Average precision of *progressive* vs. *direct* histogram downsampling. Dotted lines show the performance of the detector in each case using the same hyperparameters initially found for the baseline.

method. We begin by skipping feature extraction up to level N exclusively, while retaining it at levels that coincide with a $1/2$ downscale of the image (octave). This way, we obtain a method that is bounded in speed and average precision by the original baseline (upper bound) and the power law based results (lower bound).

The setup presented herein uses $4/5$ downscale from one level to the next, therefore octaves correspond roughly to levels 3, 6 and 9. The proposed scheme skips levels excepting those corresponding to octaves and applying the scheme up to (but excluding) level N, thus we have equivalence between levels 3-4, 6-7 and 9-10 as the same conditions are met (see Algorithm 1).

Algorithm 1 Level approximation condition for $4/5$ image downsample ratio

```
nrLevels ← 10
octaveLevel ← 3
upToLevel ← level up to which scheme is applied
i ← 1
while i < nrLevels do
  if (i ≥ upToLevel) or (i mod octaveLevel = 0)
  then
    downsample image (slow or fast)
    compute histogram from image
  else
    downsample histogram (direct or progressive)
  end if
  extract features from histogram
  i ← i + 1
end while
```

We note that the approximation of levels following an oc-

tave is done using the result that was obtained from a downsampled image, therefore the quality of the histogram is superior to the case where the approximation had continued from the first level, as is the case in the previously described results.

In Figure 9 we compare our approach with the original algorithm, power law method and the previously described level skipping strategies. As with the optimal versions of previous strategies, we performed a hyperparameter search at each level of the proposed method.

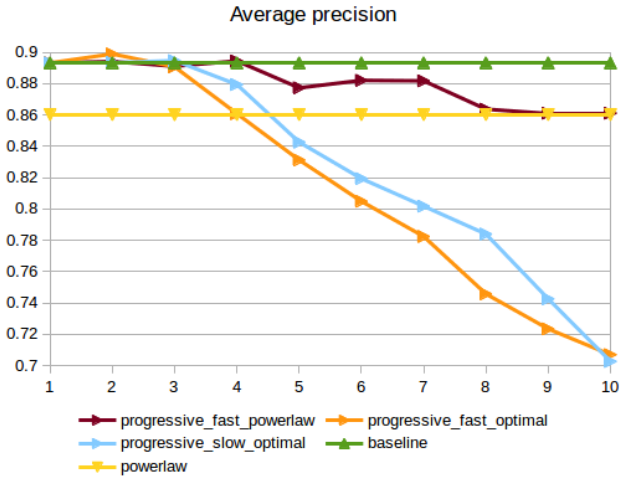

Figure 9: Average precision of studied feature approximation methods. Baseline and power-law based approaches (Dollár et al. 2014) shown as a straight lines, due to no level parameter.

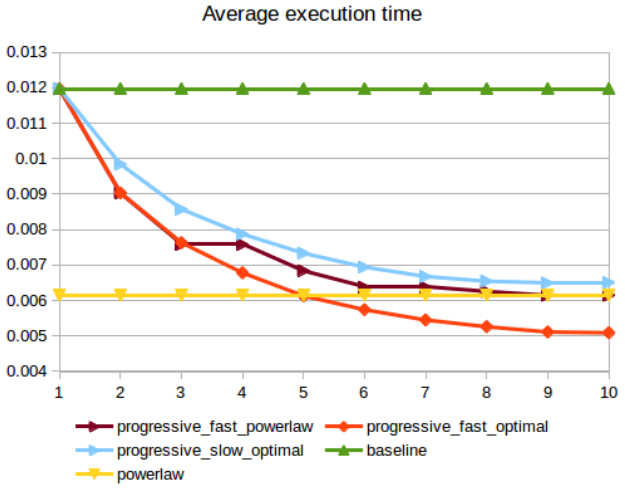

Figure 10: Average execution time (in seconds) of studied feature approximation methods on modern computer. Baseline and power-law based approaches (Dollár et al. 2014) shown as a straight lines, due to no level parameter.

We observe the importance of retaining image downsampling at octave intervals as described by (Dollár et al. 2014). Applying our scheme up to level 4 does not sacrifice average precision, even though the most computationally expensive levels are approximated. At higher levels, average precision gradually decreases until it matches its lower bound, the power law baseline.

As the previous experiments have shown, gains in execution speed are significant especially for the first few levels of the pyramid. Figure 10 illustrates how execution time drops with each level, on the modern computer. At level 4, which had no average precision loss, the scheme offers 1.57x speed increase relative to the original algorithm. Increasing the level up to which the scheme is applied to 7 gives a 1.87x speed increase with only $\sim 1\%$ decrease in average precision.

We note that the *slow* image downsampling strategy could give slightly higher average precision results, but the loss in execution speed would be much higher.

Results

The processor equipped on the NAO v4 robot platform is, according to our estimates, approximately 25 times slower than the modern CPU on which we ran the evaluation. This is due to several factors such as lower frequency, less processor cache and other aspects which are outside the scope of this paper. These differences impose a hard standard on what algorithms can be run on this model of robots.

We evaluated the scheme on the NAO robot, obtaining gains in execution speed similar to the PC version (see Table 1). In fact, the speed increase is 1.68x without average precision loss compared to the original baseline, which is higher than the PC version, due to optimizations that are not available on the robot. At level 7, we obtain 1.95x speed increase with only 1.14% loss in average precision, while at level 10 (which is equivalent to the power law approach) the speed increase is 2.05x but the loss rises to 3.21%.

level	AP	TMC	TR	FPPI	MR
1	89.3%	11.9	298.2	5.4 x 10^{-2}	11.3%
2	89.4%	9.0	227.1	8.6 x 10^{-2}	8.6%
3-4	89.4%	7.5	177.0	3.7 x 10^{-2}	10.2%
5	87.7%	6.8	162.3	2.1 x 10^{-2}	11.8%
6-7	88.2%	6.3	152.4	3.2 x 10^{-2}	11.3%
8	87.0%	6.2	148.9	4.3 x 10^{-2}	12.9%
9-10	86.1%	6.1	145.3	3.7 x 10^{-2}	13.5%

Table 1: Summary of proposed scheme performance. Average precision (AP), feature extraction execution time in milliseconds on modern computer (TMC) and on robot (TR), false positives per image (FPPI) and miss rate (MR) are shown for each level up to which the scheme is applied.

We note that the time needed to compute the feature pyramid on the robot is still elevated, and thus more optimizations will be required. However, the ~ 150 millisecond drop with minimal loss in average precision is an important improvement in this case. To retain smooth motion and cogni-

tion, the algorithm can be broken down into multiple steps, and tracking can be performed in between. The important aspect is that the number of false positives per image is low, while some such cases are actually correct hits which were not annotated in the dataset (see Figure 11).

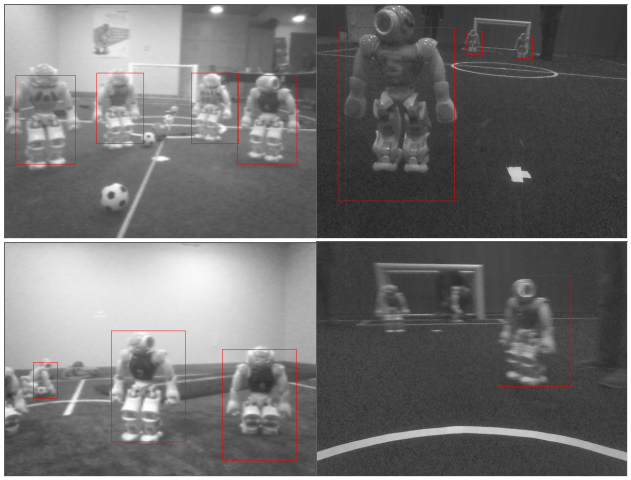

Figure 11: Robot detection examples. Top row shows true positives, bottom left shows a false positive (fallen robot was not annotated in the dataset) and bottom right shows false negatives due to excessive blur and similar background.

The algorithm is relatively robust to lighting conditions, as well as to a reasonable amout of motion blur. We notice that a limitation is represented by situations in which the robot "blends" with the background, as illustrated in the bottom-right sample in Figure 11.

Outside the scope of the RoboCup competition, we expect that our approach will offer a better, adjustable trade-off between average precision and execution speed. For more difficult object detection problems, if hardware resources are more readily available, finer sampled image pyramids may improve results, and the gain from approximating intermediate levels becomes much more pronounced.

Conclusions and Future Work

In this work we provided a detailed evaluation of the trade-off between feature extraction speed and detector average precision, at each level of the feature pyramid. In our experiments, we used histogram downsampling instead of final feature downsampling used in related work. Results showed that this trade-off is not linear and that average precision is not lost by skipping the first few levels of the pyramid, which in fact account for a major part of the total computation time. We compared these results with the initial algorithm and a state of the art method based on image downsampling power law as a baselines. Based on this analysis, we developed a hybrid method which is upper bounded by the original baseline and lower bounded by the power law approach in both execution time and average precision. In practice, the proposed method can be adapted, by changing the level up to which it is applied, to favor average precision or execution speed. This way, on a modern computer, we obtain 1.57x increase in pyramid construction speed without any loss in average precision, $\sim 1\%$ average precision loss ($> 60\%$ improvement compared to the state of the art method) with 1.87x speed increase ($\sim 96.4\%$ of the possible execution speed gain on modern computer), and finally the same results as the state of the art when our method approaches its lower bound. Further gain in execution speed may also be obtained by skipping levels at octave intervals, but at the cost of drastic performance loss.

Execution speed gains are retained on the robot implementation, where we obtain 1.68x speed increase compared to the baseline with no loss and 1.95x increase with $\sim 1\%$ average precision loss, compared with 2.05x obtained with the power law baseline that presents $\sim 3\%$ average precision loss.

Following from the observation that the first few levels of the pyramid account for the majority of execution time, and that in our approach we compute the first level (level 0), extra time should be saved by upscaling level 0 from higher levels. Future work will include performance evaluation of this idea, as well as vectorizing histogram downsampling to the extent possible. The actual robot implementation will divide the feature pyramid extraction and object detection algorithm into steps that can be executed over multiple cognition cycles to offer more control over processor load.

Acknowledgements

[removed for blind review]

References

Albani, D.; Youssef, A.; Suriani, V.; Nardi, D.; and Bloisi, D. 2016. A deep learning approach for object recognition with nao soccer robots. In *Robocup Symposium*.

Brandao, S.; Veloso, M.; and Costeira, J. P. 2011. Fast object detection by regression in robot soccer. In *Robot Soccer World Cup*, 550–561. Springer.

Budden, D.; Fenn, S.; Walker, J.; and Mendes, A. 2012. A novel approach to ball detection for humanoid robot soccer. In *Australasian Joint Conference on Artificial Intelligence*, 827–838. Springer.

Dalal, N., and Triggs, B. 2005. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, 886–893. IEEE.

Dollár, P.; Belongie, S. J.; and Perona, P. 2010. The fastest pedestrian detector in the west. In *British Machine Vision Conference*, volume 2, 7.

Dollár, P.; Appel, R.; Belongie, S.; and Perona, P. 2014. Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36(8):1532–1545.

Felzenszwalb, P. F.; Girshick, R. B.; McAllester, D.; and Ramanan, D. 2010. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence* 32(9):1627–1645.

Genter, K.; MacAlpine, P.; Menashe, J.; Hannah, J.; Liebman, E.; Narvekar, S.; Zhang, R.; and Stone, P. 2016. Ut austin villa: Project-driven research in ai and robotics. *IEEE Intelligent Systems* 31(2):94–101.

Gudi, A.; de Kok, P.; Methenitis, G. K.; and Steenbergen, N. 2013. Feature detection and localization for the robocup soccer spl. *Project report, Universiteit van Amsterdam (February 2013)*.

Khandelwal, P.; Hausknecht, M.; Lee, J.; Tian, A.; and Stone, P. 2010. Vision calibration and processing on a humanoid soccer robot. In *The Fifth Workshop on Humanoid Soccer Robots at Humanoids 2010*.

King, D. E. 2009. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research* 10(Jul):1755–1758.

King, D. E. 2015. Max-margin object detection. *arXiv preprint arXiv:1502.00046*.

Metzler, S.; Nieuwenhuisen, M.; and Behnke, S. 2011. Learning visual obstacle detection using color histogram features. In *Robot Soccer World Cup*, 149–161. Springer.