Learning Complexity-Aware Cascades for Deep Pedestrian Detection

Zhaowei Cai
UC San Diego
zwcai@ucsd.edu

Mohammad Saberian
Yahoo Labs
saberian@yahoo-inc.com

Nuno Vasconcelos
UC San Diego
nuno@ucsd.edu

Abstract

The design of complexity-aware cascaded detectors, combining features of very different complexities, is considered. 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 a pedestrian detector 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 state of the art performance on the Caltech and KITTI datasets, at fairly fast speeds.

1. Introduction

Pedestrian detection is an important problem in computer vision. Many of its applications, e.g. smart vehicles or surveillance, require real-time detection. Since, under the popular sliding window paradigm, there are close to a million windows per 640×480 pixel image, detection complexity can easily become intractable. This is an impediment to the deployment of sophisticated classifiers, such as deep learning models, in the pedestrian detection arena. The most popular architecture for real-time object detection is the detector cascade of [32]. It exploits the fact that most image patches can be assigned to the background class by evaluation of a few simple cascade stages. This guarantees computational efficiency without compromising accuracy, since the few resulting false positives can be rejected by more complex detectors, in the late cascade stages. Given that these are rarely used, their complexity is not an impediment to high detection speeds. In result, it is possible to have both efficient and accurate detection.

While the cascade detection principle is intuitive, its implementation is far from trivial. Early cascade designs required extensive heuristics to determine the cascade configuration [32, 35, 3], lacking the ability to explicitly optimize the trade-off between detection accuracy and complexity. A commonly used assumption is that all features have equivalent complexity. This significantly simplifies the design, which reduces to choosing the features that maximize detection accuracy. In fact, popular methods [3, 4] simply use a boosting algorithm (typically AdaBoost [8]) to design a non-cascaded classifier and then transform it into a cascade, by addition of thresholds. These approaches suffer from two main problems. First, they do not aim to select features that optimize the trade-off between detection accuracy and complexity. Second, the "equivalent feature complexity" hypothesis only produces sensible cascades when applied to features that indeed have similar complexity. This constraint is, however, frequently violated [1, 23, 37].

In fact, it has been remarkably difficult to accommodate, in cascade learning, features significantly heavier than those in common use. This problem is particularly pressing given the recent success of deep learning in object recognition [17, 29]. The intractable computation of a deep learning model under the sliding window paradigm is usually addressed with recourse to object proposal mechanisms [31], giving rise to a two-stage cascade that is far from optimal, in terms of the trade-off between detection accuracy and speed. For pedestrian detection, object proposals are frequently implemented with weak pedestrian detectors, sometimes cascaded detectors themselves [15]. Due to the ad-hoc nature of these solutions, deep learning models have not been competitive for pedestrian detection, contradicting their recognition and classification performance [17, 29].

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 lat-

ter, we introduce a measure of *implementation complexity* that allows the definition of a *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. 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 convolutional neural networks (CNNs).

Overall, this work makes three major contributions. First, it proposes a novel algorithm for learning a complexity aware cascade, so as to achieve an optimal trade-off between 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 detector. This generalizes the object proposal architecture, guaranteeing the seamless integration of CNN stages with stages of any other complexity. Finally, a CompACT cascade for pedestrian detection is shown to achieve state of the art results on both Caltech [6] and KITTI [11], at faster speeds than the closest competitors.

2. Related Works

Detector cascades learned with boosting are commonly used for detecting template-like objects, e.g. faces [32, 3, 35, 34], pedestrians [4, 25], or cars [26]. Early approaches used heuristics to find a cascade configuration of good trade-off between classification accuracy and complexity [32, 3, 35, 34]. More recently, optimization of the accuracy-complexity trade-off has started to receive attention [19, 25, 26, 38]. [38] empirically added a complexity term to the objective function of RealBoost. [19, 25, 26] introduced the Lagrangian formulation that we adopt, but use a single feature family throughout the cascade. Since early cascades stages must be very efficient, this implies adopting simple weak learners, e.g. decision stumps.

This has motivated extensive work on the design of efficient features. For pedestrian detection, the integral channel features of [5] have recently become popular. They extend the Haar-like features of [32] into a set of color and histogram-of-gradients (HOG) channels. More recently, a computationally efficient version of [5], denoted the aggregate channel features (ACF), has been introduced in [4]. [23] 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 [36, 37, 1, 21]. The SquaresChnFtrs of [1] reduced the large number of features of [5, 32] to 16 box-like filters of various sizes. [21] extended the locally decorrelated features of [13] to ACF, learning four 5×5 PCA-like filters from each of the ACF channels. Instead of empirical filter design, Zhang et al [36] exploited prior knowledge about pedestrian shape to design informed filters. They later found, however, that such filters are actually not needed [37]. Instead, the number of filters appears to be the most important variable: features as simple as checkerboard-like patterns, or purely random filters, can achieve very good performance, as long as there are enough of them. Although state-of-the-art performance has been achieved [23, 37], they are relatively slow, due to the convolution computations with several hundred filters.

While deep convolutional learning classifiers have achieved impressive results for general object detection [12, 14], e.g. on VOC2007 or ImageNet, they have not excelled on pedestrian detection [27, 22]. Benchmarks like Caltech [6] are still dominated by classical handcrafted features (see e.g. a recent comprehensive evaluation of pedestrian detectors by [2]). Recently, [15] transferred the R-CNN framework to the pedestrian detection task, showing some improvement over previous deep learning detectors [27, 22]. However, the gap to the state of the art is still significant. Deep models also tend to be too heavy for sliding window detection. This is usually addressed with object proposal mechanisms [12, 33, 15] that pre-select the most promising image patches. This two-stage decomposition (proposal generation and classification) is a simple cascade mechanism. In this work, we consider the seamless combination of these two stages into a cascade explicitly designed to account for both accuracy and complexity, so as to achieve detectors that are both highly accurate and fast.

3. Complexity-Aware Cascade Training

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 weaker learners $f_k(x)$,

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

using functional gradient descent on a classification risk [9, 20]. AdaBoost [8] is based on 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 training samples $S_t = \{(x_i, y_i)\}$. Boosting iterations compute the functional derivative of (2) along the direction of weak learner $g(x)$ at the location of the current predictor $F(x)$,

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

where $w_i = w(x_i) = e^{-y_i F(x_i)}$. 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 line search.

3.2. Complexity-Aware Learning

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

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

and is identical to the minimization of 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 $\xi(.)$ is the margin of example x under predictor $F(.)$ and measures the confidence of the classification. Large positive margins indicate that x is correctly classified with high confidence, large negative margins the same for incorrect classification, and a margin zero that the example is on the classification boundary. The loss $\phi(.)$ is usually monotonically decreasing, penalizing all examples with less than a small positive margin. This forces the learning algorithm to concentrate on these examples, so as to produce 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 consider complexity risks of a similar form

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

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

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

where $\Omega(F(x))$ is a function of the time required to evaluate $F(x)$, e.g. a number of machine operations or some other empirical measure of complexity. The complexity margin of (11) assigns positive (negative) complexity to positive (negative) examples, reflecting the fact that the 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 complexity 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 implemented as a sequence of classification stages $h_i(x) = sgn[F_i(x) + T_i]$, where T_i is a threshold. A popular architecture is the embedded cascade, whose predictor has the embedded structure,

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

In this paper, the cascade 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, it 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 location of 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 define $u(\epsilon)$ as $u(\epsilon) = 1$ for $\epsilon > 0$ and $u(\epsilon) = 0$ otherwise, and write

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

where $\zeta_m = 1 - \frac{m}{m+1}$ and we have used (13). Since $u(\epsilon)$ is not differentiable, it is approximated by $u(\epsilon) \approx \sigma(\epsilon)$, where $\sigma(\epsilon)$ is a differentiable function with $\sigma(0) = 0$, leading to

$$\begin{aligned}
&< \delta\mathcal{R}_C[F], g > \\
&= \frac{1}{|S_t|}\sum_i \left[\frac{d}{d\epsilon}\tau\Big[y_i\Omega\Big(F(x_i) + \epsilon g(x_i)\Big)\Big]\right]\Big|_{\epsilon=0} \\
&= -\frac{1}{|S_t|}\sum_i y_i\psi(y_i, x_i)\left[\frac{r_{m+1}(x_i)}{m+1}\Omega(g(x_i)) - \zeta_m\Omega(F(x_i))\right],
\end{aligned} \tag{16}$$

where

$$\psi(y_i, x_i) = -\tau'\left[y_i\Omega(F(x_i))\right]\sigma'(0). \tag{17}$$

Each boosting iteration updates $F(x)$ with a step along the steepest descent direction of (15) within the weak learner learner pool $\mathbf{G}$,

$$g^*(x) = \arg\max_{g\in\mathbf{G}} < -\delta\mathcal{L}[F], g > . \tag{18}$$

Combining (3), (15), and (16) 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

$$\mathcal{D}[g] = \frac{1}{|S_t|}\sum_i y_i\left[\omega_i g_i + \frac{\eta r_i\psi_i\Omega(g_i)}{m+1}\right]. \tag{19}$$

Note that the term $\zeta_m\Omega(F(x_i))$ of (16) does not depend on g and plays no role in the optimization. The optimal step size for the update is

$$\alpha^* = \arg\min_\alpha \mathcal{L}[F + \alpha g^*]. \tag{20}$$

The cascade predictor is finally updated with

$$F^{new}(x) = F(x) + \alpha^* g^*(x). \tag{21}$$

Note that, from (17), $\sigma'(0)$ is a constant that rescales all ψ_i equally. Hence, in (19), it can be absorbed into η. Without loss of generality, we assume 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 (19) 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, $S_a(F) = \{(x_i, y_i) \in S_t | r_i = 1\}$, associated with F, (19) can be replaced by

$$\mathcal{D}[g] = \frac{1}{|S_t|}\left(\sum_i y_i\omega_i g_i + \sum_{i|r_i=1} y_i\frac{\eta\psi_i\Omega(g_i)}{m+1}\right). \tag{22}$$

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 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 negative (positive) active examples, (19) reduces to

$$\mathcal{D}[g] = \frac{1}{|S_t|}\sum_i y_i\omega_i g(x_i) - \frac{\eta}{m+1}\frac{|S_a|}{|S_t|}\xi_F\Omega_g, \tag{23}$$

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

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

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.

The loss $\tau(v)$ enables fine-tuning of this general behavior, via ξ_F. In this work, we adopt the hinge loss $\tau(v) = \max(0, -v)$, for which $\psi_F^- = 1, \psi_F^+ = 0$ and $\xi_F = \pi_F^-$. This assigns no penalty to the complexity of positive examples, encouraging CompACT to focus on the fast rejection of negatives.

4. Pedestrian Detection

This section discusses the proposed pedestrian detector.

4.1. Feature Pools of Variable Complexity

CompACT seeks the optimal trade-off between accuracy and complexity, at each cascade stage. This is most effective when the feature pool is composed of features of various complexities. In the cascade literature, where most detectors use a single feature family, it is common practice to pre-compute a large number of feature responses at all image locations, before any detection takes place [21, 37, 23]. This, however, has unfeasible complexity if the feature pool is very large (e.g. the 200,000∼500,000 features proposed per patch in [37, 23]) or some features are computationally intense (e.g. the CNN features of [17, 29]). In these cases, it is neither tractable nor necessary to pre-compute all features at all image locations. For example, a cascade of 2048 decision trees of depth 2, will evaluate 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).

4.1.1 Pre-computed Features

Our pre-computed feature set consists of ACF [4], mostly due to its computational efficiency. Following [4], we extract 10 LUV+HOG channels. Since these are pre-computed, the complexity of using an ACF feature in any cascade stage is 1.

4.1.2 Just-in-time Features

The JIT pool contains several feature subsets. The ability to weigh accuracy vs. computation enables CompACT to seamlessly combine these feature sets.

SS: The self-similarity (SS) features of [28] capture the difference between local patches and have achieved good performance on edge detection tasks [18, 7]. Following [18, 7], 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 patch. Since the computation of an SS feature involves 2 ACF values, its complexity is 2.

CB: Checkerboard features (CB) are the result of convolving the ACF channels with a set of checkerboard filters. [37] has shown that a simple set of such features could achieve state-of-the-art 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 Figure 1. A CB has implementation complexity of 4.

LDA: Locally decorrelated HOG features, computed with linear discriminant analysis (LDA), have shown some superiority for object detection over HOG features [13]. [21] showed that the computation of these features on ACF channels leads to a big improvement over ACF. We adopt this feature family but, unlike [21], restrict the filter size to 3×3. LDA features have complexity 9.

CNN: In addition to operators defined over the ACF channels, we consider a set of CNN features. The CNN is a smaller version of the popular model of [17], with five convolutional layers and one fully connected layer. The CNN is applied to 64×64 image patches, the first convolutional layer has 32 filters, the remaining four have 64, and the fully connected layer consists of 1024 hidden units. All convolutional filters have size 3×3, and stride 1. The CNN model was originally trained with the ILSVRC14-DET dataset [24], using the cropped object patches, and then fine tuned on the target pedestrian dataset. For feature extraction, we only use the output of the 5^{th} convolutional layer, which can be seen as CNN feature channels, similar

to ACF. These features are denoted as CNN. 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 CNNCB features.

The complexity of CNN features is of a different nature than that of ACF features. First, the implementation on a different processor (GPU instead of CPU) makes the direct comparison of number of operations meaningless. Second, while the CNN features are computed on an "as needed" basis, the structure of the network makes it inefficient to compute each feature individually. If the CNN features are needed to classify a certain image window, it is significantly more efficient to compute the 5^{th} layer responses over the whole window than repeatedly applying the network to sub-window regions. We account for these difficulties by setting a trigger complexity Ω_{CNN} for CNN features. That is, in (23), CNN features have $\Omega_g = \Omega_{CNN}$ if no CNN feature has been used by the previous cascade stages to classify the current patch. Once the CNN features are computed, the complexity of using any CNN feature is 1, similar to ACF, while CNNCB features have complexity 4.

4.2. Embedding Large CNN Models

Large CNN models [17, 29] are now popular in computer vision. However, the use of these models in CompACT is challenging, due to the computational cost of embedding them in the iterative boosting algorithm. Our attempts to do so revealed impractical. Instead, we limited the use of a large CNN to the *final* cascade stage. Upon learning the cascade, we simply used a large CNN classifier as the final weak learner g of (21). Note that this has no loss of optimality, since α was learned with (20). The CNN is simply a descent direction of (18) unavailable to prior stages. It differs from the standard proposal+CNN approach in that 1) not only the bounding boxes but also the confidence scores of the cascade are forwarded to the deep CNN stage, and 2) the combination of the proposal mechanism (cascade) and large CNN is optimal under the well defined risk of (8).

In our implementation, we considered both the Alex [17] and VGG [29] models. Previous implementations [12] have warped cropped patches to size 227×227. However, such large patches are computationally expensive. We adopted the convolutional layers from the pre-trained models and two (randomly initialized) fully connected layers of 2048 units each. These networks were fine tuned to the pedestrian datasets using Caffe [16]. This allowed us to use the canonical 128×64 size for the pedestrian template. For Alex-Net, we used a convolution stride of 2 on the first layer, instead of 4 in the original model. For VGG-Net, we used all aspects of the original configuration other than input size and fully connected layers. While the original VGG-Net is approximately 8 times slower than the Alex-Net, the modified VGG-Net is only twice as slow.

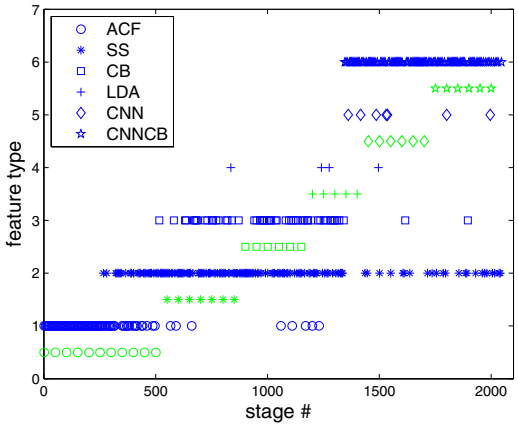

Figure 2. Stage configuration of the proposed CompACT cascade (blue) and the manually set cascade (green). Only one in five (fifty) stages is shown for the CompACT (manual) cascade.

5. Experiments

Various experiments were performed to evaluate the performance of CompACT cascades. All times reported are for implementation 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.

5.1. Cascade Configuration

We started by learning a CompACT cascade on the Caltech pedestrian dataset, using the set up of [4]. The cascade used 2048 decision trees of depth 2, and was bootstrapped 6 times during training, after stages $\{32, 128, 256, 512, 1024, 1536\}$, using the procedure of [10, 30]. Figure 2 presents the configuration of the learned cascade, showing how features of different complexities were chosen at different stages. ACF features, which are the cheapest, were the only selected for the first 200 stages, and rarely chosen after stage 500. This suggests that these features are very efficient but not very discriminant. A better trade-off between these two goals is achieved by the SS features, which were selected throughout the training process. It is particularly interesting that these features are competitive even for the later cascade stages. This suggests that they can be very discriminant despite their simplicity. Similarly, CB features were selected across a large range of cascade stages. This is unlike LDA features, which were rarely selected and do not appear to achieve a good trade-off between discrimination and complexity. More surprisingly, the CNN features were also rarely selected, with CNNCB dominating the late cascade stages. This suggests that the CNNCB representation is more discriminant. Recall that, while the CNN features are a little more efficient, CompACT boosting weighs complexity less heavily than discrimination in the late cascade stages.

Table 1. Comparison to single-feature cascades (MR: log-average miss-rate).

Method	Single Type						CompACT	
	ACF	SS	CB	LDA	CNN	CNNCB	ACF	CNN
MR	42.6	34.29	37.89	37.15	28.07	26.93	32.15	23.82
time (s)	0.07	0.08	0.23	0.16	0.87	2.05	0.11	0.28

Table 2. Comparison to multiple-feature cascades.

Method	ACF-based			ACF-based+Small CNN		
	Boosting	Manual	CompACT	Boosting	Manual	CompACT
MR	33.06	36.08	32.15	22.37	25.46	23.82
time (s)	0.41	0.11	0.11	2.69	0.28	0.28

5.2. Cascade Comparison

The CompACT cascade of the previous section was compared to cascades of other architectures. Table 1 presents a comparison to the predominant architecture in the literature: cascades of a single feature type. In this case, the complexity penalty of (23) is equal for all weak learners, and CompACT reduces to standard boosting. This was used to produce "standard" cascades of ACF, SS, CB, LDA, CNN and CNNCB features. We start by noting that the implemented ACF outperforms [4]. This is due to the use of a different bootstrapping strategy. Clearly, SS outperforms the other ACF-based features (ACF, CB, and LDA), achieving higher accuracy *and* speed. This confirms Figure 2, where SS features were selected throughout the detector. CB and LDA are more discriminant than ACF, but have higher complexity. CNN features have higher accuracy than all ACF-based features at the cost of a ten-fold increase in complexity over ACF. Finally, CNNCB has the best detection results, but only a marginal gain over CNN and much higher computation. When compared to CompACT cascades, all single feature cascades perform poorly. CompACT-ACF, which is restricted to ACF-based features, has higher accuracy than all ACF-based single feature cascades and is faster than most. CompACT-CNN, which includes all features, has the best detection performance. Note that not only its detection performance is clearly superior to the best single-feature cascade (CNNCB) but it is also 10 times *faster*.

Table 2 presents a comparison to cascades that combine multiple features. "Boosting" is a cascade learned without complexity constraints ($\eta = 0$ in (23)). This is equivalent to applying existing cascade learning algorithms to the diverse feature set considered in this work. "Manual" is an attempt to "hand-code" the behavior of CompACT, by restricting the boosting algorithm without complexity constraint ($\eta = 0$) to use certain types of features in different cascade stages. This restriction is based on feature complexity, as illustrated in Figure 2. The features were ranked by complexity and used sequentially, each feature type being used in approximately 400 stages. The two sides of Table 2 differ in that only ACF-based features were used on the left, while both these and the small CNN model were used on the right. In both cases, the "manual" cascade has low complexity but poor accuracy. "Boosting," on the other hand, can produce a more accurate cascade. The price is, however, a significant increase in complexity. CompACT achieves the best trade-off between accuracy and complexity. Note also the introduction of the small CNN model enables substantially better cascades.

Table 3. Performance of CompACT cascades using large CNNs.

Method	CompACT	Proposal		Intermediate		Embedded	
		Alex	VGG	Alex	VGG	Alex	VGG
MR	18.92	19.59	14.77	16.18	13.71	14.96	11.75
time (s)	0.25	+0.01	+0.03	+0.01	+0.03	+0.1	+0.25

5.3. Large CNN models

While the previous experiments only use small models, a number of experiments were performed with large models. These experiments were performed on both Caltech and KITTI, in both cases using cascades of 4096 decision trees of depth 5. These were bootstrapped 9 times, after stages $\{32, 128, 256, 512, 1024, 1536, 2048, 2560, 3328\}$. For Caltech, we used the training set size of [21], and the template size 64×32 as in [4]. On KITTI, test images were upsampled by 2 to detect pedestrians of height 25. This enabled the use of a single template size (64×32). After upsampling, the detected bounding boxes (minimum height of 50) had twice the actual object size. They were rescaled down by a factor of 2.

Table 3 compares the performance of the CompACT cascade with small CNNs (denoted CompACT) with several variants for the inclusion of large CNNs. In all these variants, the large CNN is computed only on windows selected by CompACT. The times noted as "+" reflect the added cost of running the image patches through it. The "Proposal" columns report to the use of the CompACT cascade as a proposal mechanism [12, 15] for the CNN. The "Embedded" columns report to the use of the large CNN as the last stage of the cascade, as discussed in Section 4.2. Finally, the "Intermediate" columns report to an intermediate between these two, in which the large CNN stage was only applied to the CompACT output, after non-maximum suppression (NMS). However, the prediction was that of (21), i.e. the CNN and CompACT scores were *combined*.

A number of interesting conclusions are possible. First, under the proposal architecture, only VGG improved on the CompACT cascade. For Alex, there was no benefit. This shows that the CompACT cascade is already a very good classifier. Second, the *embedding* of the large CNN on the CompACT model achieved the best results in all cases. This shows that the ComPACT cascade score contains information that complements that of the CNN scores. For both CNN models, it was better to combine scores with the CompACT cascade than to consider the latter simply as a proposal mechanism. Finally, the theoretically more sound embedding of the large CNN before NMS ("Embedding")

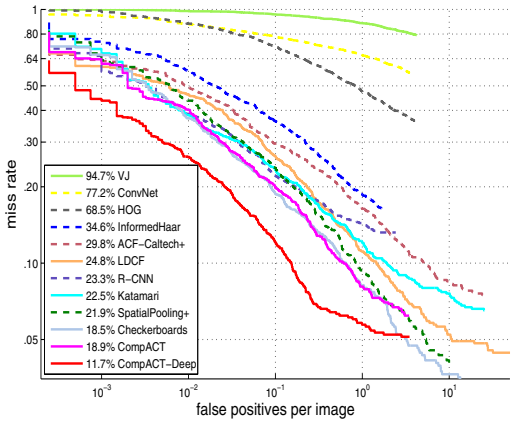

Figure 3. Comparison to state-of-the-art on Caltech (reasonable).

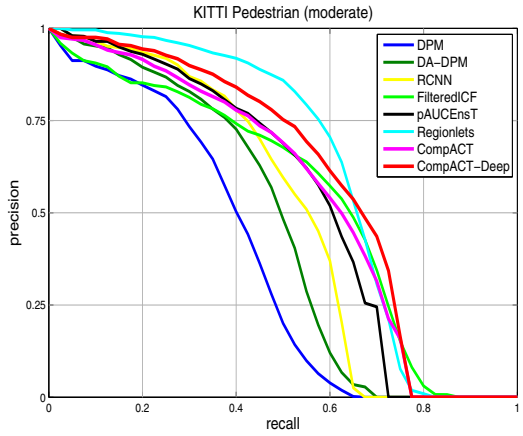

Figure 4. Comparison to state-of-the-art on KITTI Pedestrian (moderate).

always produced higher detection accuracy than the combination after NMS ("Intermediate"). This, however, had substantially less computation, since the number of bounding boxes is approximately 10 times smaller after NMS.

5.4. Comparison with the state-of-the-art

Figure 3 compares two CompACT pedestrian detectors to the state of the art on Caltech. CompACT refers to the model using "ACF + small CNN features", and CompACT-Deep to the model with the embedded VGG model in the last stage. CompACT achieves state-of-the-art performance, close to [37]. Note that the competing detectors - Katamari [2] and SpatialPooling+ [23] - combine many features (HOG, LBP, spatial covariance, optical flow, multiple detectors, etc.) and are all quite slow. The same holds for the state-of-the-art implementation of Checkerboards, which requires a large number of filter channels [37]. On the other hand, CompACT runs at 4 fps on a relatively slow processor. The CompACT-Deep cascade performs even better - 7 points better than the state-of-the-art [37] and 11 points better than the best deep pedestrian detector [15]! CompACT-Deep runs at 2fps and is faster than the competing detectors [2, 23, 37].

Figure 4 and Table 4 summarize performance on KITTI. Since test images are larger than in Caltech, running times are higher on this dataset. Nevertheless, the CompACT cascade is the fastest of all the state-of-the-art detectors. Note that it uses approximately the same number of feature channels (including the CNN model) as pAUCEnsT [23] and FilteredICF [37], which are both much less accurate and slower. R-CNN [15, 12], the only CNN detector on KITTI, is also substantially weaker than CompACT-Deep (difference larger than 8 points). Overall, the only approach competitive with the CompACT-Deep cascade is the Regionlets method of [33]. However, this work only reports classification times, excluding the time needed to generate proposals,

Table 4. Comparison to state-of-the-art detectors on KITTI. Note: * ignores the time needed to compute object proposals.

Methods	Easy	Moderate	Hard	Time (s)
DPM	45.50	38.35	34.78	10
DA-DPM	56.36	45.51	41.08	21
RCNN	61.61	50.13	44.79	4
FilteredICF	61.14	53.98	49.29	40
pAUCEnsT	65.26	54.49	48.60	60
Regionlets	73.14	61.15	55.21	1*
CompACT	65.35	54.92	49.23	0.75
CompACT-Deep	70.69	58.74	52.71	1

which can be on the order of several seconds. This is equivalent to only accounting for the processing time of the last stage of the CompACT-Deep model, which is 0.25 second.

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 state-of-the-art detection rates on Caltech and KITTI, with much faster speeds than competing methods.

Acknowledgements

This work was partially supported by NSF awards IIS-1208522 and CCF-0830535.

References

[1] R. Benenson, M. Mathias, T. Tuytelaars, and L. J. V. Gool. Seeking the strongest rigid detector. In *CVPR*, pages 3666–3673, 2013. 1, 2

[2] R. Benenson, M. Omran, J. Hosang, , and B. Schiele. Ten years of pedestrian detection, what have we learned? In *ECCV, CVRSUAD workshop*, 2014. 2, 8

[3] L. D. Bourdev and J. Brandt. Robust object detection via soft cascade. In *CVPR (2)*, pages 236–243, 2005. 1, 2

[4] 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. 1, 2, 5, 6, 7

[5] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, pages 1–11, 2009. 2

[6] 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. 2

[7] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, pages 1841–1848, 2013. 5

[8] 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. 1, 2

[9] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000. 2

[10] J. Gall and V. S. Lempitsky. Class-specific hough forests for object detection. In *CVPR*, pages 1022–1029, 2009. 6

[11] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *CVPR*, pages 3354–3361, 2012. 2

[12] 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. 2, 6, 7, 8

[13] B. Hariharan, J. Malik, and D. Ramanan. Discriminative decorrelation for clustering and classification. In *ECCV*, 2012. 2, 5

[14] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, pages 346–361, 2014. 2

[15] J. H. Hosang, M. Omran, R. Benenson, and B. Schiele. Taking a deeper look at pedestrians. In *CVPR*, 2015. 1, 2, 7, 8

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *MM*, pages 675–678, 2014. 6

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012. 1, 5, 6

[18] 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. 5

[19] H. Masnadi-Shirazi and N. Vasconcelos. High detection-rate cascades for real-time object detection. In *ICCV*, pages 1–6, 2007. 2

[20] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In *NIPS*, pages 512–518, 1999. 2

[21] W. Nam, P. Dollár, and J. H. Han. Local decorrelation for improved pedestrian detection. In *NIPS*, pages 424–432, 2014. 2, 5, 7

[22] W. Ouyang and X. Wang. Joint deep learning for pedestrian detection. In *ICCV*, pages 2056–2063, 2013. 2

[23] 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. 1, 2, 5, 8

[24] 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. 5

[25] M. J. Saberian and N. Vasconcelos. Learning optimal embedded cascades. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(10):2005–2018, 2012. 2

[26] M. J. Saberian and N. Vasconcelos. Boosting algorithms for detector cascade learning. *Journal of Machine Learning Research*, 15(1):2569–2605, 2014. 2

[27] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, pages 3626–3633, 2013. 2

[28] E. Shechtman and M. Irani. Matching local self-similarities across images and videos. In *CVPR*, 2007. 5

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 5, 6

[30] D. Tang, Y. Liu, and T. Kim. Fast pedestrian detection by cascaded random forest with dominant orientation templates. In *BMVC*, pages 1–11, 2012. 6

[31] 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. 1

[32] P. A. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004. 1, 2

[33] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In *ICCV*, pages 17–24, 2013. 2, 8

[34] R. Xiao, H. Zhu, H. Sun, and X. Tang. Dynamic cascades for face detection. In *ICCV*, pages 1–8, 2007. 2

[35] R. Xiao, L. Zhu, and H. Zhang. Boosting chain learning for object detection. In *ICCV*, pages 709–715, 2003. 1, 2

[36] S. Zhang, C. Bauckhage, and A. B. Cremers. Informed haar-like features improve pedestrian detection. In *CVPR*, pages 947–954, 2014. 2

[37] S. Zhang, R. Benenson, and B. Schiele. Filtered channel features for pedestrian detection. In *CVPR*, 2015. 1, 2, 5, 8

[38] W. Zheng and L. Liang. Fast car detection using image strip features. In *CVPR*, pages 2703–2710, 2009. 2