Learning Complexity-Aware Cascades for Pedestrian Detection

Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos, *Fellow, IEEE*

Abstract—The problem of pedestrian detection is considered. The design of complexity-aware cascaded pedestrian detectors, combining features of very different complexities, is investigated. A new cascade design procedure is introduced, by formulating cascade learning as the Lagrangian optimization of a risk that accounts for both accuracy and complexity. A boosting algorithm, denoted as *complexity aware cascade training* (CompACT), is then derived to solve this optimization. CompACT cascades are shown to seek an optimal trade-off between accuracy and complexity by pushing features of higher complexity to the later cascade stages, where only a few difficult candidate patches remain to be classified. This enables the use of features of vastly different complexities in a single detector. In result, the feature pool can be expanded to features previously impractical for cascade design, such as the responses of a deep convolutional neural network (CNN). This is demonstrated through the design of pedestrian detectors with a pool of features whose complexities span orders of magnitude. The resulting cascade generalizes the combination of a CNN with an object proposal mechanism: rather than a pre-processing stage, CompACT cascades seamlessly integrate CNNs in their stages. This enables accurate detection at fairly fast speeds.

Index Terms—Real-time pedestrian detection, detector cascades, boosting, complexity constrained learning.

1 Introduction

Pedestrian detection is an important problem in computer vision, with application to smart vehicles, surveillance, etc. Due to the real-time implementation requirements of many of these applications, the detector cascade architecture of [49] has a long history in pedestrian detection. It relies on the sliding window paradigm, processing each image patch with a sequence of binary classifiers, known as cascade stages. Each stage either rejects the image patch or passes it to the subsequent stages. This leads to two important properties. First, because most patches can be rejected by early stages of low complexity, detection can be fast. Second, because the remaining false positives can be rejected by complex detectors, deeper into the cascade, it can also be accurate. In fact, because final stages are rarely used, their complexity is not an impediment to fast detection.

While the cascade detection principle is intuitive, its implementation is far from trivial. Early cascade designs required extensive heuristics to determine the configuration of cascade stages [3], [49], [52]. A common assumption, by these methods, is that all features have equivalent complexity. This significantly simplifies the design, which reduces to choosing features so as to maximize detection accuracy. In fact, popular methods [3], [11] simply use a boosting algorithm (typically AdaBoost [16]) to design a non-cascaded classifier and then transform it into a cascade, by addition of thresholds. These approaches suffer from the problem that the "equivalent feature complexity" hypothesis only produces sensible cascades when applied to features that indeed have similar complexity.

In result, these methods have difficulties to accommodate features of widely varying complexity. This is a problem for applications that require diverse feature sets [1], [35], [57], and has become critical after the introduction of deep learning [26], [45]. On one hand, it is now well known that high detection rates require deep learning models. On the other, it is quite difficult to learn a cascade that combines these with the very efficient detectors needed in the early cascade stages. In fact, a complexity insensitive boosting algorithm, such as AdaBoost, will start by selecting the deep features, which are most accurate but also the most complex, for the early stages. This produces very slow cascades.

In object detection, an alternative strategy has emerged to address the intractability of sliding windows for deep learning. This consists of using an object proposal mechanism [48], which selects patches to process by a more powerful classifier. Various object detectors, such as the R-CNN [21], Fast-RCNN [20] and Faster-RCNN [38], are based on this architecture. While efficient, this strategy effectively implements a two stage cascade. Since there is no reason to believe that two stages guarantee the optimal trade-off between accuracy and speed, it does not necessarily guarantee the highest detection rate for a given complexity. For pedestrian detection, object proposals are frequently implemented with weak pedestrian detectors, sometimes cascaded detectors themselves [24]. The success of the somewhat arbitrary decomposition into a cascade and a deep classifier suggests that good performance should be possible for cascades with stages of deep learning models.

The introduction of deep learning has also generated a shift in the computer architectures used for object recognition. While classical cascades can be implemented on

- *Z. Cai and N. Vasconcelos are with the Department of Electrical and Computer Engineering, University of California, San Diego, San Diego, CA 92093, USA, E-mail: {zwcai,nuno}@ucsd.edu.*
- *M. Saberian is with Netflix Inc., Los Gatos, CA 95032, USA, E-mail: esaberian@netflix.com.*

Manuscript received February, 2018.

CPUs, deep learning model require dedicated GPUs. These create problems to applications, due to their high cost and energy consumption. In practice, detector design requires consideration of the trade-offs between accuracy, running time, energy consumption, and implementation cost. These different facets of "complexity" vary in importance from application to application. While speed is critical for a smart car braking system, energy (battery time) is much more important for drone applications. On platforms like drones or home surveillance, it may not be cost-effective to use a GPU-based detector of high accuracy, e.g it could be better to eliminate the GPU and complement a classic CPU-based detector with one using radar or some other modality.

In a CPU architecture, all these costs are (approximately) linear on the number of computations required for the detection. Computation (or speed) can thus be used as a universal measure of complexity, simplifying the design of the loss functions needed to learn complexity-aware algorithms. For GPU-based detectors this is much harder to accomplish. In this case, computation, speed, and energy consumption are not linearly related, due to the parallel implementation of all computations. In the detection context, where different image areas have different classification complexity, the optimal allocation of computation and energy is spatially varying. This is, however, at odds with the parallel implementation of image-wide layers, which guarantees optimal speed. Modern convolutional neural network (CNN) libraries favor the latter, leading to detectors that are fast but perform many unnecessary computations, increasing detector cost and energy consumption.

In this work, we address these problems by seeking an algorithm for optimal cascade learning under a criterion that penalizes *both* detection errors and complexity. For the latter, we introduce a definition of *complexity risk* akin to the empirical risk commonly used for classifier design. This makes it possible to define quantities such as complexity margins and complexity losses, and account for these in the learning process. We do this with recourse to a Lagrangian formulation, which optimizes for the usual classification risk under a constraint in the complexity risk. While the Lagrangian formulation can account for any of the facets of complexity discussed above, we focus on computational complexity in this work. A boosting algorithm that minimizes this Lagrangian is then derived.

This algorithm, denoted *Complexity-Aware Cascade Training* (CompACT), is shown to select inexpensive features in the early cascade stages, pushing the more expensive ones to the later stages. This enables the combination of features of vastly different complexities in a single detector. These properties are demonstrated by the successful application of CompACT to the problem of pedestrian detection, using a pool of features ranging from Haar wavelets to deep CNNs. In particular, we show that it is possible to embed deep CNNs as the final stages of classical detector cascades. This leads to mixed CPU/GPU solutions, that provide practitioners with flexible families of models, ranging from cheap and energy efficient CPU-based cascades, to cascades using GPU stages that emphasize speed or recognition accuracy.

Overall, this work makes four major contributions. First, it proposes a novel algorithm for learning complexity aware cascades, which optimally trade-off accuracy and speed. To the best of our knowledge, this is the first algorithm to *explicitly* account for variable feature complexity in cascade learning, supporting weak learners of widely different complexities. Second, CompACT seamlessly integrates handcrafted and CNN features in a unified cascaded detector. This generalizes the object proposal architecture, enabling the integration of CNN stages with stages of any other complexity. Third, it is shown that many large and expensive CNN models can be optimally embedded into the proposed CompACT cascades, enabling a range of complexities and accuracies. Finally, it is shown that the embedding of a large CNN detector, the MS-CNN [5], enables CompACT cascades to achieve accurate pedestrian detection rates on Caltech [13] and KITTI [19], at fairly fast speeds. While these contributions only address the design of cascaded detectors by boosting, we hope that the ideas will inspire subsequent research on explicit modeling of complexity into the objective functions used to train neural networks end-to-end.

2 Related Work

Cascades have long been used to detect objects such as faces [3], [49], [51], [52], pedestrians [11], [40], or cars [41]. Early approaches used heuristics to find a cascade configuration of good trade-off between accuracy and complexity [3], [49], [51], [52]. More recently, there have been efforts to optimize this trade-off [30], [40], [41], [59]. For example, [59] added a complexity term to the RealBoost loss, and [30], [40], [41] introduced the Lagrangian formulation we adopt. However, these methods use a single feature family throughout the cascade. The need for early cascades stages to be very efficient restricts this to a simple feature, e.g. a decision stump. For pedestrian detection, this is usually applied to the integral channel features of [12]. These extend the Haar-like features of [49] into a set of color and histogram-of-gradients (HOG) channels. A computationally efficient version of [49], denoted the aggregate channel features (ACF), was introduced in [11].

More recently, [35] complemented ACF with local binary patterns (LBP) and covariance features, for better detection accuracy. Several works proposed alternative feature channels, obtained by convolving different filters with the original HOG+LUV channels [1], [32], [56], [57]. The SquaresChnFtrs of [1] reduce the large number of features of [12], [49] to 16 box-like filters of various sizes. [32] extended the locally decorrelated features of [22] to ACF, learning four 5×5 PCA-like filters from each of the ACF channels. Instead of empirical filter design, [56] exploited prior knowledge about pedestrian shape to design informed filters. They later found, however, that such filters are actually not needed [57]. Instead, the number of filters appears to be the most important variable: features as simple as checkerboard patterns, or purely random filters, can achieve very good performance, as long as there are enough of them. However, these methods are relatively slow, since good performance requires convolution with large numbers of filters [35], [57].

While deep convolutional classifiers achieve impressive results on general object detection [21], e.g. on VOC, COCO or ImageNet, initial attempts with deep learning did not excel at pedestrian detection [2], [33], [43]. The difficulty of

sliding window implementation of deep models motivated the use of object proposal mechanisms [21], [24], [50] that pre-select the most promising image patches. Early models, such as the R-CNN [21] were fairly slow, but the introduction of the Fast-RCNN [20], allowing the computation of CNN features once per image, increased detection speed by an order of magnitude. Nevertheless, this model required an independent bottom-up stage for proposal generation. Later, the Faster-RCNN [38] integrated the generation of object proposals and region-wise classification within a single neural network, leading to a significant speedup.

Following these works, [27], [53], [55] presented good results for pedestrian detection. However, large CNN models tend to be costly and consume large amounts of energy, due to their GPU requirements. The massively parallel implementation required for speed makes difficult the optimal allocation of computation to different image areas. This makes it difficult to optimize the trade-offs between accuracy and speed, accuracy and energy consumption, accuracy and cost, etc. For example, [29], [37] obtained significantly higher speeds than those of the Faster-RCNN with a more efficient one-shot detection architecture but at the cost of substantially weaker detection rates. The implementation of high accuracy GPU-based detectors with low energy consumption or low-cost is mostly open at this point.

In this work, we define the optimal detector as that of maximal detection accuracy under a complexity constraint. Under this definition, the two-stage (proposal plus classification) cascade is not necessarily optimal. A more general multi-stage cascade could, in principle, achieve a better trade-off between detection accuracy and speed. This motivated us to consider multi-stage cascades that 1) include deep learning stages and 2) seek the optimal trade-off between accuracy and complexity. This is accomplished with a generalization of the classic boosting framework for cascade design, which complements the boosting loss with a complexity penalty.

The addition of this complexity term also establishes an explicit connection between complexity constrained detection and regularization. It is well known that AdaBoost can overfit when either the number of weak learners or their complexity are unchecked. In general, overfitting propensity depends on the VC-dimension or Rademacher complexity of the detector [42]. This has motivated the inclusion of complexity measures, e.g. Rademacher complexity [8], into the boosting loss. CompACT is similar in that it considers complexity albeit computation, not Rademacher complexity. Note, however, that both complexity measures are monotonically increasing functions of decision tree depth. Our experiments show that the complexity constraint in the proposed CompACT can indeed improve resistance to overfitting.

3 Complexity-Aware Cascade

In this section we introduce the CompACT algorithm.

3.1 AdaBoost

A decision rule $h(x) = sign[F(x)]$ of predictor $F(x)$ maps a feature vector $x \in \mathcal{X}$ to a class label $y \in \mathcal{Y} = \{-1, 1\}$. Boosting learns a strong decision rule by combining a set of weak learners $f_k(x)$,

$$F(x) = \sum_k f_k(x), \tag{1}$$

by functional gradient descent on a classification risk [17], [31]. AdaBoost [16] uses the exponential loss $\phi(yF(x)) = e^{-yF(x)}$, minimizing the empirical risk

$$\mathcal{R}_E[F] \simeq \frac{1}{|S_t|} \sum_i e^{-y_i F(x_i)}, \tag{2}$$

on a training set $S_t = \{(x_i, y_i)\}$. Boosting iterations compute the functional derivative of (2) along the direction of weak learner $g(x)$ at the current predictor $F(x)$,

$$\begin{aligned} < \delta\mathcal{R}_E[F], g > &= \frac{d}{d\epsilon}\mathcal{R}_E[F + \epsilon g]\big|_{\epsilon=0} \\ &= \frac{1}{|S_t|} \sum_i \Big[\frac{d}{d\epsilon} e^{-y_i(F(x_i)+\epsilon g(x_i))}\Big]\Big|_{\epsilon=0} \\ &= -\frac{1}{|S_t|} \sum_i y_i w_i g(x_i), \end{aligned} \tag{3}$$

where $w_i = w(y_i, x_i) = e^{-y_i F(x_i)}$. The predictor is updated by selecting the steepest descent direction within a weak learner pool $\mathbf{G} = \{g_1(x), \cdots, g_n(x)\}$,

$$\begin{aligned} g^*(x) &= \arg\max_{g \in \mathbf{G}} < -\delta\mathcal{R}_E[F], g > \\ &= \arg\max_{g \in \mathbf{G}} \frac{1}{|S_t|} \sum_i y_i w_i g(x_i). \end{aligned} \tag{4}$$

The optimal step size for the update is

$$\alpha^* = \arg\min_\alpha \mathcal{R}_E[F + \alpha g^*]. \tag{5}$$

For binary $g^*(x)$, this has a closed form solution

$$\alpha^* = \frac{1}{2} \log \frac{\sum_{i|y_i = g^*(x)} w_i^k}{\sum_{i|y_i \neq g^*(x)} w_i^k}. \tag{6}$$

Otherwise, the optimal step size is found by a line search.

3.2 Complexity-Aware Learning

Complexity-aware learning seeks the best trade-off between classification accuracy and complexity. This is a constrained optimization problem, where classification risk is minimized under a bound on a complexity risk $R_C[F]$,

$$F^*(x) = \arg\min_F R_E[F] \quad s.t. \quad R_C[F] < \gamma, \tag{7}$$

which is solved by minimizing the Lagrangian

$$\mathcal{L}[F] = \mathcal{R}_E[F] + \eta\mathcal{R}_C[F], \tag{8}$$

where η is a Lagrange multiplier that only depends on γ. To define a complexity risk, we note that (2) can be written as

$$\mathcal{R}_E[F] \simeq \frac{1}{|S_t|} \sum_i \phi[\xi(y_i, F(x_i))], \tag{9}$$

with $\phi(v) = e^{-v}$ and $\xi(y, F(x)) = yF(x)$. The function $\xi(\cdot)$ is the margin of example x under predictor $F(\cdot)$ and measures the confidence of the classification. Large positive (negative) margins indicate that x is correctly (incorrectly) classified with high confidence, and the margin is zero

for examples on the boundary. The loss $\phi(\cdot)$ is usually monotonically decreasing, penalizing all examples with less than a small positive margin. This forces the learning algorithm to concentrate on these examples, producing as few negative margins as possible. The exponential loss of AdaBoost makes the penalty exponential on the confidence of incorrectly classified examples.

In this work, we adopt a complexity risk of similar form

$$\mathcal{R}_C[F] \simeq \frac{1}{|S_t|}\sum_i \tau[\kappa(y_i, F(x_i))], \tag{10}$$

where $\kappa[y, F(x)]$ measures the complexity of classifying example x with $F(\cdot)$ and $\tau(\cdot)$ is a non-negative complexity loss. Drawing inspiration from the classification risk, we measure complexity with the complexity margin

$$\kappa[y, F(x)] = y\Omega(F(x)). \tag{11}$$

While $\Omega(F(x))$ can be any measure of complexity, in this work we focus on computational complexity, setting $\Omega(F(x))$ to the number of operations required to evaluate $F(x)$. (11) assigns positive (negative) complexity to positive (negative) examples, reflecting the fact that computation spent on negative examples is "wasted" or "negative" while that spent on positives is "justified" or "positive". While positives have to survive all cascade stages, negatives should be rejected with little computation. The loss $\tau(v)$ then determines the complexity-aware behavior of learning algorithms. For example, a decreasing $\tau(v)$ for $v < 0$ penalizes negative examples of large complexity. This encourages classifiers that reject negatives with as little computation as possible. On the other hand, an increasing $\tau(v)$ for $v > 0$ penalizes positives of large complexity.

3.3 Embedded Cascade

A cascaded classifier is a sequence of classification stages $h_i(x) = sgn[F_i(x) + T_i]$, where T_i is a threshold. A popular architecture, which we adopt in this work, is the embedded cascade, whose predictor has the embedded structure

$$F_k(x) = F_{k-1}(x) + f_k(x) = \sum_{j=1}^{k} f_j(x). \tag{12}$$

Classification complexity is measured by the average per stage complexity,

$$\Omega(F(x)) = \frac{1}{m}\sum_{k=1}^{m} r_k(x)\Omega(f_k(x)), \tag{13}$$

where, using $u[\cdot]$ to denote the Heaviside step function,

$$r_k(x) = \prod_{j=1}^{k-1} u\big[F_j(x) + T_j\big], \tag{14}$$

is an indicator of examples that survive all stages prior to k, i.e. $r_k(x) = 1$ if $F_i(x) + T_i > 0, \forall i < k$, and $r_k(x) = 0$ otherwise. Since the average complexity is bounded by the largest weak learner complexity, this leads to a more balanced Lagrangian in (8) than the total complexity.

3.4 Cascade Boosting

The minimization of (8) requires the functional derivative of the Lagrangian along the direction of weak learner $g(x)$ at the current predictor $F(x)$,

$$< \delta\mathcal{L}[F], g > = < \delta\mathcal{R}_E[F], g > + \eta < \delta\mathcal{R}_C[F], g >, \tag{15}$$

where $< \delta\mathcal{R}_E[F], g >$ is as in (3). To compute the derivative of the complexity risk we note that

$$\Omega(F(x) + \epsilon g(x)) = \begin{cases} \Omega(F(x)) & \text{if } \epsilon = 0 \\ \Omega(F(x) + \epsilon g(x)) & \text{otherwise.} \end{cases} \tag{16}$$

Defining $u(\epsilon)$ as $u(\epsilon) = 0$ for $\epsilon = 0$ and $u(\epsilon) = 1$ otherwise,

$$\begin{aligned}
&\Omega(F(x) + \epsilon g(x)) = \\
&= \Omega(F(x)) + u(\epsilon)\big[\Omega(F(x) + \epsilon g(x)) - \Omega(F(x))\big] \\
&= \Omega(F(x))[1 - u(\epsilon)] + u(\epsilon)\Omega(F(x) + \epsilon g(x)) \\
&= \Omega(F(x))[1 - u(\epsilon)] \\
&+ \frac{u(\epsilon)}{m+1}\left[\sum_{k=1}^{m} r_k(x)\Omega(f_k(x)) + r_{m+1}(x)\Omega(\epsilon g(x))\right] \\
&= \Omega(F(x))\left[1 - u(\epsilon) + \frac{m}{m+1}u(\epsilon)\right] \\
&+ \frac{u(\epsilon)}{m+1} r_{m+1}(x)\Omega(\epsilon g(x)) \\
&= \Omega(F(x))[1 - u(\epsilon)\zeta_m] + u(\epsilon)\frac{r_{m+1}(x)}{m+1}\Omega(\epsilon g(x)),
\end{aligned}$$

where $\zeta_m = 1 - \frac{m}{m+1}$ and we have used (13). Finally, since $\epsilon g(x)$ is simply a rescaling of $g(x)$, it is assumed that $\Omega(\epsilon g(x)) \approx \Omega(g(x))$, from which the expression above can be approximated by

$$\Omega(F(x)+\epsilon g(x)) = \Omega(F(x))[1-u(\epsilon)\zeta_m]+u(\epsilon)\frac{r_{m+1}(x)}{m+1}\Omega(g(x)). \tag{17}$$

Furthermore, since $u(\epsilon)$ is not differentiable, it is approximated by $\sigma(\epsilon) \approx u(\epsilon)$, where $\sigma(\epsilon)$ is a differentiable function with $\sigma(0) = 0$. Under these approximations,

$$\begin{aligned}
&< \delta\mathcal{R}_C[F], g > \qquad (18)\\
&= \frac{1}{|S_t|}\sum_i \left[\frac{d}{d\epsilon}\tau\Big[y_i\Omega\Big(F(x_i) + \epsilon g(x_i)\Big)\Big]\right]\Big|_{\epsilon=0} \\
&= \frac{1}{|S_t|}\sum_i y_i\tau'\left[y_i\Omega(F(x_i))\right]\left[\frac{d}{d\epsilon}\Omega\Big(F(x_i) + \epsilon g(x_i)\Big)\right]\Big|_{\epsilon=0} \\
&= -\frac{1}{|S_t|}\sum_i y_i\psi(y_i, x_i)\left[\frac{r_{m+1}(x_i)}{m+1}\Omega(g(x_i)) - \zeta_m\Omega(F(x_i))\right],
\end{aligned}$$

where

$$\psi(y_i, x_i) = -\tau'\left[y_i\Omega(F(x_i))\right]\sigma'(0). \tag{19}$$

Note that the derivative only depends on $\sigma'(0)$, other details of $\sigma(\epsilon)$ make no difference. Each boosting iteration updates $F(x)$ with a step along the steepest descent direction of (15) within the weak learner learner pool $\mathbf{G}$,

$$g^*(x) = \arg\max_{g\in\mathbf{G}} < -\delta\mathcal{L}[F], g > . \tag{20}$$

Combining (3), (15), and (18) and denoting $r_i = r_{m+1}(x_i)$, $\omega_i = \omega(y_i, x_i)$, $g_i = g(x_i)$, and $\psi_i = \psi(y_i, x_i)$, this is the direction that maximizes

$$\mathcal{D}[g] = \frac{1}{|S_t|}\sum_i y_i\left[\omega_i g_i + \eta\frac{r_i\psi_i\Omega(g_i)}{m+1}\right], \tag{21}$$

where we have disregarded the term $\zeta_m\Omega(F(x_i))$ of (18), because it does not depend on g and plays no role in the optimization. The optimal step size for the update is

$$\alpha^* = \arg\min_{\alpha} \mathcal{L}[F + \alpha g^*], \tag{22}$$

The cascade predictor is finally updated with

$$F^{new}(x) = F(x) + \alpha^* g^*(x). \tag{23}$$

Note that, from (19), $\sigma'(0)$ is a constant that rescales all ψ_i equally. Hence, in (21), it can be absorbed into η. We thus assume, without loss of generality, that $\sigma'(0) = 1$. This boosting algorithm is denoted the *complexity aware cascade training* (CompACT) boosting algorithm.

3.5 Properties

CompACT has a number of interesting properties. First, the contribution of each training example to the complexity term in (21) is multiplied by r_i. Hence, only examples that survive the current cascade F contribute to the complexity term. We refer to the x_i such that $r_i = 1$ as *active* examples. Note that, given the set of active examples

$$S_a(F) = \{(x_i, y_i) \in S_t | r_i = 1\}, \tag{24}$$

associated with F, (21) can be replaced by

$$\mathcal{D}[g] = \frac{1}{|S_t|}\left(\sum_i y_i\omega_i g_i + \sum_{i|r_i=1} y_i \frac{\eta\psi_i\Omega(g_i)}{m+1}\right). \tag{25}$$

This complies with the intuition that examples which do not reach stage $m+1$ during the cascade operation should not affect the complexity term for that stage.

Second, most implementations of cascaded classifiers use weak learners of example-independent complexity, i.e. $\Omega(g(x_i)) = \Omega_g, \forall i$. While this does not hold for the cascade in general (different examples can be rejected at different stages), it holds for the examples in S_a, i.e. $\Omega(F(x_i)) = \Omega_F, \forall x_i \in S_a$. In this case, the complexity weights of (19) only depend on the label y_i. Defining $\psi^+ = -\tau'[\Omega_F]$ ($\psi^- = -\tau'[-\Omega_F]$) as the value of ψ_i for positive (negative) examples, and π_F^+ (π_F^-) as the percentage of positive (negative) active examples, (21) reduces to

$$\begin{aligned}\mathcal{D}[g] &= \frac{1}{|S_t|}\left(\sum_i y_i\omega_i g_i + \frac{\eta}{m+1}\Omega_g \sum_{i|r_i=1} y_i\psi_i\right) &(26)\\ &= \frac{1}{|S_t|}\sum_i y_i\omega_i g(x_i) - \frac{\eta}{m+1}\Omega_g\frac{|S_a|}{|S_t|}\xi_F, &(27)\end{aligned}$$

with $\xi_F = \pi_F^-\psi_F^- - \pi_F^+\psi_F^+$. Since $|S_a|$ decreases with cascade length, the rescaling of η by $\frac{|S_a|}{|S_t|}$ gradually weakens the complexity constraint as the cascade grows. While in the early iterations there is pressure to select weak learners of reduced complexity, this pressure reduces as iterations progress. Gradually, complex weak learners are penalized less and the algorithm asymptotically reduces to a cascaded version of AdaBoost. This makes intuitive sense, since the latter cascade stages process a much smaller percentage of the examples than the earlier ones and have much less impact on the overall complexity. On the other hand, since the surviving examples are the most difficult to classify, accurate classification requires weak learner accuracy to increase with cascade length. This usually (but not always) implies that weak learner complexity increases as well because powerful features usually require heavy computation. By pushing the complexity to the later stages, the algorithm can learn cascades that are *both* accurate and computationally efficient. This effect is reinforced by the fact that $1/(m+1)$ also decreases with cascade length.

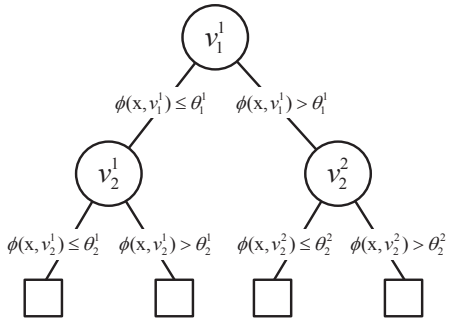


Fig. 1: Decision tree of depth 2. Circles represent classifier nodes, squares terminal nodes. $\phi(x, v)$ is the feature of x used at node v, θ a threshold.

Third, for homogeneous cascades, where every weak learner g in $\mathbf{G}$ has the same complexity, the second term of (27) is constant and has no impact on the optimization. In this case, (27) reduces to (3) and compACT boosting reduces to AdaBoost. Since this is the setting used by most previous cascade design algorithms [3], [40], [49], [52], compACT boosting can be seen as a generalization of these procedures. More generally, the weak learner pool is frequently heterogenous but of the form $\mathbf{G} = \{\mathbf{G}_1, \mathbf{G}_2, \cdots, \mathbf{G}_M\}$, where $\mathbf{G}_m$ is a homogeneous weak learner subset and M a number of heterogeneous weak learner classes. In this case, Ω_g is constant within each $\mathbf{G}_m$, only varying across the weak learner subsets, i.e. $\Omega_g = \Omega_{\mathbf{G}_m}, \forall g \in \mathbf{G}_m$. Hence, the second term of (27) becomes a penalty on the weak learner subsets $\mathbf{G}_m$. While different subsets are weighted according to their complexity, all weak learners within the same subset receive the same penalty. In this way, compACT boosting penalizes weak learner families of larger complexity but behaves like AdaBoost within each family.

3.6 Complexity Loss

The choice of complexity loss $\tau(v)$ determines the weight $\xi_F = \pi_F^-\psi_F^- - \pi_F^+\psi_F^+$ in (27). In our experience, the details of the loss function do not have a major impact on the performance of the learned cascade. For example, because the percentage of positive training examples is usually small, the behavior of $\tau(v)$ for $v > 0$ is not critical. For negative examples, the loss should be monotonically decreasing on v, but we have not seen great differences between, say, linear and exponential decay. For simplicity, we thus adopt the loss

$$\tau(v) = \max\{0, -v\}. \tag{28}$$

This is similar to the SVM hinge loss [9],

$$\tau(v) = \max\{0, 1 - v\} \tag{29}$$

but does not enforce a penalty on positive examples of low complexity. For complexity, it matters most to distinguish between positives and negatives, penalizing positives of low

Algorithm 1 Complexity-Restricted Tree Learning

Input:
Feature pool $\phi = \{\phi_1, \phi_2, \cdots, \phi_M\}$; training samples $S_t = \{(x_i, y_i)\}$.
Output:
The optimal decision tree g^*.
for $k = 1 : M$ **do**
Build the optimal homogenous decision tree g_k, using features from ϕ_k only and (4).
end for
Select the decision tree g^* from $\{g_k\}$ that maximizes (27).

complexity is not important. Instead, this loss encourages CompACT to focus on the fast rejection of negatives. Under it, $\psi_F^- = 1, \psi_F^+ = 0$ and $\xi_F = \pi_F^-$. Note that this makes the weight $\frac{|S_a|}{|S_t|}\xi_F$ of (27) equal to the ratio between the negative examples that are active and the total number of examples. This decreases as the cascade grows.

3.7 Weak Learners

CompACT supports the design of cascades with many types of weak learners. In this work, we follow the predominant trend in the literature and use binary decision trees.

3.7.1 Decision trees

As shown in Fig. 1, binary decision trees have two node types: classifier and terminal. Node v implements a classifier $g(x, v)$ by computing a feature $\phi(x, v)$ of sample x and comparing it to a threshold $\theta(v)$. The sample is then forwarded to one of the two children of the node, according to the results of this comparison. In this way, each sample follows a path from the root to a terminal node, transversing a single classifier node at each depth level. No classification or feature computation is performed in terminal nodes. Hence, in what follows, tree depth refers to classifier nodes only.

A binary decision tree g of depth L with classifier nodes $\{v_1^1, \cdots, v_l^i, \cdots, v_L^{n_L}\}$, where $n_l = 2^{l-1}$ is the number of nodes at depth l, has expected computation cost

$$\Omega(g) = \sum_{l=1}^{L}\sum_{i=1}^{n_l} p_l^i c_l^i, \tag{30}$$

where $p_l^i = p(v_l^i|x)$ is the probability that node v_l^i is traversed by sample x and c_l^i is the feature computation cost of node v_l^i. The probabilities p_l^i have some properties that derive from the tree structure, e.g. 1) if v_l^k is the parent of v_{l+1}^j then $p(v_{l+1}^j|x) = p(v_{l+1}^j|x, v_l^k)p(v_l^k|x)$; 2) $\sum_{i=1}^{n_l} p_l^i = 1$ for any depth l. Property 1) implies that $p(v_{l+1}^j|x) \leq p(v_l^k|x)$, i.e. the probabilities decrease with depth, and property 2) guarantees that the root node is traversed with probability $p(v_1^1|x) = 1$ for all x.

In this work, the feature pool ϕ used to build decision trees is composed of M classes, $\phi = \{\phi_1, \phi_2, \cdots, \phi_M\}$, where all features of class k have computational cost C_k. It follows that $c_l^i \in \{C_k\}_{k=1}^M$. Two strategies for learning decision trees, which are discussed next, were considered.

3.7.2 Complexity-restricted Tree

Under the first strategy, all decision trees use homogeneous features. At each boosting iteration, a decision tree g_k is built from feature sub-pool ϕ_k. The learning algorithm then chooses the best of the M trees g_k. Under this strategy, (30) reduces to

$$\Omega(g_k) = \sum_{l=1}^{L}\sum_{i=1}^{n_l} p_l^i C_k = LC_k, \tag{31}$$

where we have used property 2) above. Hence, given k, $\Omega(g_k)$ is a constant and the optimal weak learner of (20) is identical to that of (4), i.e. that chosen by AdaBoost. Hence, the learning of each cascade stage can be divided into two steps: 1) learn M homogeneous decision trees, by using the AdaBoost rule of (4) to find the optimal tree g_k for each each feature pool ϕ_k, and 2) select the homogeneous tree $g^* \in \{g_k\}$ that maximizes the complexity aware objective of (27). The procedure is summarized in Algorithm 1.

Algorithm 2 Complexity-Sensitive Tree Learning

Input:
Feature pool $\phi = \{\phi_1, \phi_2, \cdots, \phi_M\}$; training samples $S_t = \{(x_i, y_i)\}$; and tree depth L.
Output:
The optimal decision tree $g^* = \{g_1^1, \cdots, g_L^{n_L}\}$.
for each classifier node v **do**
1. Collect the training samples S_v that fall into node v, and compute the probability p_v that node v is traversed by a sample.
2. Find the node classifier g_v, by selecting the combination of feature from ϕ and threshold $\theta(v)$ that maximizes (32).
end for

3.7.3 Complexity-sensitive Tree

The second strategy optimizes the trade-off between complexity and accuracy even within each decision tree. Since global tree optimization is difficult, we use the popular local recursive optimization, where nodes are optimized sequentially. Consider (27) and the active sample $|S_a|$ arriving at current stage. Using S_v to denote the samples that reach node v, the empirical probabilities p_v are estimated with $p_v = \frac{|S_v|}{|S_a|}$. The binary classifier $g(x, v)$ implemented at node v is then learned with the following variant of (27)

$$\mathcal{D}[g_v] = \frac{1}{|S_v|}\sum_{i \in S_v} y_i \omega_i g_v(x_i) - \frac{\eta}{m+1}\frac{|S_a|}{|S_t|} p_v \xi_F \Omega_{g_v}, \tag{32}$$

where we use g_v to denote $g(x, v)$. The optimization procedure is described in Algorithm 2. Note that g_v can choose a feature from any of the feature pools ϕ_k. Hence, the tree is usually heterogenous and has two main properties. First, because the complexity penalty decreases with node probability, less visited nodes tend to use more complex features. Second, from property 1) above, deeper nodes have smaller probabilities than earlier ones, and feature complexity increases towards the tree leaves. Hence, early and popular nodes tend to use low complexity features, while features of higher complexity tend to be chosen for late or less popular nodes.

3.8 Bootstrapping and Thresholds

Bootstrapping is critical for effective object detection [11], [15]. A popular strategy is to collect hard negatives for an immature detector and use them to train a less immature detector. This is repeated for several rounds, until the detector is mature enough [2], [11], [32], [35], [57]. Usually,

Algorithm 3 Embedded Bootstrapping

Input:
Training images set $\mathcal{I}$; numbers $\{N_1, \cdots, N_K\}$ of weak learners at different stages, where $M = \sum_{k=1}^{K} N_k$.
Initialization:
Empty detector $\mathcal{F} = \{\}$; initial training set S_0.
Output:
Final detector $\mathcal{F} = \{\mathcal{F}_1, \cdots, \mathcal{F}_K\}$.
for $k = 1 : K$ **do**
1. Use S_{k-1} to learn a detector $\mathcal{F}_k$ of N_k weak learners.
2. Embed $\mathcal{F}_k$ in detector $\mathcal{F} = \mathcal{F} \cup \mathcal{F}_k$.
3. Run $\mathcal{F}$ to collect a new hard training set S_k.

end for

performance saturates after 4 to 5 bootstrapping rounds. A problem of this strategy is that the final training samples are not representative of the sample distribution of natural images. In fact, in the final bootstrapping round, most negatives are similar to the positives. Hence, there is no guarantee that the final detector will correctly classify the samples rejected by previous immature detectors.

To circumvent this, we use the bootstrapping strategy of Algorithm 3, which is similar to the methods of [18], [46]. The final detector is not the one learned in the final bootstrapping round. Instead, the detector learned in current round is embedded into the detector learned from all previous rounds, using (12). In our experiments, this strategy is more resistant to over-fitting than the classical one, sometimes outperforming it by a large margin.

Another important factor for cascade performance is the threshold used to reject negative examples. While thresholds can be learned [3], this increases the complexity of learning the complexity-aware cascade stages. In this work, the thresholds are selected as in [11], using a simplified version of the soft cascade of [3]. This consists of setting, in (14), $T_1 = 1$ and $T_{j+1} - T_j = \Delta$, $\forall j > 1$, where Δ is usually a small number, e.g. 0.005.

4 Pedestrian Detector Design

In this section, we discuss various details of the proposed pedestrian detector.

4.1 Heterogeneous Features

CompACT boosting seeks the optimal trade-off between accuracy and complexity, at each cascade stage. This is most effective when the feature pool contains features of various complexities. Our implementation draws such features from two main sources, the handcrafted aggregate feature channels (ACF) of [11] and a CNN feature set. Different filter sizes, computational mechanisms, platforms, etc. contribute to the diversity of feature complexities. A sample of the responses of some of these features to a pedestrian image is shown in Fig. 2. Note the diversity of details that the features highlight. This is unlike most previous works in the cascade literature, where most detectors use a single feature family.

In the literature, it is common to pre-compute a large number of feature responses at all image locations, before detection [32], [35], [57]. This, however, has unfeasible complexity for large feature pools (e.g. the 200,000∼500,000 features extracted per patch in [35], [57]) or when features are computationally intensive (e.g. the CNN features of [26], [45]). In these cases, it is neither tractable nor necessary to pre-compute all features at all locations. For example, a cascade of 2048 decision trees of depth 2, evaluates at most 4096 features per patch. Since the cascade rejects most candidate patches after a few stages, the most intensive features (e.g. CNN) are unlikely to be needed at most image locations. Hence, while pre-computation is useful for low-complexity features, complex features should be evaluated as necessary. We refer to the former as *pre-computed* features and the latter as computed *just-in-time* (JIT).

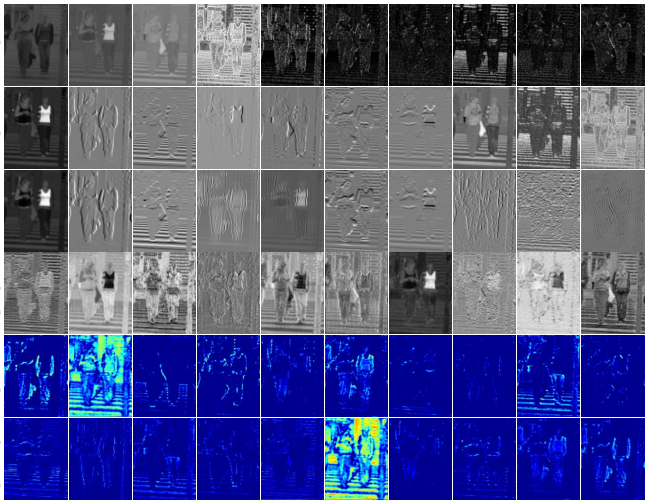


Fig. 2: Sample of the feature channels generated on a pedestrian image.

Fig. 3: Eight 2×2 checkerboard-like filters used in this work. Red (Green) is used to represent value +1 (-1).

4.1.1 Pre-computed Features

Our pre-computed feature set consists of ACF [11], mostly due to its computational efficiency. Following [11], we extract 10 LUV+HOG channels. In total, there are 16×8×10=1,280 ACF features per pedestrian patch.

4.1.2 Just-in-time Features

The JIT pool contains several feature subsets.

SS: The self-similarity (SS) features of [44] capture the difference between local patches and have achieved good performance on edge detection tasks [14], [28]. Following [14], [28], we compute SS features on a 12×6 grid of the 16×8 ACF patch. This results in $\binom{72}{2} \times 10 = 25,560$ SS features per pedestrian patch.

CB: Checkerboard features (CB) are obtained by convolving the ACF channels with a set of checkerboard filters. [57] has shown that a simple set of such features could achieve very good performance for pedestrian detection. Based on their observation that the number of features determines performance (rather than feature type), we adopt the set of 8 simple 2×2 checkerboard filters of Fig. 3. In total, there are 16×8×80=10,240 CB features per pedestrian patch.

LDA: Locally decorrelated HOG features, computed with linear discriminant analysis (LDA), have shown some superiority for object detection over HOG features [22]. [32]

Fig. 4: CNN architecture used to extract small CNN features.

showed that the computation of these features on ACF channels leads to a big improvement over ACF. We adopt this feature family, with filter size of 3×3. In total, there are 16×8×40=5,120 LDA features per pedestrian patch.

ACFCNN: SS, CB and LDA features are responses of simple hand-crafted filters to ACF features. A set of filters of higher complexity is also learned with a shallow CNN. This has the ACF feature channels as input and consists of a convolutional and a fully connected (fc) layer. Since ACF channels vary quite a bit in magnitude, they are passed through a batch normalization [25] layer before being fed to the CNN. The convolutional layer contains 32 filters of size 1×3×10 and 3×1×10, where 10 is the number of ACF channels. These filters have the same complexity and are denoted as ACFCNN. Although GPUs are used to train ACFCNN parameters, they are relatively simple filters and we use a CPU for their computation. In total, there are 16×8×64=8,192 ACFCNN features per pedestrian patch.

Small CNN: Beyond operators defined over ACF channels, we consider a set of CNN features extracted directly from the image to classify. The CNN has eight convolutional layers and one fc layer, whose details are given in Fig. 4. It was originally trained on 64×64 ImageNet images [39] and then fine tuned on a pedestrian dataset of 64×32 images. For feature extraction, we use the output of the 5^{th} and 7^{th} convolutional layers. Similarly to ACF, these can be seen as CNN feature channels and are denoted as CONV5 and CONV7, respectively. Inspired by the good performance and simplicity of the checkerboard features on ACF, we also compute them on the CONV5 feature channels. These are denoted CONV5CB features. Since CONV5 and CONV7 features are strong and can drive boosting towards over-fitting (especially CONV7), we only retain the $1/2$ channels of CONV5 and the $1/4$ channels of CONV7 that are more frequently used during learning. After the cascade is trained, we prune the unfrequently used channels and retrain it. Overall, there are 16×8×64=8,192 CONV5 features, 8×4×64=2,048 CONV7 features and 16×8×512=65,536 CONV5CB features per pedestrian patch.

4.2 Feature Complexity

Different facets of feature complexity, such as number of computations, computation cost, energy consumption, or speed, are of importance depending on the application. For features computed on CPUs, these variables are linearly related, and any of them can be used as a general measure of complexity. A reasonable choice is the number of floating-point operations (FLOPs) required by feature computation. However, complexity can depend on the implementation. For example, a Haar feature has complexity linear in the size of the image region it covers, but this leads to substantial redundant computation when overlapping features are used. The problem can be avoided by using the integral image of [49], which enables constant complexity (four integral image operations) for all features. This type of problem is much more complex for CNN features computed on GPUs. In this case, speed is optimized when features are computed in parallel. It is frequently faster to compute convolutions over the entire image, even if this computation is not needed in some areas, than to adapt the computation to the detection needs. This is, however, very inefficient in terms of energy consumption. Hence, for GPU-based CNN features, the two facets of complexity are not linearly related. The problem could be solved by shutting down individual CNN units, depending on the spatial location of their receptive field, but this is usually not supported by deep learning libraries. Hence, it is difficult to define a universal measure of complexity, of interest for all applications. This is even more complex for mixed CPU/GPU implementations, since the cost and energy consumed per FLOP are very different for the CPU and GPU architectures. In this case, the complexity measure can depend on the particular hardware implementation of the detector. We avoid this problem by introducing a generic but approximate measure of computational complexity, which relies on FLOPs to measure the complexity of CPU features and introduces the notion of "trigger complexity" for those computed on GPUs. The FLOPS and processing unit required by the various features used in this work are shown in Table 1.

TABLE 1: Feature complexity and processing unit

	ACF	SS	CB	LDA	ACFCNN	CONV5	CONV7	CONV5CB
FLOPs	1	2	4	9	30	5.84×10^7	8.67×10^7	5.84×10^7+4
Unit	CPU	CPU	CPU	CPU	CPU	GPU	GPU	GPU+CPU

4.2.1 ACF-based Feature Complexity

ACF features underlie SS, CB, LDA and ACFCNN, and are pre-computed before detection. The pre-computation cost, which occurs before cascade evaluation, can be ignored reducing complexity to a memory access. We assign to this a complexity of 1. SS, CB LDA and ACFCNN features are JIT features. Beyond ACF, they require a number of FLOPs proportional to their filter sizes. As shown in Table 1, their complexities are 2, 4, 9 and 30 respectively.

4.2.2 Trigger Complexity

The complexities of CNN and hand-crafted features are of a different nature. While CNN features are computed JIT, current libraries makes it inefficient to compute features individually. If CNN features are needed to classify an image region, it is significantly more efficient to evaluate all layers over the entire region than only the responses of the layers needed per sub-region. This makes it difficult to assign a complexity per feature. Instead, we rely on the concept of a trigger complexity Ω_{cnn}.

When (27) is evaluated for a previously unused CNN feature, $\Omega_g = \Omega_{cnn}$. Similarly to ACF, once the feature is computed its complexity is set to $\Omega_g = 1$. Different CNN features have different trigger complexities, which reflects their requirements in GPU FLOPS. In our implementation, CONV5 and CONV7 have trigger complexities of $\{\alpha_5\Omega_{cnn}, \alpha_7\Omega_{cnn}\}$, where $\alpha_5 : \alpha_7 = 1 : 1.48$. This is summarized in Table 1. When CNN architectures are independent, the same holds for their trigger complexities. However, CNN features are frequently dependent, e.g. CONV5

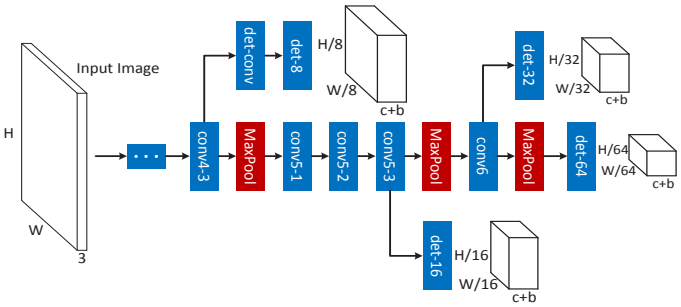


Fig. 5: MS-CNN proposal sub-network. Bold cubes are the network output tensors. c, b are the number of classes and bounding box coordinates, respectively.

and CONV7, which are features from the same network, with CONV7 requiring the computation of CONV5. To account for this, once CONV5 is computed trigger complexities become $\{0, (\alpha_7 - \alpha_5)\Omega_{cnn}\}$ and once CONV7 is computed they become $\{0, 0\}$. Once these features are triggered, complexity becomes $\{1, 1\}$ for both CONV5 and CONV7. CONV5CB features, obtained by checkerboard filtering the responses of CONV5, have trigger complexity equal to CONV5. Their after-trigger complexity is 4, the number of FLOPs required by checkerboard filters.

4.3 Embedding Large CNN Models

Large CNNs [23], [26], [45] are popular in computer vision. While, theoretically, they could be used in any cascade stage, this makes the iterative boosting optimization too computationally intensive. It is practical, however, to use a large CNN as the *final* cascade stage. This can, in principle, be done for any CNN detector in the literature. We next discuss the embedding of a few popular methods.

4.3.1 R-CNN

The R-CNN detector [21] applies a CNN to object proposals produced by an independent algorithm. It is trained with a sequential process of CNN fine-tuning, feature saving, SVM classifier training and bounding box regression training. This requires substantial human engineering, time and memory, due to the discrepancy between CNN classification and SVM detection scores. Some components, e.g. feature saving and SVM training, can be avoided by using CNN scores as detection scores. Due to the low computational efficiency of the R-CNN [21], embedding it in a CompACT cascade, which is already very discriminant and produces few detection candidates, can be very beneficial.

4.3.2 Fast-RCNN

The Fast-RCNN addresses many limitations of the R-CNN by sharing computation across proposals at different locations and scales. It requires a single forward image pass to produce all detections. This makes it much more efficient than the R-CNN. The original Fast-RCNN [20] was trained with third-party generic object proposals [48], which is not efficient nor effective for pedestrian detection. Better results could, in principle, be obtained by training a Faster-RCNN model [38] to detect pedestrians first, and then replace the RPN proposals by CompACT cascade proposals. The latter can, in this case, be seen as a strong proposal mechanism for the Fast-RCNN. To test this premise, we used Fast-RCNN detectors based on AlexNet, VGG-Net and ResNet-101.

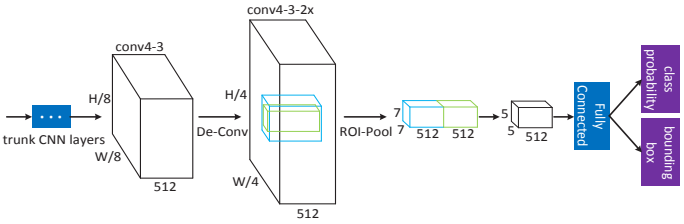


Fig. 6: MS-CNN object detection sub-network. "Trunk CNN layers" are shared with the proposal sub-network. W and H are the width and height of the input image. Green (blue) cubes represent object (context) region pooling.

4.3.3 MS-CNN

Despite the good Faster-RCNN results for general object detection, its pedestrian detection performance can be weak. This is because pedestrians can have wildly varying sizes, and the Faster-RCCN proposal network uses filters of fixed receptive field size. To overcome this inconsistency, we recently introduced a multi-scale extension of the Faster-CNN [38], denoted MS-CNN [5]. This consists of an object proposal sub-network and a detection sub-network, with the architectures of Fig. 5 and 6. By adding outputs at multiple layers of the object proposal network, each layer is encouraged to focus on objects within a certain scale range (See Fig. 5). This set of complimentary proposal generators is then combined into a strong multi-scale generator.

The MS-CNN is based on the VGG-Net [45]. Like the Faster-RCNN, it uses an RoI pooling layer. However, instead of operating on "conv4-3" feature maps, this is applied after a deconvolutional layer, which upsamples the feature maps twice for higher resolution. In addition to the pooling from the exact object region (green cube in Fig. 6), RoI pooling also operates on a context region (blue cube in Fig. 6) 1.5 times larger. An extra convolutional layer is used after this contextual RoI pooling, to reduce the number of parameters, and compress redundant context and object information. The proposal and detection sub-networks are trained jointly. To use the MS-CNN with CompACT, the proposal generation of Fig. 5 is ignored. Only the detection network of Fig. 6 is embedded in the CompACT cascade. This is similar to the embedding of the Fast-RCCN (of VGG-Net), but uses a more accurate pedestrian detector. It could thus enable better trade-offs between accuracy and complexity.

4.3.4 Embedding Mechanisms

The simplest way to embed a large CNN in a CompACT cascade is the "Proposal" mechanism of the R-CNN [21]. The CompACT cascade generates proposals, which are fed to the CNN. The final detection scores are produced by the CNN alone. This strategy assumes that the proposal detector is much less discriminant than the large CNN. This is frequently not true for CompACT cascades which, at least for pedestrian detection, can even be stronger than the large CNN. To address this, we propose an "Embedded" mechanism, where the large CNN acts as the final weak learner g of (23). Note that this has no loss of optimality, if α is learned with (22). While avoiding the computationally intensive fine-tuning of the CNN to the cascade proposals, it guarantees that the cascade is still *optimal* in an end-to-end manner. The CNN is simply a descent direction of (20) that is unavailable to prior stages. In summary, "Embedding"

differs from the "Proposal" mechanism in that 1) not only the bounding boxes but also the confidence scores of the cascade are forwarded to the deep CNN, and 2) the combination of the proposal mechanism (cascade) and deep CNN is optimal under the well defined risk of (8).

It should be noted that the stride (step between window evaluations) of CompACT is usually small, e.g. 4 pixels, while that of large CNN feature maps is relatively large and dependent on the network architecture, e.g. 16 pixels for the CONV5 features of VGG-Net. However, this is not a problem when the R-CNN or Fast-RCNN are embedded in the CompACT cascade, because the networks operate on object proposals. It suffices to feed the proposals to the network by image warping (R-CNN) or RoI pooling (Fast-RCNN). Another potential problem is that, despite its small stride, the cascade is not always strong enough to guarantee good localization, sometimes scoring the ground-truth window lower than neighboring windows. This, however, is not a problem for Fast-RCNN detectors, which can recover from poor proposal localization by relying on bounding box regression [20], [38]. Since the latter has a trivial computational cost, it is always used when either the Fast-RCNN or MS-CNN are embedded in a CompACT cascade.

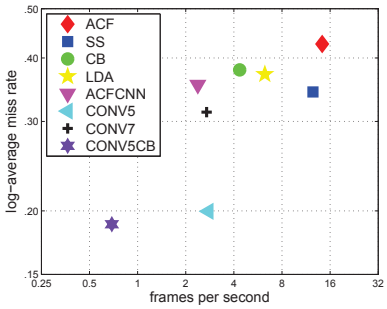
It remains to decide what CompACT proposals are forwarded to the CNN. As is common in the literature, when the cascade is used by itself, its output is post-processed with non-maximum suppression (NMS). Since there are frequently many overlapping proposals, this drastically reduces the number of deep CNN evaluations, the costliest stage from a computational point of view. On the other hand, it is sub-optimal to insert the NMS operation between the two stages from a detection point of view. We study the impact of embeddings both with and without NMS on the detection accuracy/complexity in the experimental section.

5 Experiments

In this section, we discuss various experiments performed to evaluate CompACT pedestrian detection cascades, using the Caltech [13] and KITTI [19] datasets.

5.1 Experimental Setting

We started by performing various ablation studies on Caltech, all using cascades of 2048 decision trees of depth of 2. These were learned with 7 bootstrapping rounds, using $\{32, 96, 128, 256, 512, 512, 512\}$ decision trees. The training and testing sets were as in [11]. A set of final experiments was then conducted using cascades of 4096 decision trees of depth of 5 and 9 bootstrapping rounds ($\{32, 96, 128, 256, 512, 768, 768, 768, 768\}$ trees). These were performed on both Caltech and KITTI (a decision tree depth of 4 was used in KITTI, due to its much smaller number of pedestrian instances). The pedestrian template size was 64×32 including a contextual border region of 14×12, as in [11]. On Caltech, we used the training set of [32], on KITTI we used the standard training set. The standard evaluation metrics were used on each dataset: MR (log-average miss-rate) on Caltech, and mAP (mean average precision) on KITTI. Since the minimum pedestrian height on KITTI is half of that of Caltech (25 pixels), the original test images were upsampled by 2. The detected bounding boxes (minimum height of 50) were then downsampled by 2. All experiments were run on a single CPU core (2.10GHz) of an Intel Xeon E5-2620 server with 64GB of RAM. An NVIDIA Tesla K40M GPU was used for CNN computations, with the remaining computations performed on CPU.



feature	MR	time (s)
ACF-based		
ACF	42.6	0.07
SS	34.29	0.08
CB	37.89	0.23
LDA	37.15	0.16
ACFCNN	35.41	0.42
CNN-based		
CONV5	19.96	0.36
CONV7	32.34	0.37
CONV5CB	18.83	1.35

Fig. 7: Cascade accuracy v.s. complexity for different features.

5.2 Homogeneous Cascade Comparison

We started with homogeneous cascades of a single feature family, the predominant architecture in the literature. In this case, the complexity penalty of (27) is equal for all weak learners, and CompACT reduces to AdaBoost. Fig. 7 compares homogeneous cascades based on ACF, SS, CB, LDA, ACFCNN, CONV5, CONV7 and CONV5CB. Note the improved performance of our ACF cascade implementation over [11], due to the bootstrapping strategy of Section 3.8.

Clearly, SS outperforms the other ACF-based features (ACF, CB, LDA, ACFCNN), achieving higher accuracy *and* speed. CB and LDA are more discriminant than ACF, but more complex. ACFCNN is slightly more accurate than CB, but its heavy features produce the slowest ACF cascade. The cost/benefits of CNN features are shown on the lower table. CONV5 has higher accuracy than all ACF-based features but five times the complexity. CONV7 has much worse detection performance than CONV5. We believe this is due to a coarser sliding window stride, since (see Fig. 4) CONV7 is downsampled twice more than CONV5. CONV5CB has the best detection, but only a marginal gain over CONV5 and four times the computation. Fig. 7 also shows that the spectrum of feature accuracies v.s. complexities is quite broad. This enables CompACT to select features of optimal accuracy vs complexity trade-off at each cascade stage.

5.3 CompACT Cascade Configuration

To test the ability of compACT to trade-off accuracy for complexity, we started by considering cascades of complexity-restricted trees (Section 3.7.2). Fig. 8 illustrates the configuration of cascades learned with different η in (8), showing the feature type selected every fifth stage. Consider the case of $\eta = 500$. The cheapest features (ACF) were the only selected for the first 200 stages, and rarely used after stage 500. This suggests that ACF features are very efficient but not very discriminant. A better trade-off between the two goals is achieved by SS features, which were selected throughout the training process. Note that SS features are competitive even for the later cascade stages, suggesting that they can be very discriminant in spite of their simplicity. Similarly, CB features were selected across a large range of cascade

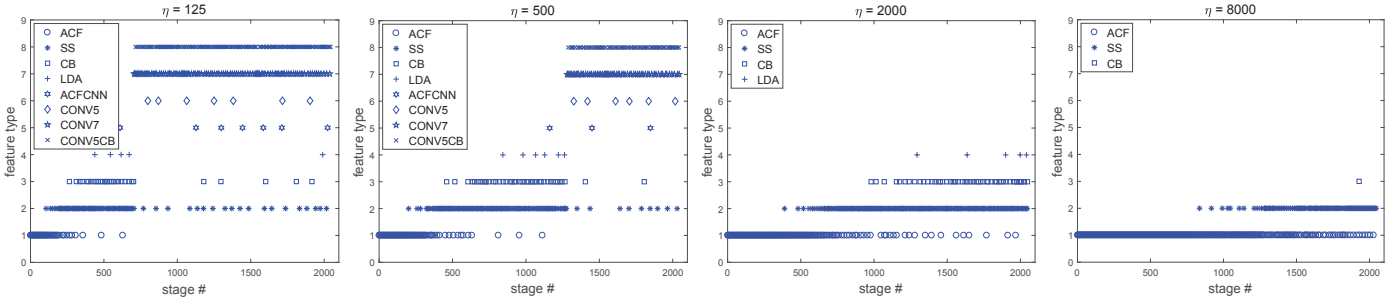
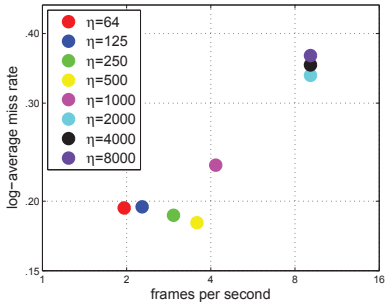


Fig. 8: Stage configuration learned with different trade-off coefficient η. Only one in five stages is shown.



η	MR	time (s)
64	19.45	0.51
125	19.54	0.44
250	18.87	0.34
500	**18.29**	0.28
1000	23.21	0.24
2000	33.65	0.11
4000	35.13	0.11
8000	36.52	0.11

Fig. 9: Accuracy v.s. complexity for different trade-off coefficients η.

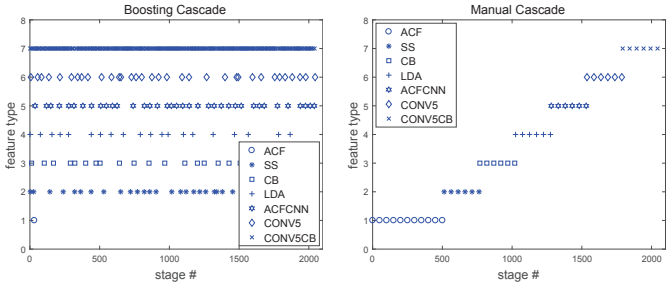


Fig. 10: Stage configuration for "Boosting" and "Manual" cascades.

stages. This is unlike LDA and ACFCNN, whose rare selection suggests a weak discrimination vs. complexity trade-off. More surprisingly, CONV5 features were also rarely selected, with CONV7 and CONV5CB dominating the late stages. This suggests that CONV7 and CONV5CB are more discriminant. Recall that, while CONV5 is more efficient in Table 1, CompACT boosting weighs complexity less heavily than discrimination in the late stages.

Fig. 8 also shows that the optimal cascade configuration depends strongly on the Lagrange multiplier η that controls the trade-off between accuracy and complexity. Recall, from (7)-(8), that η has an inverse relationship with the complexity bound γ. Hence, a smaller η (larger γ) should produce a slower and more accurate cascade, and vice versa. This is supported by Fig. 8, where large η leads to a cascade composed exclusively of low complexity features, while a small η allows the use of more complex features and the use of those features earlier on in the cascade. The overall trade-off between accuracy and complexity is shown in Fig. 9, which compares cascades learned with different η. Small η, which as shown in Fig. 8 induce CompACT to select expensive CNN features in the early stages, produce slower but more accurate cascades. Larger η, which prevent the selection of expensive features, produce faster but less accurate detectors. In the limit of $\eta \to \infty$ only the cheapest features are chosen throughout the cascade, and CompACT reduces to AdaBoost on the ACF feature pool. This is the method of [11].

Fig. 9 shows that between the two extremes of η, cascade speeds vary from about 2 to about 10 fps. Interestingly, accuracy plateaus as speed decreases. Note that accuracy does not improve as η decreases beyond $\eta = 500$, even though speed continues to decrease. Since the knee of the curve corresponds to the lightest cascade with the best accuracy on this dataset, this is the cascade of optimal trade-off between accuracy and complexity. The fact that cascades with heavier features have weaker detection rate suggests some over-fitting and, as discussed in Section 2, that the complexity penalty acts as a regularizer. In these experiments the optimal trade-off was achieved with $\eta = 500$.

One sensible question is whether CNN features would suffice to implement all cascades. After all, ACF features perform convolutional operations and a well trained CNN model should be able to learn them. Fig. 8 suggests this is not the case. This is likely because ACF features were designed with the explicit purpose of computational efficiency. Although they cannot guarantee high accuracy, they achieve a strong trade-off between accuracy and run-time. Note that, in Fig. 8, they are always chosen to implement the early cascade stages. On the other hand, CNN features are never chosen for cascades that emphasize computational efficiency (large η). While the CNN could, in principle, learn the ACF features, this would require 1) a CNN learning algorithm that somehow penalized feature complexity, which we are not aware of, and 2) probably some form of hand-coding, through specification of the architecture of early CNN layers.

5.4 Benefits of optimal accuracy/complexity trade-off

We next compared CompACT to algorithms for learning cascades of heterogenous features. Since there is no literature in this area, we considered two baselines. The first learns a cascade without complexity constraints ($\eta = 0$ in (27)), and is denoted as "Boosting". It is equivalent to applying existing cascade learning algorithms to the diverse feature set considered in this work[1]. The second attempts to "hand-code" the behavior of CompACT, restricting the unconstrained boosting algorithm ($\eta = 0$) to use certain feature types in different cascade stages. This restriction is based on feature complexity: features were ranked by complexity and used sequentially. ACF features were used in the first 512 stages and the other feature types used equally in the remaining stages. The resulting cascade is denoted as "Manual".

1. Since, in this case, all feature channels must have the same size (downsampling rate), CONV7 features were not used in this section.

TABLE 2: Comparison to "Boosting" and "Manual" cascades.

Method	ACF-based			ACF-based+Small CNN		
	Boosting	Manual	CompACT	Boosting	Manual	CompACT
MR	32.09	34.39	33.25	17.39	23.51	18.29
time (s)	0.76	0.11	0.11	2.46	0.23	0.28

TABLE 3: Comparison of complexity restricted and sensitive trees.

η		125	250	500	1000	2000	4000	8000	16000
restricted	MR	30.97	29.23	29.56	29.84	30.35	30.94	30.59	31.68
	Time (s)	0.105	0.104	0.103	0.102	0.101	0.101	0.100	0.100
sensitive	MR	29.32	28.30	29.33	29.78	30.16	30.28	30.91	31.31
	Time (s)	0.109	0.111	0.103	0.104	0.104	0.101	0.102	0.105

Fig. 10 shows the stage configurations of the "Boosting" and "Manual" cascades. Since "Boosting" does not penalize complexity, it frequently selects CONV5 and CONV5CB features for the early cascade stages. Note that this makes it too costly to compute features JIT, as discussed in Section 4.1.2. Instead, all channel features are computed at the beginning, enabling feature sharing. Only SS is computed JIT. Despite this feature sharing, the "Boosting" cascade was still several times slower than the CompACT cascade. This can be seen in Table 2, which compares CompACT ($\eta = 500$), "Boosting", and "Manual". The two sides of the table differ in that only ACF-based features were used on the left, while both these and the small CNN based features were used on the right. In both cases, the "Manual" cascade has low complexity but poor accuracy. "Boosting" learns a more accurate but significantly more complex cascade. CompACT has the best trade-off between accuracy and complexity. Note also that the introduction of the small CNN enabled substantially better cascades.

It is also instructive to compare these results to those of the homogeneous feature cascades commonly used in the literature, shown in Fig. 7. When compared to the heterogeneous CompACT cascades, all homogeneous cascades performed poorly. The CompACT cascade of multiple ACF features had higher accuracy than all homogeneous ACF-based feature cascades and was faster than most. The CompACT cascade of multiple ACF+Small CNN features had the best overall detection performance. Not only its detection accuracy beat that of the best single-feature cascade (CONV5CB in Fig. 7), it was also 5 times *faster*. These results illustrate the benefits of combining features of multiple complexities and optimizing the trade-off between accuracy and complexity.

5.5 Complexity Restricted vs. Sensitive Trees

So far, we have reported on heterogenous feature cascades using complexity restricted trees as weak learners. The next set of experiments compared these to cascades of complexity sensitive trees (Section 3.7). Here, we found that the inclusion of CNN features could lead to some instability. Although these features have high trigger complexity, they can have small penalty under (32) if the corresponding nodes are visited with small enough probability. Since small probabilities are difficult to estimate, this can lead to significant errors in node complexity estimates. In result, CNN features can be selected too often in early cascade stages. To avoid this, we used only ACF based features in these experiments. Decision trees of depth 3 were also used to magnify the differences between the two approaches.

As shown in Table 3, for equal η, complexity-sensitive trees are more accurate but slower than complexity-restricted trees. However, the differences are quite small, indicating there is no need to enforce trade-off optimization inside decision trees. This is a positive finding, in three ways. First, it shows that CompACT is quite robust, making it unnecessary to carefully optimize each weak learner. Second, it is in agreement with the boosting philosophy, whose point is exactly to aggregate "weak" (i.e. sub-optimal) learners . The requirement for a very detailed optimization of the decision tree would violate this principle. Third, complexity-restricted trees are simpler and easier to implement than complexity-sensitive trees, making the approach more practical. Given all this, we use complexity-restricted trees in what follows.

TABLE 4: Faster-RCNN baseline results.

Method	Alex	VGG	ResNet	MS-CNN
MR	30.29	15.59	14.91	10.0

TABLE 5: Embedding of large CNNs in CompACT cascades. "+" denotes additional time, after embedding the CNN, over CompACT.

CompACT	
MR	15.27
time (s)	0.34

R-CNN

	after NMS							
	Proposal				Embedded			
	Alex	VGG	ResNet	SDS	Alex	VGG	ResNet	SDS
MR	17.33	12.05	13.23	14.66	14.80	11.37	12.31	11.82
time (s)	+0.02	+0.06	+0.07	+0.07	+0.02	+0.06	+0.07	+0.07

	before NMS							
	Proposal				Embedded			
	Alex	VGG	ResNet	SDS	Alex	VGG	ResNet	SDS
MR	17.09	10.61	12.40	11.38	13.46	9.61	10.65	9.44
time (s)	+0.13	+0.21	+0.72	+0.54	+0.13	+0.21	+0.72	+0.54

Fast-RCNN

	after NMS							
	Proposal				Embedded			
	Alex	VGG	ResNet	MS-CNN	Alex	VGG	ResNet	MS-CNN
MR	27.55	14.83	15.93	12.65	20.69	11.95	13.00	10.91
time (s)	+0.03	+0.24	+0.26	+0.24	+0.03	+0.24	+0.26	+0.24

	before NMS							
	Proposal				Embedded			
	Alex	VGG	ResNet	MS-CNN	Alex	VGG	ResNet	MS-CNN
MR	25.32	13.23	12.64	10.12	18.49	10.01	10.65	9.06
time (s)	+0.03	+0.24	+0.28	+0.24	+0.03	+0.24	+0.28	+0.24

5.6 Embedding Large CNN models

We next considered the benefits of embedding large CNNs in CompACT cascades. A cascade of 4096 decision trees of depth 5 was used in these experiments[2]. Table 5 compares the performance of the CompACT cascade with small CNNs (denoted CompACT), versus its combination with the R-CNN and Fast-RCNN embedding of several popular models, including AlexNet [26], VGG-Net [45], and ResNet-101 [23], and the recently proposed SDS-RCNN [4] (excluding its RPN part). As discussed in Section 4.3, the large CNNs operated only on proposals generated by CompACT. The Fast-RCNN models were produced by removing the proposal generator of the Faster-RCNN models, whose baseline results are shown in Table 4. We investigated the benefits of 1) applying NMS to the cascade output before vs. after application of the large CNN, and 2) simply passing cascade detections to the CNN as a set of proposals (denoted "Proposal") vs. actually embedding the large CNN in the

2. Because the combination of CONV7 features and these deeper trees can sometimes over-fit, CONV7 features were not used in this section.

cascade with (12), to obtain an integrated detector (denoted "Embedded").

A number of interesting observations follow from the table. First, the theoretically sounder embedding of "Embedded" outperformed the more ad-hoc "Proposal" mechanism, in all cases. This shows that CompACT cascade scores and large CNN scores contain complimentary information. Second, CompACT is already a good pedestrian detector. Under "Proposal," Alex does not improve on the CompACT cascade. In fact, its MR (15.27) is comparable to the Faster-RCNN result of ResNet (14.91) in Table 4. Third, higher detection accuracy was always obtained by applying the large CNN models before NMS of the cascade output. Fourth, the embedding of existing detectors on the CompACT cascade can significantly enhance their performance. The comparison between the columns of "Proposal" under "before NMS" and the baseline Faster-RCNN results in Table 4, shows that the proposal generation by CompACT is stronger than the innate proposal generation of the Faster-RCNN. In addition, the combination of "Embedded" and "before NMS" outperformed the baseline Faster-RCNN by 11.8, 5.58, and 4.26, points, for Alex, VGG, and ResNet, respectively! Finally, the overall best results were obtained by embedding the MS-CNN before NMS. Although the MS-CNN uses the backbone VGG-Net, it is better suited for multi-scale detection than the vanilla Faster-RCNN. Since pedestrians can appear at very diverse scales, its use leads to a better trade-off between the accuracy and complexity of the pedestrian detector. These observations show that many CNN detectors can be successfully embedded in CompACT, producing an embedded cascade of higher accuracy than *both* the original CompACT and the large CNN detectors.

Table 5 also confirms a property that we have observed very consistently in all our experiments: the complementarity between ACF and CNN features. This can, for example, be seen by comparing the performance of the single CNN feature cascade of Fig. 7 (CONV5 19.96, CONV7 32.34, and CONV5CB 18.83) to the cascades using ACF+CNN features in Table 2 (Boosting 17.39 and CompACT 18.29). Note that, for the combination ACF+CNN, the CompACT cascade is not only faster (0.28 vs. 0.36, 0.37, 1.35) but also more accurate (18.29 vs. 19.96, 32.34, 18.83) than all single CNN feature cascades in Fig. 7. More surprisingly, the ACF features are complementary even with large CNN features. This can be seen in Table 5, where *all* large CNN models achieve better performance when embedded in the CompACT cascade than when simply using the latter as a proposal mechanism.

With respect to detection complexity, Table 5 shows that there is no additional cost for "Embedded" over "Proposal". However, in both cases, the cost of embedding the R-CNN increases linearly with the number of proposals. This, and the fact that NMS rejects many overlapping proposals, explains why "before NMS" is much more expensive than "after NMS," for this model. On the other hand, because the Fast-RCNN computation is shared among proposal regions, the complexity of embedding the Fast-RCNN is nearly constant, regardless of the number of proposals. These different settings make it possible to obtain multiple CompACT cascades of different detection accuracies and complexities. On the high-end of detection rates, the best trade-off between complexity and accuracy is achieved by the combination of Fast-RCNN, MS-CNN, "Embedded," and "before NMS." On the high-end of detection speeds, this holds for the combination of R-CNN, Alex, "Embedded," and "after NMS."

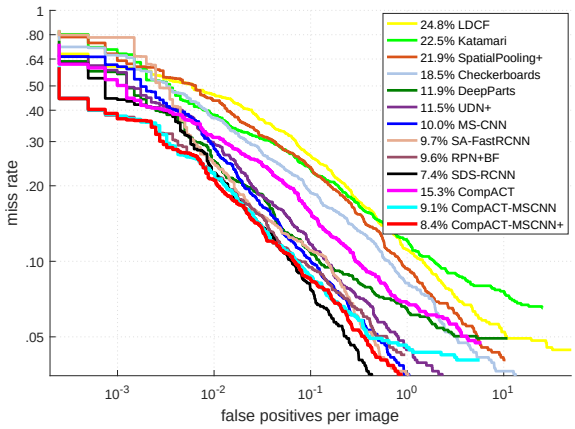


Fig. 11: Comparison to the state-of-the-art on Caltech (reasonable).

Finally, we note that the use of different embedding strategies and large CNN models enables a range of speed/accuracy trade-offs of interest for application developers. In this context, the flexibility enabled the family of models of Table 5 can far outweigh simple performance measures, such as speed or accuracy. By simply implementing CompACT, which does not use large CNN features, a developer can achieve a MR of 15.27 and a running time of 0.34. This is fairly cheap, and, as will be shown below, it is both faster and more accurate than other pedestrian detectors with this property. If higher accuracies are needed, the table offers a number of other choices. By augmenting CompACT with an R-CNN based on VGG-Net, "embedded" after NMS, accuracy can be increased by 3.9 points with a very marginal increase of 0.06s in running time. This requires a large CNN, but has virtually no additional cost in running time. It is a GPU based solution that emphasizes running time over accuracy. On the other hand, augmenting CompACT with a Fast-RCNN based on the MS-CNN, "embedded" before NMS, significantly increases accuracy (6.2 points) at the cost of a running time increase of 0.24s. This is one of the top detectors in the literature accuracy-wise, at the cost of a non-trivial increase in running time. The point is that the developer has a range of solutions to chose from, that cover a range of budgets in terms of accuracy, running time, energy consumption, and implementation cost.

5.7 Pedestrian Detection on Caltech

We next compared CompACT cascades to state of the art pedestrian detectors. We start by a comparison on Caltech, with the results of Fig. 11. As usual, we present curves of miss-rate v.s. false-positives per image (FPPI). CompACT uses ACF features and a small CNN, and outperforms all state of the art detectors of handcrafted features [2], [35], [57]. Note that some of these use many more features (HOG, LBP, spatial covariance, optical flow, multiple detectors, etc) and all are quite slow. On the other hand, CompACT runs at 3 fps on a relatively slow processor. CompACT-MSCNN embeds the MS-CNN in the last cascade stage. It outperforms most of the detectors using large CNNs, e.g. DeepParts [47], RPN+BF [55], SA-FastRCNN [27] and

TABLE 6: Pedestrian detection mAP comparison on KITTI.

Methods	Easy	Moderate	Hard	Time (s)
DPM	50.01	38.35	34.78	10
FilteredICF [57]	61.14	53.98	49.29	40
pAUCEnsT [35]	66.11	54.58	48.49	60
regionlet [50]	72.96	61.16	55.22	1
RCNN [24]	62.05	50.20	44.85	4
DeepParts [47]	70.46	58.68	52.73	1
CompACT	68.62	58.08	52.61	0.5
RPN+BF [55]	75.58	61.29	56.08	0.6
SDS-RCNN [4]	-	63.05	-	0.2
SDP+CRC [54]	77.81	64.25	59.31	0.6
Faster-RCNN [38]	78.35	65.91	61.19	2
Mono3D [6]	77.30	66.66	63.44	4.2
3DOP [7]	82.36	67.46	64.71	3
MM-MRFC [10]	82.37	69.96	64.76	0.05
F-PointNet [36]	87.81	77.25	74.46	0.17
CompACT-MSCNN	78.95	68.86	63.67	0.75

UDN+ [34]. The only competitive detector is the SDS-RCNN of [4]. This has a lower MR than CompACT-MSCNN, but the miss-rate v.s. FPPI curves behave very differently. CompACT-MSCNN dominates at low FPPI and SDS-RCNN at high FPPI. Overall, the two detectors are comparable. The comparison is also somewhat unfair because CompACT-MSCNN does not use a CNN to generate proposals. For completeness, we report on a modified version, denoted CompACT-MSCNN+, that combines the proposals generated by the MS-CNN and CompACT, and computes the final scores using the "Embedded" strategy. This improves performance at high FPPIs, which becomes close to that of SDS-RCNN.

5.8 Pedestrian Detection on KITTI

Table 6 compares the mAP of CompACT and CompACT-MSCNN to the state of the art on KITTI. Since KITTI test images are larger than those of Caltech, running times are higher for this dataset. Nevertheless, the CompACT cascade is among the fastest detectors. Among hand-crafted detectors (shown at the top of the table) only regionlet [50] has higher mAP. However, it uses very different features and is twice as slow. A more informative comparison is with channel based detectors, such as pAUCEnsT [35] and FilteredICF [57]. CompACT uses approximately the same number of feature channels (including the CNN model) but is both more accurate and faster. Again, CompACT even outperforms or is competitive with some detectors that use large CNNs, e.g. RCNN [24] and DeepParts [47].

The CompACT-MSCNN cascade, which includes a large CNN, outperforms these methods by a large margin. The second part of the table compares it to models based on large CNNs. The table is ordered by increasing mAP in the moderate task. The detectors slightly faster than CompACT-MSCNN, RPN+BF [55], SDS-RCNN [4] and SDP+CRC [54], have substantially weaker mAP, e.g. a loss of about 4 mAP points for SDP+CRC. The next three detectors, Faster-RCNN, Mono3D [6] and 3DOP [7], have weaker mAP and speed than CompACT-MSCNN. For completeness, we also present the very best results on this dataset, i.e. MM-FRC [10] and F-PointNet [36]. These methods use stereo and LIDAR data, which the CompACT-MSCNN does not leverage. They show that 3D information is very helpful on KITTI. The design of cascades leveraging this information is left for future work. Note that the F-PointNet cannot be applied to color images alone, e.g. datasets like Caltech, and MM-MRFC has much weaker performance without 3D data, e.g. 12% on Caltech.

TABLE 7: The results on CityPersons validation set.

Method	ACF	CompACT	Faster R-CNN	MS-CNN	CompACT-MSCNN+
MR	33.13	25.15	18.35	16.32	14.46

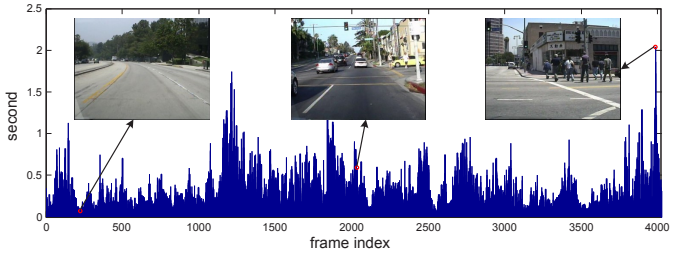


Fig. 12: Processing time spent per Caltech test image.

These results show that CompACT can generate high quality proposals even on datasets like KITTI, where proposal generation is known to be difficult [7]. It is particularly interesting that CompACT-MSCNN outperforms methods based on region proposal networks (RPN) [38], such as RPN+BF [55] or the Faster-RCNN. This shows that, using only small CNN models, CompACT can generate better proposals than the much more expensive RPN. Finally, it is also interesting that the closest competitors on Caltech, RPN+BF [55] and DeepParts [47], and the recent SDS-RCNN [4], are outperformed by CompACT-MSCNN on KITTI Moderate by a large margin: 7.57, 10.18 and 5.81 points, respectively.

5.9 Pedestrian Detection on CityPersons

CityPersons [58] is a recent pedestrian detection dataset, collected across multiple European cities. It has 2,975 training and 500 validation images, and 1,575 images for testing with held annotations. Images have size 2048×1024 and detection performance is evaluated as in Caltech. Table 7 compares the performace of the proposed detectors with the baselines of ACF [11] and Faster R-CNN [38], on the CityPersons validation set. For fair comparison, all detectors are implemented without upsampling the input images, and the Faster R-CNN was trained using the publicly available codebase of [5]. The CompACT cascade significantly outperforms ACF, and the CompACT-MSCNN+ cascade achieves better results than the Faster R-CNN. Note that the CompACT-MSCNN+ is an embedded cascade of higher accuracy than both the CompACT cascade and the very strong MS-CNN detector. These observations are consistent with the Caltech experiments.

5.10 CompACT as Attention Mechanism

One interesting property of CompACT, is that a detector cascade can be seen as an attention mechanism that assigns computation to image locations according to their detection complexity. Fig. 12 illustrates this property, summarizing the computation spent on each of the Caltech test images. Clearly, time complexity varies significantly with image content. The image on the left depicts a simple scene, with few objects and no pedestrian. Since simple and efficient features are enough to reject all windows, the image is processed with low complexity. The rightmost image depicts a more complicated scene with details of various scales

(foreground vs. background buildings) and several pedestrians. In this case, many image regions propagate until the final stages of the cascade, and complexity is high. In this sense, CompACT behaves more like the human visual system. For example, a human annotator quickly realizes that the left image contains no pedestrians, but requires significantly longer time to count the number of pedestrians on the rightmost image.

This behavior is very different from channel-wise object detectors [35], [57] and CNN based object detectors [5], [55], whose time complexity is nearly independent of image content. From an application point of view, this can have benefits and disadvantages. For example, in Fig. 12, processing time ranges from 0.05 to 2.0 seconds per frame, with a variance of about 0.04 seconds. This can create problems for applications that require a fixed processing time. For these, channel-wise or CNN detectors may be a better solution. On the other hand, there are benefits to the allocation of computation as needed, especially when power is an issue. For example, operating a Titan X GPU card 24-hours/day requires 6 kWh per day, i.e. 6 times the consumption of a fridge. This alone can preclude deployment on applications such as home surveillance, edge devices, drones, etc.

The ability of CompACT to detect the absence of pedestrians in a scene with simple CPU-based features enables drastic energy savings for applications on the home or edge, where such scenes can happen 99% of the time. On more complex and rare scenes the detector can activate energy intensive GPU features or even even ship the image regions in question to a central GPU, which serves all devices. Note that, on Caltech, the average image region size processed by a large CNN is 5.94% and 11.34% when embedding before and after NMS, respectively. This is opposed to the 100% of standard CNN implementations, e.g. the Faster R-CNN [38]. Besides saving energy, this would split the large cost of a GPU by various edge devices, which could themselves be inexpensive.

6 Conclusion

In this work, we proposed the CompACT boosting algorithm for learning complexity-aware detector cascades. By optimizing classification risk under a complexity constraint, CompACT produces cascades that push features of high complexity to the later cascade stages. This has been shown to enable the seamless integration of multiple feature families in a unified design. This integration extends to features, such as deep CNNs, that were previously beyond the realm of cascaded detectors. The proposed CompACT cascades also generalize the popular combination of object proposals+CNN, which they were shown to outperform. Finally, we have shown that a pedestrian detector learned by application of CompACT to a diverse feature pool achieves accurate detection rates on Caltech and KITTI, at fairly fast speeds.

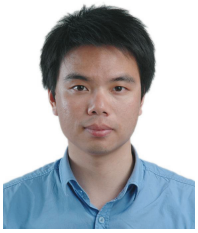
Acknowledgments

This work was funded by NSF Awards IIS-1208522 and IIS-1637941, ONR award UCSD-SUBK-N141-076, and a GPU donation from NVIDIA.

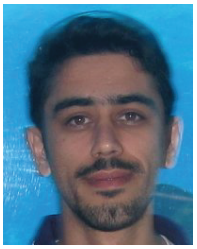
References

[1] R. Benenson, M. Mathias, T. Tuytelaars, and L. J. V. Gool. Seeking the strongest rigid detector. In *CVPR*, pages 3666–3673, 2013.
[2] R. Benenson, M. Omran, J. Hosang, , and B. Schiele. Ten years of pedestrian detection, what have we learned? In *ECCV, CVRSUAD workshop*, 2014.
[3] L. D. Bourdev and J. Brandt. Robust object detection via soft cascade. In *CVPR (2)*, pages 236–243, 2005.
[4] G. Brazil, X. Yin, and X. Liu. Illuminating pedestrians via simultaneous detection and segmentation. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 4960–4969, 2017.
[5] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV*, pages 354–370, 2016.
[6] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *CVPR*, pages 2147–2156, 2016.
[7] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals for accurate object class detection. In *NIPS*, pages 424–432, 2015.
[8] C. Cortes, M. Mohri, and U. Syed. Deep boosting. In *ICML*, pages 1179–1187, 2014.
[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
[10] A. D. Costea, R. Varga, and S. Nedevschi. Fast boosting based detection using scale invariant multimodal multiresolution filtered features. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 993–1002, 2017.
[11] P. Dollár, R. Appel, S. Belongie, and P. Perona. Fast feature pyramids for object detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(8):1532–1545, 2014.
[12] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, pages 1–11, 2009.
[13] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(4):743–761, 2012.
[14] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, pages 1841–1848, 2013.
[15] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, 2010.
[16] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT*, pages 23–37, 1995.
[17] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.
[18] J. Gall and V. S. Lempitsky. Class-specific hough forests for object detection. In *CVPR*, pages 1022–1029, 2009.
[19] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *CVPR*, pages 3354–3361, 2012.
[20] R. B. Girshick. Fast R-CNN. In *ICCV*, pages 1440–1448, 2015.
[21] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587, 2014.
[22] B. Hariharan, J. Malik, and D. Ramanan. Discriminative decorrelation for clustering and classification. In *ECCV*, 2012.
[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.
[24] J. H. Hosang, M. Omran, R. Benenson, and B. Schiele. Taking a deeper look at pedestrians. In *CVPR*, 2015.
[25] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.
[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.
[27] J. Li, X. Liang, S. Shen, T. Xu, and S. Yan. Scale-aware fast R-CNN for pedestrian detection. *CoRR*, abs/1510.08160, 2015.
[28] J. J. Lim, C. L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *CVPR*, pages 3158–3165, 2013.
[29] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. In *ECCV*, pages 21–37, 2016.

[30] H. Masnadi-Shirazi and N. Vasconcelos. High detection-rate cascades for real-time object detection. In *ICCV*, pages 1–6, 2007.
[31] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In *NIPS*, pages 512–518, 1999.
[32] W. Nam, P. Dollár, and J. H. Han. Local decorrelation for improved pedestrian detection. In *NIPS*, pages 424–432, 2014.
[33] W. Ouyang and X. Wang. Joint deep learning for pedestrian detection. In *ICCV*, pages 2056–2063, 2013.
[34] W. Ouyang, H. Zhou, H. Li, Q. Li, J. Yan, and X. Wang. Jointly learning deep features, deformable parts, occlusion and classification for pedestrian detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(8):1874–1887, 2018.
[35] S. Paisitkriangkrai, C. Shen, and A. van den Hengel. Strengthening the effectiveness of pedestrian detection with spatially pooled features. In *ECCV*, pages 546–561, 2014.
[36] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. pages 918–927, 2018.
[37] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, pages 779–788, 2016.
[38] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015.
[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015.
[40] M. J. Saberian and N. Vasconcelos. Learning optimal embedded cascades. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(10):2005–2018, 2012.
[41] M. J. Saberian and N. Vasconcelos. Boosting algorithms for detector cascade learning. *Journal of Machine Learning Research*, 15(1):2569–2605, 2014.
[42] R. E. Schapire, Y. Freund, P. Barlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *ICML*, pages 322–330, 1997.
[43] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, pages 3626–3633, 2013.
[44] E. Shechtman and M. Irani. Matching local self-similarities across images and videos. In *CVPR*, 2007.
[45] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
[46] D. Tang, Y. Liu, and T. Kim. Fast pedestrian detection by cascaded random forest with dominant orientation templates. In *BMVC*, pages 1–11, 2012.
[47] Y. Tian, P. Luo, X. Wang, and X. Tang. Deep learning strong parts for pedestrian detection. In *ICCV*, pages 1904–1912, 2015.
[48] K. E. A. van de Sande, J. R. R. Uijlings, T. Gevers, and A. W. M. Smeulders. Segmentation as selective search for object recognition. In *ICCV*, pages 1879–1886, 2011.
[49] P. A. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.
[50] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In *ICCV*, pages 17–24, 2013.
[51] R. Xiao, H. Zhu, H. Sun, and X. Tang. Dynamic cascades for face detection. In *ICCV*, pages 1–8, 2007.
[52] R. Xiao, L. Zhu, and H. Zhang. Boosting chain learning for object detection. In *ICCV*, pages 709–715, 2003.
[53] B. Yang, J. Yan, Z. Lei, and S. Z. Li. Convolutional channel features. In *ICCV*, pages 82–90, 2015.
[54] F. Yang, W. Choi, and Y. Lin. Exploit all the layers: Fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers. In *CVPR*, pages 2129–2137, 2016.
[55] L. Zhang, L. Lin, X. Liang, and K. He. Is faster R-CNN doing well for pedestrian detection? In *ECCV*, pages 443–457, 2016.
[56] S. Zhang, C. Bauckhage, and A. B. Cremers. Informed haar-like features improve pedestrian detection. In *CVPR*, pages 947–954, 2014.
[57] S. Zhang, R. Benenson, and B. Schiele. Filtered channel features for pedestrian detection. In *CVPR*, 2015.
[58] S. Zhang, R. Benenson, and B. Schiele. Citypersons: A diverse dataset for pedestrian detection. In *CVPR*, pages 4457–4465, 2017.
[59] W. Zheng and L. Liang. Fast car detection using image strip features. In *CVPR*, pages 2703–2710, 2009.



Zhaowei Cai is a Ph.D. candidate in Electrical and Computer Engineering Department at the University of California, San Diego. He received the B.S. degree in Automation from Dalian Maritime University in 2011. From 2011 to 2013, he worked as research assistant at Institute of Automation, Chinese Academy of Sciences. His current research interests are in computer vision and machine learning, including object detection and recognition.



Mohammad Saberian received the BS degrees in Electrical Engineering and Computer Science from Sharif University of Technology, Iran, and the MS and PhD degrees from Electrical and Computer Engineering Department at the University of California, San Diego in 2014. From 2014 to 2016, he was a research scientist at Yahoo labs and in 2016 he joined Netflix. He was the recipient of several awards including UC San Diego fellowship in 2008 and Yahoo Key Scientific Challenges award in 2011. He authored more than 20 peer-reviewed publications. His work spans various areas, including computer vision and machine learning.

Nuno Vasconcelos received the licenciatura in electrical engineering and computer science from the Universidade do Porto, Portugal, and the MS and PhD degrees from the Massachusetts Institute of Technology. He is a Professor in the Electrical and Computer Engineering Department at the University of California, San Diego, where he heads the Statistical Visual Computing Laboratory. He has received a NSF CAREER award, a Hellman Fellowship, several best paper awards, and has authored more than 150 peer-reviewed publications. He has been Area Chair of multiple computer vision conferences, and is currently an Associate Editor of the IEEE Transactions on PAMI. In 2017, he was ellected Fellow of the IEEE.