Real-Time Pedestrian Detection With Deep Networks Cascades

Anelia Angelova[1]
anelia@google.com
Alex Krizhevsky[1]
akrizhevsky@google.com
Vincent Vanhoucke[1]
vanhoucke@google.com
Abhijit Ogale[2]
ogale@google.com
Dave Ferguson[2]
daveferguson@google.com

[1] Google Research
1600 Amphitheatre Parkway
Mountain View, CA, USA

[2] Google X
1600 Amphitheatre Parkway
Mountain View, CA, USA

Abstract

We present a new real-time approach to object detection that exploits the efficiency of cascade classifiers with the accuracy of deep neural networks.

Deep networks have been shown to excel at classification tasks, and their ability to operate on raw pixel input without the need to design special features is very appealing. However, deep nets are notoriously slow at inference time.

In this paper, we propose an approach that cascades deep nets and fast features, that is both extremely fast and extremely accurate. We apply it to the challenging task of pedestrian detection. Our algorithm runs in real-time at 15 frames per second. The resulting approach achieves a 26.2% average miss rate on the Caltech Pedestrian detection benchmark, which is competitive with the very best reported results. It is the first work we are aware of that achieves extremely high accuracy while running in real-time.

1 Introduction

Pedestrian detection has been an important problem for decades, given its relevance to a number of applications in robotics, including driver assistance systems [8], road scene understanding [16] or surveillance systems. The two main practical requirements for fielding such systems are very high accuracy and real-time speed: we need pedestrian detectors that are accurate enough to be relied on and are fast enough to run on systems with limited compute power. This paper addresses both of these requirements by combining very accurate deep-learning-based classifiers within very efficient cascade classifier frameworks [38].

Pedestrian detection methods have employed a variety of techniques and features [4, 5, 11, 12, 33, 38, 39, 42]. Some have focused on increasing the speed of detection [4, 5, 38], whereas others have focused on accuracy [25, 31, 42]. Recently a novel range of methods have emerged, based on Deep Neural Networks, showing impressive accuracy gains [24].

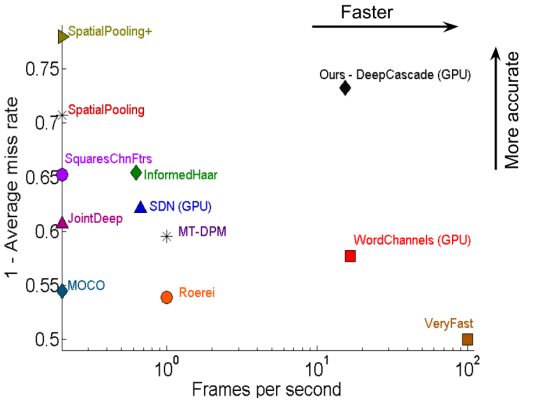

Figure 1: Performance of pedestrian detection methods on the accuracy vs speed axis. Our DeepCascade method achieves both smaller miss-rates and real-time speeds. Methods for which the runtime is more than 5 seconds per image, or is unknown, are plotted on the left hand side. The SpatialPooling+ method uses additional motion information.

However, Deep Neural Network (DNN) models are known to be very slow [19, 21, 23], especially when used as sliding-window classifiers. We propose a very competitive DNN approach for pedestrian detection that is both very accurate and runs in real-time. To achieve this, we combine a fast cascade [4] with a cascade of DNNs [24]. Compared to existing approaches, ours is both very accurate and very fast, running in real-time at 67 milliseconds on GPU per image, or 15 frames per second (FPS). To our knowledge, no competing method exists that achieves both very high accuracy and real-time performance. Figure 1 visualizes existing methods as plotted on the accuracy - computational time axis, measured on the challenging Caltech pedestrian detection benchmark [12]. As can be seen in this figure, our approach is the only one to reside in the high accuracy, high speed region of space, which makes it particularly appealing for practical applications.

We are not the first to consider the use of a DNN in a cascade. Ouyang et al. and Luo et al. [25, 29] both apply deep networks in combination with fast feature cascades. These approaches are conceptually similar to ours, but have not been able to capture the full speed benefit of cascades, with processing times still too slow for real-time performance, e.g. 1-1.5 seconds per image on GPU [25]. Conversely, we here take advantage of the very fast features for elimination, VeryFast [4], and of small and large deep networks [3, 24]. Our proposed approach is unique, as it the first one to produce a pedestrian detector at real-time speeds (15 FPS) that is also very accurate.

In agreement with previous DNN research, our method shows that, applying DNNs to raw image values provides excellent results. It is also advantageous to achieve real-time performance. This is in contrast to common DNN-based approaches [25, 29], who apply DNNs only to processed, edge-like features rather than raw pixel input, which is a disadvantage in terms of speed. Further, we show that cascades incorporating DNNs are very good at generalization and transfer learning, with strong performance on the Caltech pedestrian dataset from a system that used no Caltech data during training.

Getting real-time solutions for pedestrian detection has been hard. Recently proposed WordChannel features [9] provide a real-time solution on the GPU (16 FPS), but at a notable

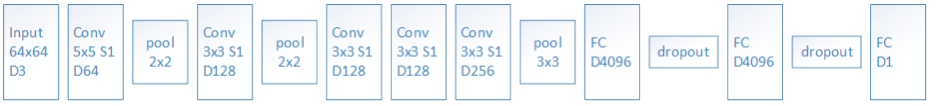

Figure 2: The architecture of the baseline deep network for pedestrian detection. S denotes the step size, whereas D refers to the number of convolutional kernels or units per layer.

loss in average miss rate (42%). The seminal VeryFast method [4] runs at 100 FPS but with even further loss in miss rate. In the related domain of general object detection, which utilize similar capacity DNNs as ours, and also take advantage of evaluating fewer candidates per image, the most accurate method is of Girshick et al. [19], which takes 53 seconds per frame on CPU and 13 seconds per frame on GPU, which is 195 times slower.

We further note that our approach is easy to implement, as it is based on open source code. More specifically, we use the 'Doppia' open source implementation provided by Benenson and collaborators [2] of the VeryFast algorithm [4]. Our deep neural networks are implemented in the open source cuda-convnet2 code [1, 24].

The main contribution of this work is a pedestrian detection system that is both more accurate than previous works and runs in real-time. As such, it can be practically deployed within a real-life pedestrian detection system. No other prior work has demonstrated such capabilities. We expect our work to impact future methods by providing a simple to implement, accurate and effective real-time solution. Thus, future methods can continue to further push the boundaries in accuracy in pedestrian detection, while simultaneously keeping the methods fast and practically relevant.

2 Previous work

Pedestrian detection has been a central topic in computer vision research, spanning more than 20 years of research [6, 28, 32, 36]. A wide variety of methods have been applied to pedestrian detection over the years, with continued improvement in performance [4, 5, 6, 7, 9, 10, 12, 13, 33, 38, 39, 42]. Some methods focus on improving the base features used [6, 9, 10, 42], whereas others focus on the learning algorithms [13, 25], or other techniques such as incorporating Deformable Parts Models [33, 40] or using context [13, 29, 40, 41]. Dollar et al. [12, 14] developed a benchmark and evaluation toolbox that has been instrumental in tracking progress in the field. Benenson et al. [6] have recently proposed a comparative paper that evaluates performance of various features and methods on pedestrian detection.

Viola and Jones proposed a cascade-of-classifiers approach [38], which has been widely used for real-time applications. The method has been extended by employing different types of features and techniques [13, 26, 27], but fundamentally the concept of the cascade, with early rejection of majority of test examples, has been widely utilized to achieve real-time performance. Perhaps the most popular feature used for pedestrian detection (and several other image-based detection tasks) is the HOG feature developed by Dalal and Triggs [10]. Although not real-time, about 1 FPS, this work has been instrumental to the development of faster and more accurate features for pedestrian detection, which are used in the top performing methods in combination with SVM or Decision forests [5, 12, 26]. Deformable Parts Models [17] have shown success on the pedestrian detection task [33, 40]. Deep learning-based techniques have also been applied to pedestrian detection and have led to improve-

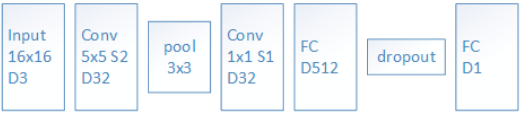

Figure 3: The architecture of the tiny deep network for pedestrian detection, which is a part of the DNN cascade.

ments in accuracy [3, 25, 29, 30, 37]. These approaches are still slow, ranging from over a second per image [25] to several minutes [37]. The faster approaches do not apply deep nets to the raw pixel input so their accuracy is reduced.

Improving the speed of pedestrian detection has also been an active area. Benenson et al. proposed a method reaching speeds of 100 to 135 FPS [4] for detection in a 480x640 image, albeit with significantly lower accuracy. Other researchers have focused specifically on speeding up Deep Neural Networks [19, 21, 23], but with no real-time solutions yet.

3 Deep Networks Cascades

This section describes our main architecture and our approach to build very fast cascades with deep networks. By far the most accurate detector is based on deep networks [24]. However it is extremely slow when applied to 100,000 candidate windows. To complement it, we use a tiny deep network from our prior work [3] that can make the classifier work at 2 FPS, thus speeding it up by 80x. Finally, we insert a fast feature cascade [4] with which our method achieves a real-time 15 FPS solution (Section 3.1).

The architecture of the baseline deep neural network is based on the original deep network of Krizhevsky et al. [24], which has been widely adopted and used by many researchers. However it is extremely slow when ran in a sliding window fashion. One key difference here is that we reduced the depths of some of the convolutional layers and the sizes of the receptive fields, which is specifically done to gain speed advantage. Our baseline architecture is shown in Figure 2.

Even with the proposed speedsups, this network is still slow and not appropriate for real-time applications. To speed it up we utilize the idea of a tiny convolutional network from our prior work [3]. The tiny DNN classifier features only three hidden layers: a 5x5 convolution, a 1x1 convolution, and a very shallow fully-connected layer of 512 units; it is so designed for speed. When ran in a cascade, it will process all image patches first, and pass through only the patches that have high confidence values. The architecture of the tiny network is shown in Figure 3. It reduces the massive computational time that is needed for a sliding window detector to evaluate a full DNN at all candidate locations and scales. The combination of these two deep networks works 80 times faster than the original sliding window. While not true real-time, this simple method obviously saves a large amount of computations and is a viable solution for offline processing, as well. This speedup is also a crucial component in achieving real-time performance in our fast cascade method, described below.

3.1 Fast Deep Networks Cascade

Deep network approaches, are very accurate, as seen later in our experiments, but also slow. Benenson et al. [4] introduced a cascade-based approach that can achieve an impressive 100

or 135 frames per second. We propose here a hybrid approach that will combine the best of both approaches. More specifically, we employ the soft-cascade from Benenson et al. [4] to evaluate all patches in an image and reject the easy ones. The VeryFast cascade quickly starts to reduce recall with each stage, and, although fast, produces a high average miss rate in the end [6]. Our goal is to be able to eliminate many patches and at the same time keep the recall high. For that purpose we used only 10% of the stages in that cascade. Specifically, we use a cascade of only 200 stages, instead of the 2000 used in the original work.

We note that our network is trained using the publicly available 'cuda-convnet2' code provided by Krizhevsky [1] and fast cascade follows the GPU implementation provided by Benenson et al. [2, 4], making our approach easily accessible for re-implementation and use.

3.2 Runtime

We measure the runtime on a standard NVIDIA K20 Tesla GPU. The soft-cascade takes about 7 milliseconds (ms). About 1400 patches are passed through per image from the fast cascade. The first stage of the cascade runs at 0.67 ms per batch of 128, so it can process the patches in 7.3 ms. The second stage of the cascade (which is the baseline classifier) takes about 53ms. The overall runtime is about 67ms per 640x480 image, which is 15 frames per second. Previous methods, e.g. Luo et al. [25] have similarly employed a hybrid approach, HOG-based cascade and a deep network at the bottom of the cascade, but their runtime is about 1-1.5 seconds also on GPU, which is 22 times slower.

3.3 Implementation details

Pretraining. We make use of pre-training, that is, the weights are initialized from the weights of a network that has been trained on Imagenet. This is standard practice for deep networks, as the number of parameters is much larger than the available data for training. We compared experimentally to a network trained without pre-training, and observed a positive effect on the accuracy. Other works have noted similar effects, and since pretraining is easy to incorporate, it is a preferred choice in our work and others [19].

Data generation. A standard procedure for data generation is used, in which we crop a square box around pedestrian examples. At the time of data generation, the cropped square images are resized to 72x72. Additional random crops of size 64x64, to match the input size of the network, are taken at each iteration during training of the DNN. This is a standard data augmentation technique for training convolutional networks and allows more diverse 'views' of the training examples. The tiny network follows the same procedure, with an additional resizing of the input. We further collect hard negatives, which is important because the initial generated dataset is sampled uniformly from the available examples, and contains a large fraction of easy examples. Additionally, we eliminated the pedestrian examples that are smaller than 10 pixels in width, as these examples are indistinguishable when seen as individual patch and are not useful for methods that do not apply motion (as is ours).

4 Experimental evaluation

4.1 Datasets

We evaluated our results on the now standard Caltech Pedestrian detection dataset [12], as this dataset serves as a standard benchmark for pedestrian detection methods. We further consider additional pedestrian datasets. Details are below.

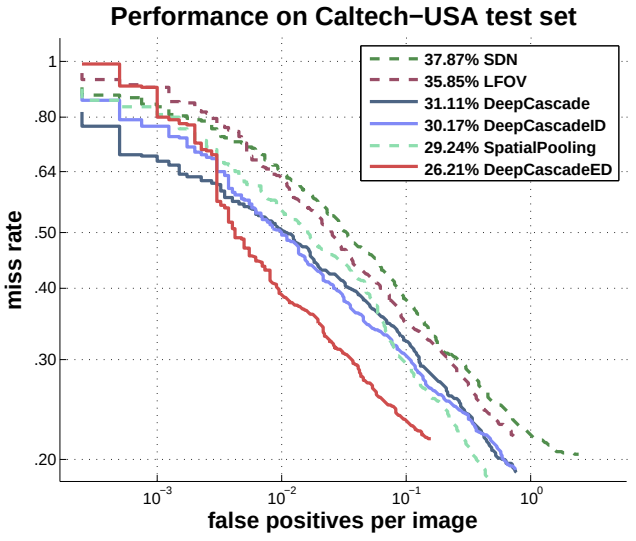

Figure 4: Results of our DeepCascade methods on the Caltech test data for pedestrians. The best state-of-the-art method (SpatialPooling) and the best state-of-the-art Deep Networks methods SDN, LFOV are shown for comparison. All other methods are shown in Table 1. Our network performs much better in the area of the curve with small number of false positives, which is the target area for a practical pedestrian detection system.

Caltech dataset. The Caltech dataset contains about 50,000 labeled pedestrians. The dataset is collected from a dashboard color camera and contains suburban and city scenes. In our experiments we used about two thirds of the available pedestrians as some of them are of very small sizes and poor quality. We further generated about 4M negative examples.

Independent pedestrian dataset. We independently collected pedestrian dataset which is comparable in size to the Caltech one. It contains higher quality images of pedestrians due to the better resolution of the cameras used. We wanted to measure performance when not training on the Caltech data, but on an independent dataset of similar size.

Extra pedestrian dataset. We complemented the Caltech training set with additional examples to further experiment with the effects of adding more data. This dataset consists of the Caltech training dataset as above, plus the publicly available ETH [16] and Daimler [15] pedestrian datasets. The Daimler dataset [15] contains only grayscale images and the ETH dataset [16] is collected from a mobile platform, in this case a stroller.

4.2 Evaluation of the Fast Deep Network Cascade

In our evaluation, we use the training and test protocols established in the Caltech pedestrian benchmark and report the results by measuring the average miss rate as prior methods did. We use the code provided in the toolbox of this benchmark [12] to do the evaluation.

We first evaluate the performance of the Fast Deep Networks Cascade as described in Section 3.1. Our results are summarized in Table 1, where we list the current state-of-the-art pedestrian detection methods. Figure 4 shows the performances of our DeepCascade algorithm, as compared to the state-of-the-art methods. We tested on pedestrians of at least 50

Dataset	Avg. miss rate	FPS
Roerei [5]	46.13	1
MOCO[7]	45.5	
WordChannels [9]	42.3	16
JointDeep[29]	39.3	
MT-DPM+Context [40]	37.64	
ACF+SDt [34] (w. motion)	37.3	
SDN [25]	37.8	0.67
LFOV [3]	35.8	3.6
InformedHaar [42]	34.6	0.63
Hosang et al. [22]	32.4	
SpatialPooling [31]	29.0	0.13
Hosang et al. (w. extra data) [22]	23.3	
SpatialPooling+/Katamari (w. motion): [6, 31]	22.0	
DeepCascade	31.11	15
DeepCascadeID on indep. data	30.17	15
DeepCascadeED w/ extra data	26.21	15

Table 1: Summary of results of the DeepCascade methods compared to the state-of-the-art methods. Our method is the only one that has a low average miss rate and runs in real-time, at 15 frames per second (FPS).

pixels ('reasonable set'). Our fast deep networks cascade, trained on Caltech is denoted as DeepCascade. We denote DeepCascadeID and DeepCascadeED, the cascade when trained on independent dataset and when trained on the extra dataset. Compared to the state-of-the-art, we can see that our methods with average miss rates of 31.11%, 30.17%, and 26.21%, outperform most approaches, including all deep learning-based ones. The only exception are the approaches that use additional motion features SpatialPooling+ and Katamari [6, 31] which both perform at 22%. A recent method of Hosang et al. [22] reported very competitive methods ranging from 32.4% to 23.3% for various scenarios, including pre-training and extra data (results available after preparation of our manuscript). This method directly applies DNNs in a sliding window fashion which is very slow in practice. The DeepCascade approach, is outperformed by the SpatialPooling method that directly optimizes the area under the curve and by the recent method of [22]. Thus the SpatialPooling method performs better than most other methods for very high false positive values, and this part of the curve is not useful for practical applications. For example, for driving assistance applications, the goal is to reduce the false alarms as much as possible because they may cause unexpected or dangerous behavior of the vehicle.

When looking at the performance of our DeepCascadeID method, which is not trained on Caltech, we can see that it is competitive with the best methods that are trained on Caltech data. When comparing to prior methods that use an independent dataset only, we can see the performance of our deep-net based system is even more impressive. For example, when training on the KITTI pedestrian dataset [18], the best known average miss rate is 61.2%, whereas when training on INRIA [10], the average miss rate is 50.2% [6]. Both miss rates are much higher than 31.1% of our method. Our results with extra data (DeepCascadeED) outperform the state-of-the-art methods on this benchmark and achieves a 26.2% average miss rate. This points to the strengths of DNNs, namely that more data can eliminate engineering

Dataset	Average miss rate
DNN-only cascade of DNNs	35.90
DNN-only cascade, indep data	32.52

Table 2: Average miss rate comparison for the DNN-only cascade of DNNs. The DNN-only cascade is run with a less dense sampling.

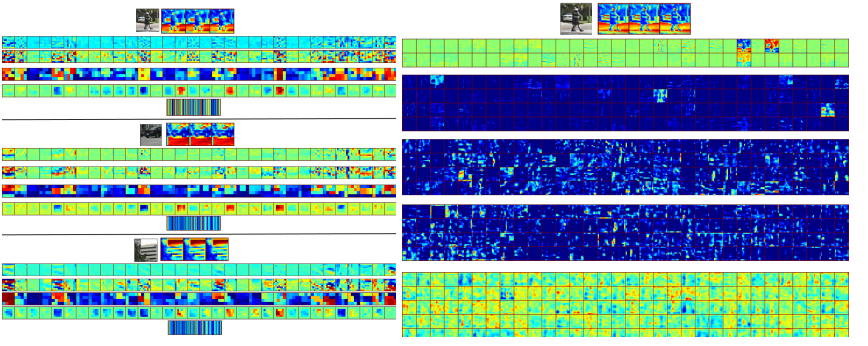

Figure 5: Left: Activations for the tiny deep neural network for different input examples. Each panel shows the convolutional layer, the normalization and pooling, then the 1x1 convolutional one and finally the fully-connected one. Right: Activations for the baseline deep neural network for an input example. Only the convolutional layers are shown.

of special features. More specifically, we see that more and high quality data can give similar or better results as do well engineered features, and obtain outstanding performance.

Apart from achieving very good accuracy, our method is much faster and runs at 15 frames per second, which is real-time performance. Other real-time algorithms, we are aware of, VeryFast [4] at 100 FPS and WordChannels [9] at 16 FPS (on GPU) have high average miss rate of 50% and 42%, respectively. Figure 7 visualizes example detections on the Caltech test data of the fast cascade DNN method when trained on Caltech data (top) and when trained on Independent data (bottom). As we can see, although, the networks are trained on non-overlapping data, they perform quite similarly qualitatively, as well.

4.3 Evaluation of the deep net-only cascade

We investigate in this section the performance of the cascade that involves only the two deep networks. Table 2 shows the performance of the DNN-only cascade. We also include results when trained on the Independent data and see that it has very good performance, similar to the fast cascade. We note that the results of both DNN-only cascades are also better than all prior methods with the exception of SpatialPooling [31] and the motion feature-based ones [6]. The results of this cascade are slightly worse in terms of average miss rate, than the fast cascade, because a coarser sampling is applied here. We chose to use a coarser sampling to optimize for speed. It is clear, though that more dense sampling can be reasonably expected to produce results comparable or better than the DeepCascade. To gain more understanding of the behavior of the system, we visualize the activations of each of the stages of the DNN-only cascade, respectively the first tiny network and the final stage, Figure 5.

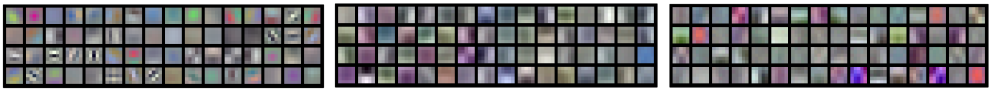

Figure 6: The filters learned by the first convolutional layer when training on the Caltech training set with pre-training (left) and on the same dataset without pre-trainig (middle); training on the extra pedestrian dataset without pre-training produces the filters at the right. The non-pretrained filters resemble parts of pedestrians, but the pre-trained filters seem to be better at distinguishing what is not a pedestrian and better handle hard negatives.

4.4 The effects of pre-training

Since starting from a pre-trained model on the ImageNet dataset is very convenient, as it allows the method to converge faster and/or use less data to achieve good results, we investigated here what effect pre-training has for this particular task. In summary, we find that pre-training is helpful by a small amount. We have observed a very similar trend in several other unrelated image recognition and detection problems [20, 35].

When we compared the pretrained and the non-pretrained models, the model that is trained without pre-training in the fast cascade obtained a larger average miss rate of by 3.27%. In terms of precision and recall, we observed that its precision is much poorer, for similar recall values. Figure 6 visualizes the filters that have been learned in the first convolutional layer with and without pre-training. As seen, the filters that have been trained without any pre-training are actually resembling of low level filters that are common in the dataset for pedestrian detection. Conversely, the filters learned with pre-training are more general and do not change significantly from their initialization. However, as noted above, pre-training does seem to help in the final performance. Our observation is that the pre-trained model is more successful in eliminating false alarms, which may indicate that although the filters without pre-training are better at detecting pedestrians, the filters trained on 1000 diverse object categories are better to discriminate and eliminate what is not a pedestrian. We also included the learned filters when training on the larger extra data. Comparing to the one trained on Caltech only, the former filters seem to be more expressive, which can also result from having more diverse training data. We naturally observed that the models trained on extra data perform better.

Discussion. While using additional data makes the method not strictly comparable to others, we observe notable improvements, which have been obtained with no special tuning or changes in the infrastructure. Additionally, we observed that using independent, similar size, but better quality data, can achieve better results, even when possibly paying a penalty for mismatch between the training and test data. These findings can be useful for future works which may want to focus on getting more and diverse high quality data.

5 Conclusion and future work

We have presented a Deep Neural Network-based algorithm for pedestrian detection which combines the ideas of fast cascade and a deep network. It is simple to implement as it is based on open source implementations. Our method ranks among the best ones for pedestrian detection and runs in real-time, at 15 frames per second. This is the only method we are aware of that is both real-time and achieves high accuracy.

Figure 7: Example detections of the Fast Cascade of DNNs on the Caltech test data. The top row corresponds to detections with a model trained on Caltech data, the bottom row is for a model trained on Independent data. Interestingly, the two models often make similar mistakes, although they are trained on non-overlapping datasets (see the right-most column).

We hope this method will help future works to continue to improve pedestrian detectors in terms of both accuracy and speed so that more methods can be usable for pedestrian detection for real-time applications. Future work can include increasing the depth of the deep networks cascade by adding more tiny deep networks and exploring the efficiency-accuracy trade-offs. We further plan to explore using motion information from the images, because clearly motion cues are extremely important for such applications. One challenge is to be able to do this at sufficiently high computational speeds.

References

[1] https://code.google.com/p/cuda-convnet/ A. Krizhevsky, Cudaconvnet.

[2] https://bitbucket.org/rodrigob/doppia Doppia: open source implementation of pedestrian detection algrithms.

[3] A. Angelova, A. Krizhevsky, and V. Vanhoucke. Pedestrian detection with a large-field-of-view deep network. *ICRA*, 2015.

[4] R. Benenson, M. Matthias, R. Tomofte, and L. Van Gool. Pedestrian detection at 100 frames per second. *CVPR*, 2012.

[5] R. Benenson, M. Matthias, T. Tuytrlaars, and L. Van Gool. Seeking the strongest rigid detector. *CVPR*, 2013.

[6] R. Benenson, M. Omran, J. Hosang, and B. Schiele. Ten years of pedestrian detection, what have we learned? *2nd Workshop on Road scene understanding and Autonomous driving, ECCV*, 2014.

[7] G. Chen, Y. Ding, J. Xiao, and T. Han. Detection evolution with multi-order contextual co-occurrence. *CVPR*, 2013.

[8] E. Coelingh, A. Eidehall, and M. Bengtsson. Collision warning with full auto brake and pedestrian detection - a practical example of automatic emergency braking. *13th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2010.

[9] A. Costea and S. Nedevschi. Word channel based multiscale pedestrian detection without image resizing and using only one classifier. *CVPR14*, 2014.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *CVPR*, 2005.

[11] Y. Ding and J. Xiao. Contextual boost for pedestrian detection. *CVPR*, 2012.

[12] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. *CVPR*, 2009.

[13] P. Dollar, R. Appel, and P Perona. Crosstalk cascades for frame-rate pedestrian detector. *ECCV*, 2010.

[14] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *PAMI*, 2012.

[15] M. Enzweiler and D. M. Gavrila. Monocular pedestrian detection: Survey and experiments. *IEEE Trans. on Pattern Analysis and Machine Intelligence (T-PAMI)*, 2009.

[16] A. Ess, B. Leibe, K. Schindler, , and L. van Gool. A mobile vision system for robust multi-person tracking. *CVPR*, 2008.

[17] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *PAMI*, 2010.

[18] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. *CVPR*, 2012.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *http://arxiv.org/pdf/1311.2524v4.pdf*, 2013.

[20] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR*, 2014.

[21] A. Giusti, D. Ciresan, J. Masci, L. Gambardella, and J. Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks. *ICIP*, 2013.

[22] J. Hosang, M. Omran, R. Benenson, and B. Schiele. Taking a deeper look at pedestrians. *CVPR*, 2015.

[23] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv technical report*, 2014.

[24] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[25] P. Luo, X. Zeng, X. Wang, and X. Tang. Switchable deep network for pedestrian detection. *CVPR*, 2014.

[26] J. Martin, D. Vazquez, A. Lopez, J. Amores, and B. Leibe. Random forests of local experts for pedestrian detection. *ICCV*, 2013.

[27] M. Matthias, R. Benenson, R. Timofte, and L. Van Gool. Handling occlusions with franken-classifiers. *ICCV*, 2013.

[28] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. *CVPR*, 1997.

[29] W. Ouyang and X. Wang. Joint deep learning for pedestrian detection. *ICCV*, 2013.

[30] W. Ouyang and X. Wang. Single pedestrian detection aided by multi-pedestrian detection. *CVPR*, 2013.

[31] S. Paisitkriangkrai, C. Shen, and A. van den Hengel. Strengthening the effectiveness of pedestrian detection. *ECCV*, 2014.

[32] C. Papageorgiou, T. Evgeniou, and T. Poggio. A trainable pedestrian detection system. *Proceedings of Intelligent Vehicles, Stuttgart, Germany*, 1998.

[33] D. Park, D. Ramanan, and C. Folwkes. Multiresolution models for object detection. *ECCV*, 2010.

[34] D. Park, L. Zitnick, D. Ramanan, and P. Dollar. Exploring weak stabilization for motion feature extraction. *CVPR*, 2013.

[35] A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. *CVPR Workshop on Deep Learning in Computer Vision*, 2014.

[36] K. Rohr. Incremental recognition of pedestrians from image sequences. *CVPR*, 1993.

[37] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. *CVPR*, 2013.

[38] P. Viola, M. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. *ICCV*, 2003.

[39] C. Wojek, S. Walk, and B. Schiele. Multi-cue onboard pedestrian detection. *CVPR*, 2009.

[40] J. Yan, X. Zhang, Z. Lei, S.Liao, and S. Li. Robust multi-resolution pedestrian detection in trafiňĄc scenes. *CVPR*, 2013.

[41] X. Zeng, W. Ouyang, and X. Wang. Multi-stage contextual deep learning for pedestrian detection. *ICCV*, 2013.

[42] S. Zhang, C. Bauckhage, and A. Cremers. Informed haar-like features improve pedestrian detection. *CVPR*, 2014.