C^4: A Real-time Object Detection Framework

Jianxin Wu, *Member, IEEE,* Nini Liu, Christopher Geyer, and James M. Rehg, *Member, IEEE*

Abstract

A real-time and accurate object detection framework, C^4, is proposed in this paper. C^4 achieves 20 fps speed and state-of-the-art detection accuracy, using only one processing thread without resorting to special hardwares like GPU. Real-time accurate object detection is made possible by two contributions. First, we conjecture (with supporting experiments) that contour is what we should capture and signs of comparisons among neighboring pixels are the key information to capture contour cues. Second, we show that the CENTRIST visual descriptor is suitable for contour based object detection, because it encodes the sign information and can implicitly represent the global contour. When CENTRIST and linear classifier are used, we propose a computational method that does not need to explicitly generate feature vectors. It involves no image preprocessing or feature vector normalization, and only requires $O(1)$ steps to test an image patch. C^4 is also friendly to further hardware acceleration. It has been applied to detect objects such as pedestrians, faces, and cars on benchmark datasets. It has comparable detection accuracy with state-of-the-art methods, and has a clear advantage in detection speed.

Index Terms

Object detection, CENTRIST, Real-time.

J. Wu is with the National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: wujx@lamda.nju.edu.cn.

N. Liu is with the School of Computer Engineering, Nanyang Technological University, Singapore. E-mail: liunini2012@gmail.com.

C. Geyer is with the iRobot Corporation, Bedford, MA 01730, USA. E-mail: cgeyer@irobot.com.

J. Rehg is with the Center for Behavior Imaging and the School of Interactive Computing at the Georgia Institute of Technology, Atlanta, GA 30332, USA. E-mail: rehg@cc.gatech.edu.

I. Introduction

Object detection in images and videos is important in a wide range of applications that intersect with many aspects of our lives: surveillance systems and airport security, automatic driving and driver assistance systems in high-end cars, human-robot interaction and immersive, interactive entertainments, smart homes and assistance for senior citizens that live alone, and people-finding for military applications. The wide range of applications and underlying intellectual challenges of object detection have attracted many researchers' and developers' attention from the very early age of computer vision and image processing; and they continue to act as hot research topics in these fields.

The goal of this paper is to detect objects from grayscale images in real-time, with a high detection rate, and few false positives. In particular, for certain applications with special hardware or environmental constraints, e.g., human detection on-board a robot, the computational efficiency of the detector is of paramount importance. Not only must human detection run at video rates, but it also can use only a small number of CPU cores (or a small percentage of a single CPU core's cycles) so that other important tasks such as path planning and navigation will not be hindered. Thus, real-time or faster than real-time object detection has high impact in both research and applications.

Recent progresses in object detection have advanced the frontiers of this problem in many aspects, including features, classifiers, detection speed, and occlusion handling. For example, detection systems for some object types have been quite accurate, like faces [1], [2], [3] and pedestrians [4], [5], [6], [7]. However, at least two important questions still remain open:

- **Real-time detection**. The speed issue is very important, because real-time detection is the prerequisite in most of the real-world applications [8] and in a robot in particular.
- **Identify the most important information source**. Features like Haar [1], HOG [4] and LBP [7] have been successful in practice. But we do not know clearly yet what is the critical information encoded in these features, or why they achieve high detection performance in practice.

In this paper, we argue that these two problems are closely-related, and we demonstrate that an appropriate feature choice can lead to an efficient detection framework. In fact, feature computation is the major speed bottleneck in existing methods. Most of the time is spent in computing the features (including image preprocessing, feature construction, and feature vector normalization, etc.)

This paper makes three contributions. First, we present conjectures (with supporting experiments) that the contour defining the outline of an object is essential in many object detection problems (Sec. III-A). Similarly, we conjecture that *the signs of comparisons among neighboring pixels are critical to represent* *a contour, while the magnitudes of such comparisons are not as important.* Thus, for objects that can be distinguished mainly based on their contours, we need to capture this information.

Second, we propose to detect such objects using the contour cues, and show that the recently developed CENTRIST [9] feature is suitable for this purpose (Sec. III-B). In particular, it encodes the signs of local comparisons, and has the capability to capture global (or large scale) structures and contours. We also compare CENTRIST and other features in Sec. III-C.

Third, CENTRIST is very appealing in terms of detection speed. In Sec. IV, we describe *a method for using CENTRIST and a linear classifier for object detection, which does not involve image preprocessing or feature vector normalization.* In fact, we show that it is not even necessary to explicitly compute the CENTRIST feature vector, because it is seamlessly embedded into the classifier evaluation, achieving video-rate detection speed. Beyond that, we use a cascade classifier. After fast rejections by the linear classifier, we use an additional non-linear classifier to achieve quality detection. The proposed framework is named as C^4, since we are detecting objects emphasizing the human $\underline{\text{c}}$ontour using a $\underline{\text{c}}$ascade $\underline{\text{c}}$lassifier and the $\underline{\text{C}}$ENTRIST visual descriptor. The complete C^4 framework is presented in Sec. V. Pedestrian detection using C^4 was originally published in [10] as a conference presentation.

Finally, the C^4 framework is applied to detect objects on benchmark datasets (pedestrians, faces, and cars) in Sec. VI. Experimental results on benchmark datasets show that the C^4 framework achieves comparable detection quality as state-of-the-art detectors for these respective objects, and C^4 has a clear advantage in the detection speed. Using only one single CPU core (not involving GPU or other special hardware), C^4 achieves 20 fps detection speed on 640×480 images, and 109 fps on 320×240 images. The limitations and drawbacks of C^4 are also raised, with suggestions for future improvements.

The C^4 detection software is available at https://sites.google.com/site/wujx2001/home/c4, with pre-trained models for pedestrian detection.

II. Related work

There have been numerous works published on object detection. Before presenting the contributions of this paper, we first briefly review a few closely related papers.

Accurate detection is still a major interest in object detection, especially in terms of high detection rate with low FPPI (false positive per image) [5], [11]. Achievements have been made in two main directions: features and classifiers.

Various features have been applied to detect objects, e.g., Haar features for faces [1] and pedestrians [12], and edgelets for pedestrians [13]. However, HOG is probably the most popular feature in object detection [4], [6], [14], [15], [7]. The distribution of edge strength in various directions seem to efficiently capture objects in images, especially for pedestrians. Recently, variants of Local Binary Pattern (LBP) also show high potentials [7]. A recent trend in human detection is to combine multiple information sources, e.g., color, local texture, edge, motion, etc. [16], [7], [17], [15]. Introducing more information channels usually increases detection accuracy rates, at the cost of increased detection time.

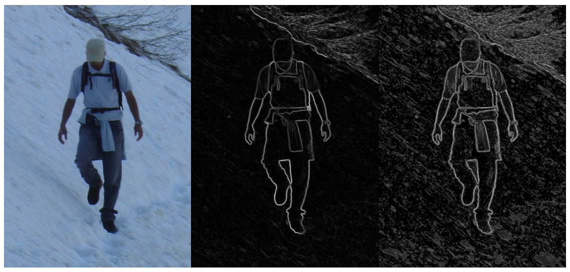
In terms of classifiers, linear SVM is widely used, probably for its fast testing speed. With the fast method to evaluate Histogram Intersection Kernel (HIK) [14], [18], HIK SVM was used to achieve higher accuracies with slight increase in testing / detection time. Sophisticated machine learning algorithms also play important roles in various object detection systems. A very influential approach is the part-based, discriminatively trained latent SVM classifier by Felzenszwalb et al. [6]. This detector is a general object detection framework that have been applied to detect tens of object types. The cascade classifier [1] has been successfully used for detecting faces and pedestrians [12], [1]. In [19], [20], sparse learning techniques is used to select a small number of features and construct cascade classifiers. Hough forests, which performs a generalized Hough transform for object detection, was proposed in [21] and achieved high detection accuracy in benchmark detection datasets.

Another important research topic in object detection is to improve the speed of detection systems. It is a common practice to use extra hardware like the GPU to distribute the computing task to hundreds of GPU cores in parallel, e.g., in [22], [7], [23]. The cascade classifier framework is an algorithmic approach that first makes face detection run in real-time [1], by using a data structure called integral images. The cascade classifier in [1], however, is only applicable to objects that has a fixed aspect ratio. Another algorithmic advance, ESS (Efficient Subwindow Search) [24], can greatly accelerate the search for objects with arbitrary aspect ratio, by using a branch and bound strategy.

III. CENTRIST Based Object Detection

In this section, we begin to present the C^4 object detection framework, starting from the question "which feature or visual representation is to be used?"

For this question, we present the following two *conjectures* and provide some supporting arguments: contour is a key information for detection of many objects; and, the signs of comparisons among neighboring pixels are key to capture useful information in the contour. Both hypotheses are supported by experiments presented in this section, using pedestrian detection as an example.



(a) Original image (b) Sobel image (c) Only signs

Fig. 1. Detecting humans from their contours (Fig. 1b) and signs of local comparisons (Fig. 1c).

A. Conjectures: Signs of local comparisons are critical for encoding contours and object detection

Hypothesis 1: For many object detection tasks, the important thing is to encode the contour, and it is the information that the HOG descriptor is mostly focusing on. Local texture can be harmful, e.g., the paintings on a person's T-shirt may confuse a human detector. In Fig. 1b, we compute the Sobel gradient of each pixel in Fig. 1a and replace a pixel with the gradient value (normalized to $[0\ 255]$). The Sobel image smooths high frequency local texture information, and the remaining contour in Fig. 1b clearly indicates the location of a human.

Fig. 6 in [4] also indicated that image blocks related to the human contour are important in the HOG detector. However, we do not know clearly what information captured by HOG makes it successful in human detection. Thus, we try to experimentally show that contour is the important information captured by HOG. We used the original HOG detector in [4], but tested on the Sobel version of test images of the INRIA dataset.[1] The original HOG SVM detector was trained with features where contour and other information (e.g., fine-scale textures on the clothes) are interwoven with each other (cf. Fig. 1a). It is unusual that without modification it will detect humans on Sobel testing images where contour is the main information (cf. Fig. 1b). Surprisingly, although training and testing images are *different in nature*, the detection accuracy is 67% at 1 FPPI, higher than 7 out of 13 methods evaluated in [16]. Thus, our conjecture is that contour is the key information captured by HOG for pedestrian.

Hypothesis 2: Signs of comparisons among neighboring pixels are key to encode the contour. The community usually use image gradients to detect contours, which are computed by comparing neighboring pixels. We suggest that the signs of such comparisons are key to encode contours while the magnitudes of comparisons are not as important.

[1]The HOG detector is from http://pascal.inrialpes.fr/soft/olt/. Details of the INRIA dataset are presented in Sec. VI-B.

In order to verify this hypothesis, for a given image I, we want to create a new image I' that retains signs of local comparisons but ignores their magnitudes. In other words, we want to find an image I' such that

$$\operatorname{sgn}\big(I(p_1) - I(p_2)\big) = \operatorname{sgn}\big(I'(p_1) - I'(p_2)\big), \tag{1}$$

for any neighboring pair of pixels p_1 and p_2. An example is shown in Eq. 2.

$$I : \left(\begin{array}{c|c|c} 32 & 2 & 8 \\ \hline 38 & 96 & 64 \end{array}\right), \qquad I' : \left(\begin{array}{c|c|c} 1 & 0 & 1 \\ \hline 2 & 3 & 2 \end{array}\right). \tag{2}$$

Note that the pixel 96 is converted to a value 3, because of the path of comparisons $2 < 32 < 38 < 96$. In other words, although the magnitude of comparisons in I are ignored in I', the spatial relationships among multiple comparisons in I will enforce a "pseudo-magnitude" in I'. Another important observation is that gradients computed from I and I' will have quite different magnitudes. Fig. 1c shows such a sign comparison image I' (in which pixel values are scaled to [0 255]) when I is Fig. 1b. We can easily detect the human contour in Fig. 1c.

We can further support hypothesis 2 using human detection experiments. Applying the original HOG detector to sign comparison testing images (like Fig. 1c) of the INRIA dataset, we achieved 61% detection accuracy at 1 FPPI (better than 6 out of 13 methods evaluated in [16]).

Although we observe lower detection rates when the Sobel images or the sign comparison images are used as test images, it is important to note that the classifier was trained using the original images (that is, training and testing images are different in their characteristics.) The fact that we obtain higher accuracies than many existing detectors without modifying the HOG detector is noteworthy.

Thus, we conjecture that the key information for human and many other object detection is the global contour information, and the signs of comparisons among neighboring pixels are the key to capture useful information in the contour.

B. The CENTRIST visual descriptor

We then propose to use the CENTRIST visual descriptor [9] to detect objects, because it succinctly encodes the "signs of neighboring comparisons" information. We will compare CENTRIST with other popular descriptors in Sec. III-C.

Census Transform (CT) is originally designed for establishing correspondence between local patches [25]. Census transform compares the intensity value of a pixel with its eight neighboring pixels, as illustrated in Eq. 3.

$$\begin{array}{c|c|c} 32 & 64 & 96 \\ \hline 32 & \mathbf{64} & 96 \\ \hline 32 & 32 & 96 \end{array} \Rightarrow \begin{array}{ccc} 1 & 1 & 0 \\ 1 & & 0 \\ 1 & 1 & 0 \end{array} \Rightarrow (11010110)_2 \Rightarrow \text{CT} = 214. \tag{3}$$

If the center pixel is bigger than (or equal to) one of its neighbors, a bit 1 is set in the corresponding location. Otherwise a bit 0 is set. The eight bits generated from intensity comparisons can be put together in any order (we collect bits from left to right, and top to bottom), which is consequently converted to a base-10 number in $[0\ 255]$. This is the CT value for the center pixel. The CT image C of an input image I is generated by replacing a pixel with its CT value. The CENTRIST descriptor is a histogram with 256 bins, which is a histogram of these CT values in an entire image or a rectangular region in an image [9].

As shown in Eq. 3, CT values succinctly encode the signs of comparisons between neighboring pixels. What seems to be missing from CENTRIST, however, is the power to capture global (or larger scale) structures and contours beyond the small 3×3 range.

More importantly, if we are given an image I with CENTRIST $\boldsymbol{f}$, then among the small number of images I' that has a matching CENTRIST descriptor, we expect that I' will be similar to I, especially in terms of global structure or contour, which we illustrate in Fig. 2. Fig. 2a shows a 108×36 human contour. We divide this image into 12×4 blocks, thus each block has 81 pixels. For each block I, we want to find an image I' that has the same pixel intensity histogram and CENTRIST descriptor as I.[2] As shown in Fig. 2b, the reconstructed image is similar to the original image. The global characteristic of the human contour is well preserved in spite of errors in the left part of the image.

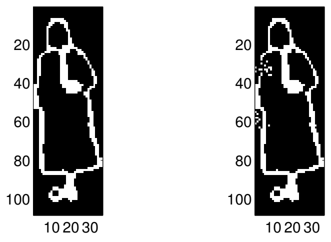
The fact that CENTRIST not only encodes important information (signs of local comparisons) but also implicitly encodes the global contour encourages us to use it as a suitable representation for object detection.

C. Comparing CENTRIST with HOG and LBP

Now we will compare CENTRIST with HOG and LBP, two visual descriptors that are popular in object detection.

For classification tasks, the feature vectors of examples in the same class should be similar to each other, while examples in different classes should have dissimilar descriptors. For any example $\boldsymbol{x}$, we will

[2]We choose to work with small blocks with 81 pixels and binary images to make simulated annealing converge in a reasonable amount of time. Please refer to [9] for details of the reconstruction algorithm.



(a) human contour (b) reconstruction

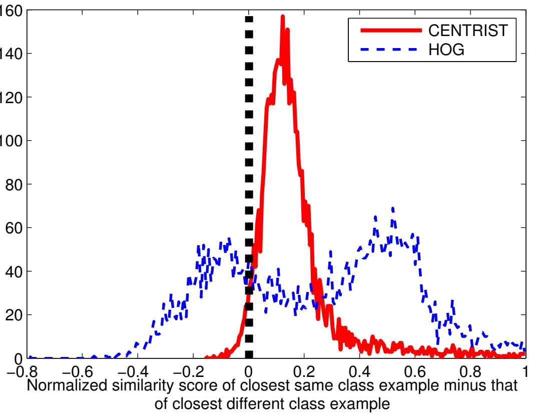
Fig. 2. Reconstruct human contour from CENTRIST.

compute the similarity score between $\boldsymbol{x}$ and all other examples. Let $\boldsymbol{x}_{\text{in}}$ be the most similar example to $\boldsymbol{x}$ within the same class. Similarly, let $\boldsymbol{x}_{\text{out}}$ be the most similar example that is in a different class (e.g., background). Obviously we want $s_{\text{NN}} = s(\boldsymbol{x}, \boldsymbol{x}_{\text{in}}) - s(\boldsymbol{x}, \boldsymbol{x}_{\text{out}})$ to be positive and large, where $s(\boldsymbol{x}, \boldsymbol{y})$ is the similarity score between $\boldsymbol{x}$ and $\boldsymbol{y}$. A positive s_{NN} means that $\boldsymbol{x}$ is correctly classified by a nearest neighbor (1-NN) rule. Thus, s_{NN} is an intuitive and easy-to-compute measure to determine whether a descriptor suits certain tasks.

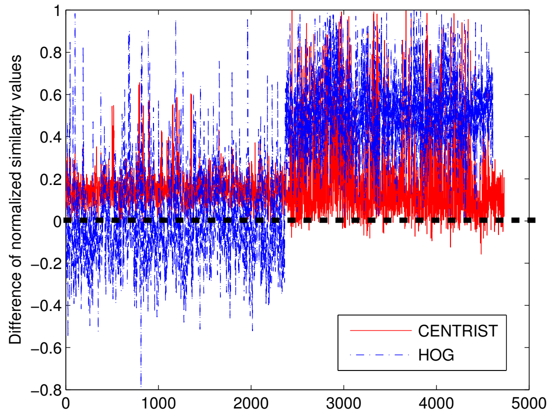
Fig. 3 compares CENTRIST (on Sobel images) and HOG (on original input images) using the INRIA human detection dataset [4]. In Fig. 3a we use all the 2416 human examples, and randomly generate 2 non-human examples from each negative training image which leads to 2436 non-human examples. Fig. 3a shows the distribution (histogram) of s_{NN} for CENTRIST and HOG. A negative s_{NN} (i.e., in the left side of the black dashed line) is an error of 1-NN classifier. It is obvious that the CENTRIST curve resides almost entirely in the correct side (2.9% 1-NN error), while a large portion of the HOG curve is wrong (28.6% 1-NN error). Fig. 3b plots the s_{NN} values for CENTRIST and HOG, and further shows that HOG errors (i.e., any value that is below the black dashed line) are mostly in the first half of the dataset, which are human examples.[3]

It is argued in [9] that visual descriptors such as HOG or SIFT [26] pays more attention to detailed

[3]We provide more experimental details here. HOG feature vectors are extracted using the source code from [4], with its default settings (so HOG feature vectors are normalized using the L2-Hys strategy.) s_{NN} values are normalized to the range $[-1\ 1]$, by dividing the maximum $|s_{\text{NN}}|$ value of all 4852 examples of a given descriptor (HOG or CENTRIST). We tried both the dot product kernel and the histogram intersection kernel as the similarity metric. Fig. 3 shows the best s_{NN} result for each descriptor: by using dot product for HOG and HIK for CENTRIST. If we use HIK for HOG, the 1-NN error is very high (46%). A possible reason is that HOG encodes the magnitudes of gradients, which expects unequal ℓ_1 or ℓ_2 norm of the vectors. However, normalized versions of such vectors are not suitable for HIK. If we use the Laplace kernel $\exp(-|x-y|)$ on HOG, the 1-NN error of HOG can be reduced to 23.3%.



(a) Histogram of s_{NN}



(b) Plot of s_{NN}

Fig. 3. Histogram and plot of similarity score differences (best viewed in color).

local textural information instead of structural properties (e.g., contour) of an image. We further speculate that this is due to the fact that the magnitudes of local comparisons used in HOG pay more attention to local textures. It is also obvious that we can not reconstruct an image from its HOG or SIFT descriptor.

CENTRIST has close relationship with LBP, another popular feature for object detection. If we switch all bits '1' to '0' and vice versa in Eq. 3, the revised formula is an intermediate step to compute the LBP value for the same 3×3 region [27]. However, the more important difference is how the LBP values are utilized. Object detection methods use "uniform LBP" [28], [7], in which certain LBP values that are called "non-uniform" are lumped together. We are, however, not able to reconstruct the global contour because the non-uniform values are missing. In addition, [28] and [7] involve interpolation of pixel intensities. These procedures make their descriptors to encode a blurred version of the signs of neighboring pixel comparisons. We computed the distribution of s_{NN} for the uniform LBP descriptor. It had an error rate of 6.4% for the 1-NN classifier (using HIK, which outperforms the dot product kernel in this experiment), more than twice of the error rate for CENTRIST (2.9%). However, LBP has better s_{NN} distribution than HOG (28.6% 1-NN error). Our conjecture is that the incomplete and blurred local sign information in LBP is still less sensitive than HOG in the presence of noise and distractions from local textures.

We trained linear SVM classifiers for the CENTRIST and uniform LBP dataset for further comparison.[4] The Fisher separation criterion applied to the SVM decision values can also provide a way to compare these two descriptors in this context:

$$\frac{(m_+ - m_-)^2}{\sigma_+^2 + \sigma_-^2}, \tag{4}$$

where $m_\pm$ and $\sigma_\pm^2$ are the mean and variance of the SVM decision value on positive (pedestrian) and negative (background) examples, respectively. The Fisher criterion value for CENTRIST is 2.03, which is also better than the value for uniform LBP (1.62).

Finally, we want to add a note of caution that designing visual feature is an integral and complex task. In different designs, various types of information may play different roles other than their roles in CENTRIST. For example, experiments in [4] showed that "unsigned" HOG has slightly higher accuracy than the "signed" HOG descriptors. In other words, when the magnitude information is used in HOG, merging the 18 directional voting bins of HOG ("signed") into 9 bins without direction information ("unsigned") is useful.

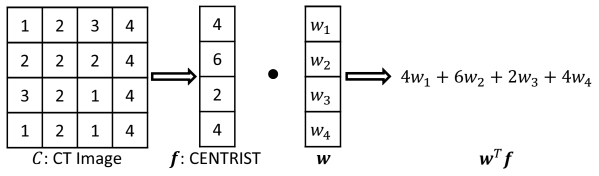
IV. Linear classifiers for fast rejection

One additional virtue of using the CENTRIST feature for object detection is that: when CENTRIST is used together with a linear classifier to classify an image patch as the object of interest or the background, we can design algorithms that are extremely efficient. Specifically, only a constant number of operations are needed (i.e., $O(1)$ complexity) per pixel. Before presenting the full details of C^4, we first introduce these algorithms.

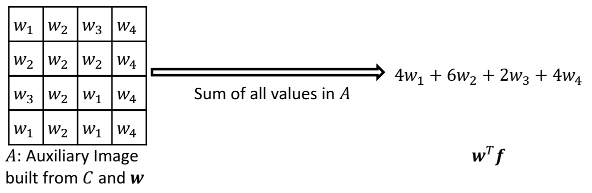
A. Fast rejection for the holistic case

Given an image I, its corresponding Sobel image S, and CT image (of S) C, suppose an image patch (detection window) is represented by a CENTRIST vector $\boldsymbol{f} \in \mathbb{R}^{256}$ extracted from this patch. If we

[4] In order to avoid overfitting, for each dataset we trained two linear classifiers through two-fold cross validation (CV). The CV procedure produced an SVM decision value for every example.



(a) By computing $\boldsymbol{f}$



(b) The proposed method: avoid computing $\boldsymbol{f}$

Fig. 4. Illustration of ways to compute $\boldsymbol{w}^T\boldsymbol{f}$. For simplicity, we assume only 4 different CT values.

have already trained a linear classifier $\boldsymbol{w} \in \mathbb{R}^{256}$, an image patch is classified as an object of interest if and only if $\boldsymbol{w}^T\boldsymbol{f} \geq \theta$. Inspired by [24], we propose an algorithm to compute $\boldsymbol{w}^T\boldsymbol{f}$ using a fixed number of machine instructions for each image patch.

If there are k pixels in the CT image's detection window that have value i, the CENTRIST histogram will satisfy $f_i = k$. Furthermore, all such pixels will contribute $w_i f_i = kw_i$ to $\boldsymbol{w}^T\boldsymbol{f}$, since $\boldsymbol{w}^T\boldsymbol{f} = \sum_{i=1}^{256} w_i f_i$. If we distribute this contribution to every such pixel, that is, assign a value w_i to every pixel with CT value i, all these k pixels will sum up to the same value kw_i.

Let us build *an auxiliary image* A which directly combines the CT image C and the classifier $\boldsymbol{w}$. The auxiliary image has the same size as the CT image, and is defined as:

$$A(x, y) = w_{C(x,y)}, \tag{5}$$

that is, if a pixel has CT value i, we assign a value w_i to the auxiliary image. According to the above analysis, if we sum the assigned values in the auxiliary image in the entire detection window, the summation result is exactly equal to $\boldsymbol{w}^T\boldsymbol{f}$, the term we want to compute. Note that with the auxiliary image A, we do *not* need to compute the CENTRIST vector $\boldsymbol{f}$. Fig. 4 illustrates how to compute $\boldsymbol{w}^T\boldsymbol{f}$ by computing $\boldsymbol{f}$ (Fig. 4a), or by avoiding the computation of $\boldsymbol{f}$ (Fig. 4b).

Thus, computing $\boldsymbol{w}^T\boldsymbol{f}$ is equivalent to summing the values in a rectangular region. It is now a classic result that if we build an integral image of the auxiliary image A, it only requires 4 memory access and 3 summations or subtractions to find the sum of values in any rectangular region [1]. Furthermore, building the integral image requires only 2 memory access and 2 summations per pixel location. In summary, applying a linear classifier for object detection with CENTRIST requires only a small constant amount of machine instructions per detection window.

It is worth noting that the proposed algorithm is inspired by and similar to the method in [24]. In [24], pixel-wise contribution is summed at sparse locations to compute a bound for a set of linear classifiers; while we use this strategy to sum up values densely at all pixel locations to compute the decision value of a single linear classifier. The advantage of using CENTRIST is that it does not require normalization of the feature vector, thus we can distribute the contribution to every pixel *without explicitly generating the CENTRIST vector* $\boldsymbol{f}$. In contrast, normalization is essential in the HOG-based detectors [4].

A linear classifier may not be powerful enough to produce accurate detections alone. However, because of its efficiency, we can learn a linear classifier to quickly reject most of (e.g. 99%) the detection windows that are not objects, while keeping most of the objects of interest for further scrutiny.

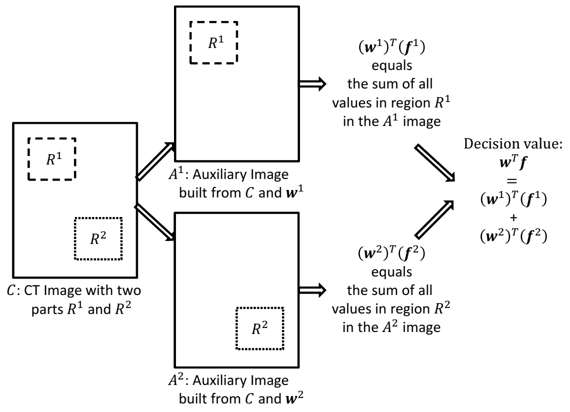
B. Fast rejection for objects with multiple parts

In the above, we consider the detection window as a whole, and extract a single holistic CENTRIST from it. However, a holistic representation is usually lacking in representational power, and an object is better encoded by using multiple parts. In this section, we show that the efficiency of linear classifiers is maintained when the object of interest contains multiple parts.

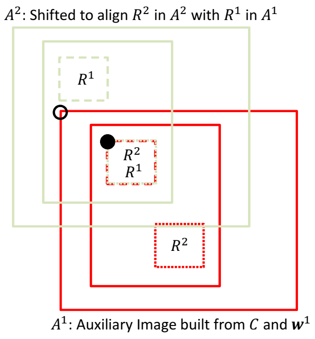
Suppose the object of interest contains K parts, specified by rigid rectangular regions $R^1, \ldots, R^K$. We extract a CENTRIST from each of them. Thus, the object is now represented as $\boldsymbol{f} = [\boldsymbol{f}^1, \ldots, \boldsymbol{f}^K]$, where $\boldsymbol{f}^i$ is a CENTRIST vector extracted from R^i, $\boldsymbol{f}^i \in \mathbb{R}^{256}$, and $\boldsymbol{f} \in \mathbb{R}^{256K}$. Correspondingly, a linear classifier has decision boundary $\boldsymbol{w} = [\boldsymbol{w}^1, \ldots, \boldsymbol{w}^K] \in \mathbb{R}^{256K}$, and the decision value is $\boldsymbol{w}^T \boldsymbol{f} = \sum_{i=1}^K (\boldsymbol{w}^i)^T (\boldsymbol{f}^i)$. Fig. 5a illustrates an example with $K = 2$ parts.

A straightforward way to compute $\boldsymbol{w}^T \boldsymbol{f}$ is similar to the method for ESS with spatial pyramid matching in [24]. We can generate K auxiliary images $A^1, \ldots, A^K$, where A^i is generated by combining the CT image C with part of the classifier $\boldsymbol{w}^i$, which is corresponding to the i-th part, in the same way as in the holistic case. That is, $A^i(x, y) = w^i_{C(x,y)}$, where $C(x, y)$ is the CT value, $\boldsymbol{w}^i \in \mathbb{R}^{256}$, and $w^i_{C(x,y)}$ is the entry in $\boldsymbol{w}^i$ corresponding to this CT value. In A^i, pixels outside of R^i are set to 0.

By summing the values in region R^i of the auxiliary image A^i respectively for $1 \leq i \leq K$ into K sums, $\boldsymbol{w}^T \boldsymbol{f}$ is simply the sum of all the K partial sums. This strategy is illustrated in Fig. 5a. Using this strategy, K auxiliary images (and their integral images) are needed. For a detection window, $4K$ memory access and $4K$ summations or subtractions are needed to compute the partial sums, and $K - 1$ additional summations are required for the final result. When K is big, a large number of instructions


R^1
R^2
C: CT Image with two parts R^1 and R^2
A^1: Auxiliary Image built from C and $\boldsymbol{w}^1$
A^2: Auxiliary Image built from C and $\boldsymbol{w}^2$
$(\boldsymbol{w}^1)^T(\boldsymbol{f}^1)$ equals the sum of all values in region R^1 in the A^1 image
$(\boldsymbol{w}^2)^T(\boldsymbol{f}^2)$ equals the sum of all values in region R^2 in the A^2 image
Decision value: $\boldsymbol{w}^T\boldsymbol{f}$ = $(\boldsymbol{w}^1)^T(\boldsymbol{f}^1)$ + $(\boldsymbol{w}^2)^T(\boldsymbol{f}^2)$

(a) Method similar to that in [24] using K auxiliary images


A^2: Shifted to align R^2 in A^2 with R^1 in A^1
R^1
R^2
R^1
R^2
A^1: Auxiliary Image built from C and $\boldsymbol{w}^1$

(b) The proposed method: shift to align auxiliary images, and add into a single one

Fig. 5. Illustration of ways to compute $\boldsymbol{w}^T\boldsymbol{f}$ with $K = 2$ parts.

are needed per detection window.

We propose a new strategy, illustrated in Fig. 5b, to reduce the complexity. In Fig. 5b, we show the entire auxiliary image (same size as the CT image C), an detection window inside it, and two parts within the detection window. After building the auxiliary images A^i, except for A^1, all other A^i are shifted such that the top-left corner of the region R_i $(i > 1)$ in A^i is shifted to the same position as that of R^1 in A^1 (cf. Fig. 5b, the filled circle.) The part of shifted A^i that is overlapping with A^1 (that is, starting from the hollow circle in Fig. 5b) is added to A^1 in a pixel-wise manner. After all A^i $(i > 1)$ are added to A^1, the new A^1 is a sum of all properly shifted auxiliary images (according to offsets of the parts R^i). Then, comparing Fig. 5a and Fig. 5b, it is also obvious that $\boldsymbol{w}^T\boldsymbol{f}$ is simply the sum of values in the detection window inside the new A^1.

Thus, we first build a grand auxiliary image A (which is exactly the new A^1 as described above) and its integral image, then $\boldsymbol{w}^T\boldsymbol{f}$ requires only 4 memory access and 3 summations or subtractions, which is fewer than $1/K$ of that of the method with K auxiliary images.

It is worthwhile to note that although Fig. 5 shows two non-overlapping, equal-size parts, in reality the parts can overlap with each other, and can have different sizes.

C. Organization of parts in C^4

In our C^4 detection framework, we use multiple rigid parts to detect objects. The parts are organized following [4]. Suppose a detection window is of fixed size $h \times w$, we first divide the detection window into $n_x \times n_y$ non-overlapping equal-size blocks. Every block has size $(h_s, w_s) = (h/n_x, w/n_y)$. Then, any adjacent 2×2 blocks are combined into a superblock. There are in total $(n_x - 1) \times (n_y - 1)$ superblocks that may overlap with each other, and each superblock has size $(2h/n_x, 2w/n_y)$. We use a superblock as a part. Thus, the C^4 framework contains $(n_x - 1) \times (n_y - 1)$ parts in a detection window.

We then create auxiliary images $A^{i,j}$ for $1 \le i \le n_x - 1$, $1 \le j \le n_y - 1$ with the same size as the input image I. Note that now we use a pair (i, j) to index a part or an auxiliary image. The (x, y) pixel of $A^{i,j}$ is set to $A^{i,j}_{x,y} = w^{i,j}_{C(x,y)}$ as described above.

The grand auxiliary image A is then defined as:

$$A(x, y) = \sum_{i=1}^{n_x-1} \sum_{j=1}^{n_y-1} w^{i,j}_{C((i-1)h_s+x,(j-1)w_s+y)}. \tag{6}$$

Note that $(i - 1)h_s$ and $(j - 1)w_s$ in Eq. 6 are the offsets between the part $R^{i,j}$ and the first part $R^{1,1}$, which shift $A^{i,j}$ to properly align with $A^{1,1}$. Then, a detection window with top-left corner (t, l) has a decision value

$$\boldsymbol{w}^T\boldsymbol{f} = \sum_{x=2}^{2h_s-1} \sum_{y=2}^{2w_s-1} A(t + x, l + y). \tag{7}$$

Note that because the Census Transform is not defined for border pixels in a rectangle, for a size $(2h_s, 2w_s)$ part, we only sum in the region $2 \le x \le 2h_s - 1$, $2 \le y \le 2h_w - 1$.

Only one grand integral image is needed to compute Eq. 7, which saves not only large storage space but also computation time. In practice, Eq. 7 runs about 3 to 4 times faster than using one auxiliary image for each part. Also note that the proposed technique is general, and can be used to accelerate other related computations, e.g., ESS with spatial pyramid matching in [24].

One final note is about the grand auxiliary image A. Based on Eq. 6, we only need the CT image C and the classifier $\boldsymbol{w}$, and do not need to compute the separate auxiliary images $A^{i,j}$. In practice, generating A using Eq. 6 is about twice faster than computing all auxiliary images $A^{i,j}$.

In summary, the proposed method for linear classifier and CENTRIST does not involve image pre-processing (e.g., smoothing) or feature normalization. The feature extraction component is seamlessly embedded into classifier evaluation and $\boldsymbol{f}$ is not even generated. These properties together contribute to a real-time object detection system.

V. The C^4 Detection Framework

After the representation of a detection window and the classification with a linear classifier (early rejection) are introduced, in this section, we present all the details of the C^4 framework. The framework is first described in Sec. V-A, and Sec. V-B specifies how the two classifiers used in C^4 are trained.

A. Description of the C^4 detection framework

C^4 uses a brute-force search strategy for object detection. That is, C^4 examines all possible detection windows (of size $h \times w$) in the input image, and classifies each detection window as the object of interest or the background. The top-left corners of all possible detection windows form a regular grid with step size g. If faster detection speed (and potentially lower accuracy) is desired, g can be set to a larger value. In order to detect objects bigger than $h \times w$, the input image is successively scaled down by a ratio 0.8, until the image is smaller than $h \times w$. Brute-force search is also performed in every resized version.

Two classifiers are used to classify a detection window. The input image I is converted to a Sobel version S, and the CT image C (of S). By building a grand integral image of C using Eq. 6, the result of linear classifier (which we name as H_{lin}) for each detection window just requires $O(1)$ operations. Note that Eq. 6 requires $O(1)$ operations per detection window, and we only need to fill its values on the grid of top-left corners (that is, only $1/g^2$ values of the grand auxiliary image need to be calculated).

A cascade structure [1] is used in C^4. The linear classifier H_{lin} is very fast, but not powerful enough for accurate object detection. The threshold of H_{lin} is adjusted such that a large portion (e.g., 99%) of objects will be correctly classified as objects by H_{lin}. Any detection window that is considered as background by H_{lin} is immediately rejected, and only the small number of windows that pass the test of H_{lin} will be further considered by a more powerful classifier. Note that no CENTRIST vector is generated at this stage.

Since the CENTRIST representation of a detection window is a natural histogram, the histogram intersection kernel (HIK) will lead to high accuracy [29]. In C^4, the second classifier is an SVM classifier using the non-linear histogram intersection kernel. For the small number of detection windows that pass the test of H_{lin}, the CENTRIST representation is now calculated, and acts as the input for the HIK SVM classifier H_{hik}. An HIK SVM has the same complexity as a linear classifier during the test time [29], which also contributes to the real-time detection speed of C^4. Only if a detection window is classified as object by both H_{lin} and H_{hik}, it is considered as an object of interest.

The last step of C^4 is non-maximal suppression (NMS) or post-processing. In C^4, we use a slightly more aggressive variant of the NMS strategy in [6]. A detected object contains a rectangle (i.e., detected location) and a score (decision value of H_{hik}). First, we sort all detections in the decreasing order of their scores. Then, given any two detections D_i and D_j with $i > j$ (that is, D_i has higher score than D_j), if they have large intersections (intersection region size more than 60% of size of either D_i or D_j), we remove the D_j detection.

Finally, a detection with rectangle R_d and a groundtruth with rectangle R_g is considered as a true match if

$$\frac{\text{Area}(R_d \cap R_g)}{\text{Area}(R_d \cup R_g)} > 0.5. \tag{8}$$

We also require that one groundtruth rectangle can only match to at most one detected window in one input image. These matching criteria are commonly used in object detection [11].

B. Training classifiers

An important part of C^4 is to train accurate classifiers H_{lin} and H_{hik}, which we present in this section.

In the training phase, we have a set of $h \times w$ positive training image patches $\mathcal{P}$ and a set of larger negative images $\mathcal{N}$ that do not contain any object of interest. We first randomly choose a small set of patches from the images in $\mathcal{N}$ to form a negative training set $\mathcal{N}_1$. Using $\mathcal{P} \bigcup \mathcal{N}_1$, we train a linear SVM classifier H_1.

A bootstrap process is used to generate a new negative training set $\mathcal{N}_2$: H_1 is applied to all patches in the images in $\mathcal{N}$, and any detection (i.e., false positives) are added to $\mathcal{N}_2$. We then train H_2 using $\mathcal{P}$ and $\mathcal{N}_2$. This process is repeated until all patches in $\mathcal{N}$ are classified as negative by at least one of $H_1, H_2, \ldots$ We then train a linear SVM classifier using $\mathcal{P}$ and the combined negative set $\bigcup_i \mathcal{N}_i$, which is H_{lin}.

The threshold of H_{lin} is adjusted to classify a large portion of the objects as positive. We then use H_{lin} on $\mathcal{N}$ to bootstrap a new negative training set $\mathcal{N}_{\text{final}}$, and train an SVM classifier using the libHIK HIK SVM solver of [29], which is H_{hik}. In the testing / detection phase, the cascade with two nodes H_{lin} and H_{hik} is used.

Finally, we call the proposed method C^4, as we are detecting objects based on their $\underline{\text{c}}$ontour information using a $\underline{\text{c}}$ascade $\underline{\text{c}}$lassifier and the $\underline{\text{C}}$ENTRIST representation.

VI. Experimental Results

In this section, we empirically show the performance of the C^4 detection framework on 4 different objects detection benchmark datasets: INRIA pedestrian, Caltech pedestrian, face and car. The same C^4 framework is used for detecting different objects, by applying to different training images. There are five parameters in C^4: detection window size h and w, number of blocks in the detection window n_x and n_y, and brute-force searching's grid step size g. For each dataset, the parameters are adjusted to suit its characteristic, e.g., faces and cars have different aspect ratios. The parameter setup will be detailed with each dataset.

We will first present experiments illustrating the detection speed of C^4 in Sec. VI-A using pedestrian detection as an example, followed by reports of C^4's detection accuracy (Sec. VI-B to Sec. VI-E). Finally, we discuss the limitations and drawbacks of the proposed C^4 detection framework in Sec. VI-F, and their possible improvements.

A. Detection speed

1) Running on a desktop computer: It is obvious that C^4's detection speed is independent of the type of objects to detect. Thus, we use pedestrian detection as an example to study its detection speed.

C^4 achieves faster speed than existing pedestrian detectors. On a 640×480 video, its speed is 20.0 fps, using only 1 processing core of a 2.8GHz CPU.[5] As far as we know, this is the fastest detector on CPU, which has a reasonably low false alarm rate and high detection rate. Existing system, in order to achieve real-time detection speed, has to resort to the parallel processing cores of a GPU hardware. Using CPU+GPU, the state-of-the-art detection speed is 50 fps [23], with a Nvidia GeForce GTX 470 (448 cores). The major computations in C^4, including building the grand auxiliary image and its integral image, linear rejection, building CENTRIST for remaining detection windows, H_{hik} testing, resizing images, and NMS, can all be easily adapted to further GPU acceleration. When multiple CPU cores and GPU are used, it is expected that C^4 will run at least 10 to 20 times faster than the current CPU version, according to results on published system [22].

Real-time processing is a must-have property in most object detection applications. Our system is already applicable in some domains, e.g., robot systems. However, there is still huge space for speed improvements, which will make C^4 suitable even for the most demanding applications, e.g., automatic

[5]This is also the default hardware environment of all experiments in this paper, if not otherwise specified.

TABLE I

DISTRIBUTION OF C^4 COMPUTING TIME (IN PERCENTAGE).

Processing module	Percent of used time
Sobel gradients	16.55%
Computing CT values	9.36%
Grand Auxiliary & Integral Image	44.65%
Resizing image	5.68%
Brute-force scan	23.75%
Post-processing	0.02%

driver assistance. Table I is the break-down of time spent in different components of C^4. Most of these components are very friendly to acceleration using special hardware (e.g., GPU).

The fact that we do not need to explicitly construct feature vectors for H_{lin} is not the only factor that makes our system extremely fast. H_{lin} is also a powerful classifier. It filters away about 99.43% of the candidate detection windows, only less than 0.6% patches require attentions of the expensive H_{hik} classifier on the INRIA pedestrian dataset.

C^4 runs faster in smaller images. In a 480×360 YouTube video with many pedestrians in most frames, its speed is 36.3 fps. Its speed is 109 fps on 320×240 frames.

2) Running on a Robot: We integrated the C^4 pedestrian detection algorithm onto an iRobot PackBot in order to achieve on-board pedestrian detection and to enable pedestrian following. The implementation first captured images from a TYZX G2 stereo camera system and then processed the imagery using an Intel 1.2 GHz Core 2 Duo embedded in an add-on computational payload. We used the raw camera imagery to perform the detection and used the stereo range data to estimate the distance to the pedestrian. Note that it is not feasible to use GPU processing on board a robot.

From the stereo data we used RANSAC to estimate a ground plane, and we sampled the depths along the ground plane's horizon. With the depth and coordinates of the plane we can calculate a box that would contain a pedestrian standing on the plane at the given position on the horizon and given distance. This gives us far fewer windows to test with the detector. Note that the C^4 detector was tailored to work with the robot to a 3-layer cascade instead of 2-layer for faster speed (but less accurate).

The combined approach is at approximately 20 frames per second (50 milliseconds) on the embedded Intel 1.2 GHz Core 2 Duo. Alone, the C^4 runs at approximately 8 frames per second (120 milliseconds)

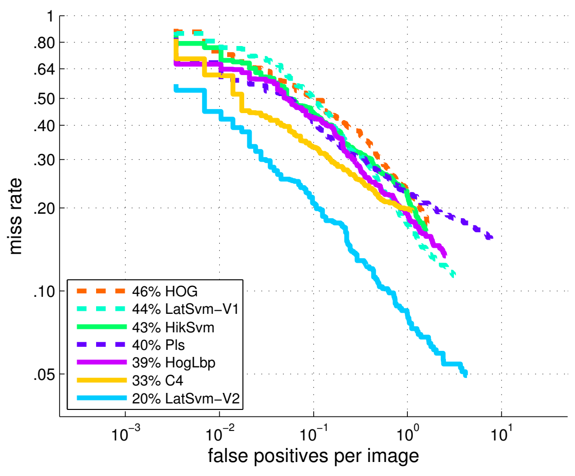


Fig. 6. Pedestrian detection results on the INRIA dataset (best viewed in color).

on the same hardware.

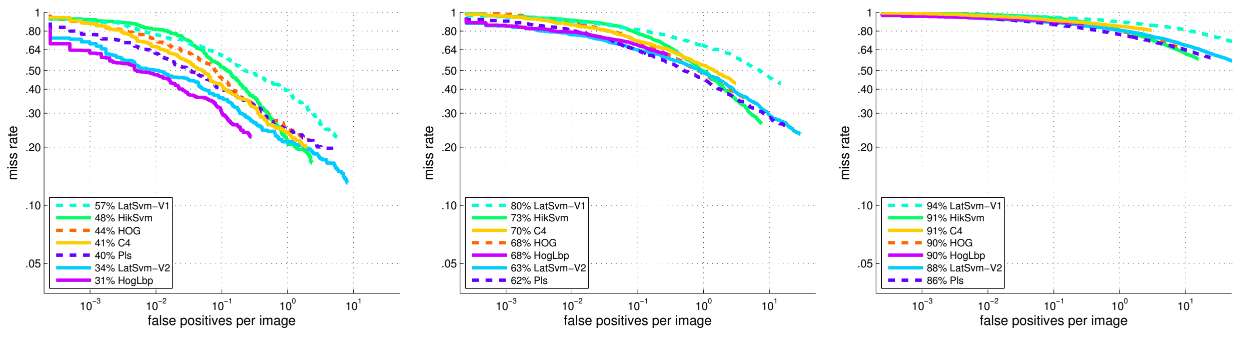
B. Detecting pedestrians (INRIA benchmark)

For pedestrian detection, we first experiment on the INRIA dataset [4]. There are 2416 positive training image patches and 1218 negative training images for bootstrapping in the INRIA dataset. This dataset contains large pedestrian images: 90% of the pedestrians in it are taller than 139 pixels [11]. The training positive patches are 128×64, but we crop the examples to 108×48 pixels, which allows a reasonable sized margin on all sides around a pedestrian. The C^4 parameters are $h = 108$, $w = 48$, $n_x = 12$, $n_y = 6$, and $g = 3$. The entire C^4 training process took 530 seconds (less than 9 minutes) on this dataset.

During testing time, we use the converted test frames and evaluation code in [11] to calculate the detection performance. Fig. 6 compares the performance of C^4 with a few state-of-the-art detectors. Note that Fig. 6 only lists those detectors with their features extracted from a single grayscale frame. Detectors using other information (such as color and motion) can achieve better accuracy, which we will further discuss in Sec. VI-C. [6]

Besides the detection accuracy curves, Fig. 6 also shows the log-average miss rate of each method, which is the "averaging miss rate at nine FPPI rates evenly spaced in log-space in the range 10^{-2} to

[6]Results of these detectors in both Fig. 6 and Fig. 7 are downloaded from http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/, from where the details about every detector can also be found. The PLS method used a 3-bin color histogram, but Fig. 4 in [15] showed that color (although useful) is not the major working force in this method.



(a) NEAR (b) REASONABLE (c) ALL

Fig. 7. Pedestrian detection results on the INRIA dataset (best viewed in color).

10^0" [11] (the smaller this value the better a detector is). C^4 outperforms HOG (which uses the HOG descriptor), HIKSVM (which uses a multi-level version of HOG), PLS (which uses HOG, color frequency histogram, and texture), and HOGLBP (which uses HOG and LBP histogram). This result verifies the usefulness of the CENTRIST descriptor to pedestrian detection.

C^4's detection accuracy is lower than that of LATSVM-V2 (the part-based latent SVM detector in [6] which uses PCA of HOG features), which suggests that a deformable part-based approach may be used in C^4 to seek better detection results in the future. One advantage of C^4 is that it strives a balance between detection accuracy and speed: C^4 has real-time detection speed; however, in the evaluation of [11], LATSVM-V2 has a speed of 0.629 fps; and other detectors in Fig. 6 ranged from 0.062 fps (HOGLBP) to 0.239 fps (HOG).

C. Detecting pedestrians (Caltech benchmark)

We further experiment C^4 pedestrian detection on the Caltech pedestrian detection dataset, which is more challenging than the INRIA dataset. It contains 128k training video frames (with 640×480 pixels), which contains 192k pedestrians. The testing data contains 121k video frames (640×480 pixels) and 155k pedestrians. The pedestrians in this dataset are low resolution ones: half of them have heights lower than 48 pixels [11], which makes this dataset challenging.

The same C^4 detector as the one used in Sec. VI-B is used for this dataset. We test on all videos (including both training and testing videos) and follow the standard testing protocol of the Caltech dataset. Several subsets of tests are defined in this dataset, according to different conditions (e.g., pedestrian heights). C^4 detection results are shown in Fig. 7, which is generated by the evaluation code provided by this dataset. Fig. 7a shows results on the NEAR subset. This subset include pedestrians whose heights are larger than 80 pixels; and in the C^4 detector we not only scales a testing image down in order to detect larger pedestrians, but also scale it up by $0.8^{-1} = 1.25$ to detect pedestrians shorter than 108 pixels. Fig. 7b contains results for the subset of pedestrians which are 50 pixels or taller (the REASONABLE subset), and Fig. 7c are results for all pedestrians in the Caltech dataset (the ALL subset). In the latter two subsets, images are scaled up by 0.8^{-2} in C^4 to detect smaller pedestrians.

C^4 outperforms LATSVM-V1 and HIKSVM in all cases. HOG has mixed results compared to C^4: C^4 wins in the NEAR subset while HOG takes over in the REASONABLE and ALL subsets. However, C^4 still maintains the advantage in testing speed. In the REASONABLE subset, C^4 detects at 6.6 fps speed; while the fastest method evaluated in [11] is 2.67 fps.

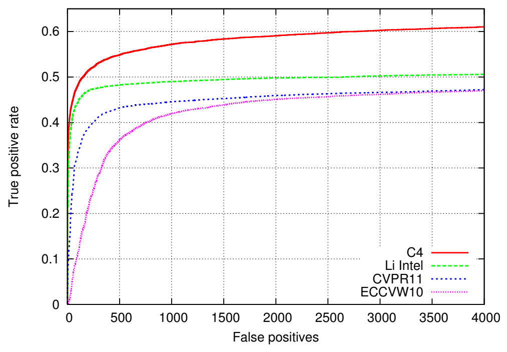
One common trend in Fig. 7 is that all detectors' accuracy rates degenerate when pedestrians become smaller. However, when we compare Fig. 6 and the sub-figures in Fig. 7, it is obvious that both HOGLBP and PLS improved their accuracy rank among the methods when pedestrians become smaller; while LATSVM-V2 gradually loses its advantage over other methods. Our interpretation to this phenomenon is that when the pedestrians become smaller, using more information channels is a good boost to the detectors' accuracy: HOGLBP adds LBP to HOG and PLS adds color and texture. The color and texture information seems more effective since PLS is the most accurate method in Fig. 7c, although its accuracy is only in the middle of Fig. 6. Thus, adding more information channels to C^4 seems a promising way to improve it for low resolution pedestrian detection applications.

In fact, if we do not restrict ourselves to application scenarios where only grayscale images are available, C^4 could be further improved by integrating other information such as color [16], [15], 3D [23], and motion [30].

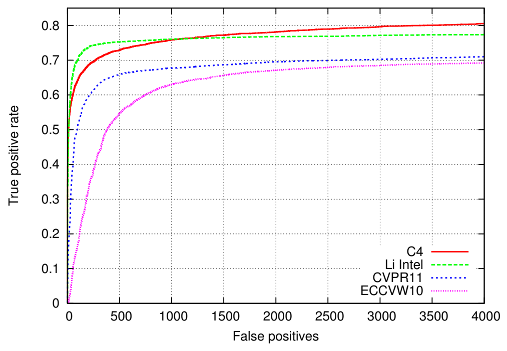
D. Detecting Faces

Next, we test C^4 on face detection tasks. The test set is FDDB, a new multi-pose face detection benchmark [31]. The FDDB dataset has 2845 images with 5171 faces taken under various conditions (illumination, pose, resolution, size, etc.) While training the C^4 detector, a subset of images (EXPRESSION and LIGHTS) from the CMU PIE database [32] are used. For every positive example, its left-right flipped image is also used. Furthermore, a resized version is included (i.e., every positive training example is expanded to three. For details about resizing the images, please refer to [33].) The C^4 parameters are $h = w = 30$, $n_x = n_y = 6$, $g = 2$. The entire C^4 training process took 771 seconds on this dataset.

We use the evaluation groundtruth, protocol (10 fold cross validation), and software provided by FDDB to generate two kinds of scores: discrete and continuous, as shown in Fig. 8. In the discrete case, C^4



(a) Continuous score



(b) Discrete score

Fig. 8. Face detection results on the FDDB dataset (best viewed in color).

outputs a binary decision (face or not) using Eq. 8, and the discrete score measures the commonly used number of false positives vs. true positive rate, as shown in Fig. 8b. The continuous score in Fig. 8a, however, takes into account the ratio of overlap between C^4's output and the groundtruth annotation.

In Fig. 8, we compare C^4 with previous state-of-the-art on the FDDB dataset.[7] The continuous score of C^4 is consistently higher than the compared methods. However, the comparison is more complicated in the discrete case. C^4 consistently outperforms the CVPR11 method [31] and the ECCVW10 method. When comparing with the LI INTEL method [33], C^4 has an edge when there are only very few false positives, while LI INTEL is the winner when there are roughly 50–1100 false positives; finally, C^4 wins again with $>$1100 false positives.

The difference in relative performance between C^4 and the LI INTEL method is due to the different metrics. Our conjecture is that: when consider the quality of detected faces, these two methods are

[7]Available at http://vis-www.cs.umass.edu/fddb/results.html. This page also contains more details of the methods compared in Fig. 8.

comparable to each other. However, the higher continuous score of C^4 indicates that the detected bounding boxes of C^4 overlaps more with the groundtruth. That is, C^4's localization of faces is more accurate. If the threshold in Eq. 8 is raised to a higher overlapping ratio, C^4 will have better detection accuracy than the LI INTEL method in the discrete case too.

Note that the CMU PIE database contains pictures from only 68 subjects. If a larger multiple pose training set is used, we expect C^4 to achieve higher detection accuracy. It is also worth noting that Fig. 8 only compares C^4 with a few best performing methods on the FDDB dataset. Results of other methods, such as the classic cascade detector by Viola and Jones [1], are also provided by the FDDB dataset's homepage. For simplicity of presentation, we omitted these results, because their scores are significantly lower than those presented in Fig. 8.

E. Detecting cars

Car is another important type of objects to detect, e.g., in surveillance applications. We use the UIUC side view car dataset [34] for both training and testing. We use the positive (car) training images from the UIUC dataset, and the negative images in the INRIA pedestrian dataset [4] as negative training images. Similar to Sec. VI-E, flipped and resized versions of positive examples are also used. The C^4 parameters are $h = 36$, $w = 96$, $n_x = 6$, $n_y = 8$. This dataset has two setup: Single scale (*UIUC-Single*) and multiple scale car images (*UIUC-Multi*). We use grid step size $g = 2$ for *UIUC-Single* and $g = 1$ for *UIUC-Multi*. The entire C^4 training process took 199 and 196 seconds for the two setups on this dataset, respectively.

This dataset also provides software for evaluating various detectors' performance, using the accuracy at a specific point on the ROC curve (EER, when the false positive rate equals the false negative rate) as the evaluation metric. Table II shows the accuracy of C^4 and existing methods.

From Table II, C^4 is slightly inferior to the methods in [24] and [21], but better than other methods. In the *UIUC-Multi* case, C^4 has 4 false detections and 3 missed cars. However, one of the false detection (in the image TEST-95.PGM, top-right of the image) is in fact a positive (car). This car is missed in the annotations of cars in this dataset's groundtruth.

F. Limitations and drawbacks of C^4

As seen in the above experimental results, the major drawback of C^4 is that it produces more false positives in some cases, comparing to the state-of-the-art detector for specific objects. In detecting faces, when C^4 has mixed results with competing methods, C^4 is inferior when the number of false positives

TABLE II

CAR DETECTION ACCURACY (AT EER) ON THE UIUC DATASET

	UIUC-Single	*UIUC-Multi*
C^4	97.5%	97.8%
Leibe et al. [35]	97.5%	95%
Mutch and Lowe [36]	99.9%	90.6%
Lampert et al. [24]	98.5%	98.6%
Gall et al. [21]	98.5%	98.6%

are small (Fig. 8b); but C^4 gradually becomes the winning method, when the number of false positives increases. In the pedestrian case, C^4 also has more false positives than the best pedestrian detectors.

The possible cause is that the classifiers H_{lin} and H_{hik} are not powerful enough to distinguish the false positives from true object patches. Since the major purpose of H_{lin} is to reject most negative patches quickly, we may improve the discrimination power of H_{hik}, or adding a third classifier after it. Currently, although we treat every superblock as a "part", H_{hik} is trained in a way that treats each image patch as a whole. One possible improvement is to discriminatively train these parts as in the LATSVM way [6]. Another possibility is to use a representation that is complementary to CENTRIST, and train a third classifier in the cascade to further remove false detections. For example, when color, 3D, motion or other information sources are available, we expect that C^4 object detection accuracy will be further improved.

The C^4 framework's capacity is also limited by the representational power of the CENTRIST visual descriptor. For example, if some objects are to be distinguished by textures instead of contours, the C^4 framework is expected to have poor detection results, since it is designed to detect object mainly based on the contour information. Similarly, we expect that adding more information channels will help C^4 overcome this difficulty.

VII. CONCLUSIONS

In this paper, we proposed a real-time and accurate object detection framework C^4: detecting objects based on their contour information using a cascade classifier and the CENTRIST representation. C^4 detects various objects using the contour cues, a cascade classifier, and the CENTRIST visual descriptor. Examples of objects that are detected by C^4, including pedestrians, faces and cars, are shown in this paper through experiments.

First, we conjecture that contour is the key information source for pedestrian detection, and the signs of comparisons among neighboring pixels are the key to encode contours. These conjectures are accompanied with supporting experiments.

We then show that CENTRIST [9] is suitable for object detection, because it succinctly encodes the sign information during neighboring pixel comparisons, and is able to capture large scale structures or contours.

A major contribution of this paper is an extremely fast object detection component in the C^4 framework. A linear classifier in the C^4 framework can quickly reject background detection windows, using only a small number of instructions per detection window. Time consuming preprocessing and feature vector normalization are not needed in C^4. Furthermore, we do not need to explicitly generate the CENTRIST feature vectors in the linear rejection stage.

Drawbacks and potential improvements to C^4 are also discussed. In a part-based object detection system, we can use C^4 to detect object parts and enable C^4 to detect object with non-constant aspect ratios. C^4 can also be further accelerated by reducing computations, e.g., accelerating the H_{hik} evaluation by branch and bound [24], [37], or by resizing the features instead of resizing images [23] (thus the grand integral image, whose computation currently covers about half of C^4's running time, is required for only 1 scale).

Acknowledgment

We thank the anonymous reviewers for their useful comments and suggestions. We are grateful for the support of National Natural Science Foundation of China under NSFC NO. 61105043. Part of this work was done when J. Wu was with the Nanyang Technological University, Singapore.

References

[1] P. Viola and M. Jones, “Robust real-time face detection,” *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[2] J. Wu, S. C. Brubaker, M. D. Mullin, and J. M. Rehg, “Fast asymmetric learning for cascade face detection,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 30, no. 3, pp. 369–382, 2008.

[3] C. Huang, H. Ai, Y. Li, and S. Lao, “High-performance rotation invariant multiview face detection,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 29, no. 4, pp. 671–686, 2007.

[4] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proc. IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 886–893.

[5] P. Dollár, C. Wojek, B. Schiele, and P. Perona, “Pedestrian detection: A benchmark,” in *Proc. IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, 2009.

[6] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[7] X. Wang, T. X. Han, and S. Yan, “An HOG-LBP human detector with partial occlusion handling,” in *Proc. IEEE Int’l Conf. on Computer Vision*, 2009.

[8] D. Gerónimo, A. M. López, A. D. Sappa, and T. Graf, “Survey on pedestrian detection for advanced driver assistance systems,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 32, no. 7, pp. 1239–1258, 2010.

[9] J. Wu and J. M. Rehg, “CENTRIST: A visual descriptor for scene categorization,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1489–1501, 2011.

[10] J. Wu, C. Geyer, and J. M. Rehg, “Real-time human detection using contour cues,” in *Proc. IEEE Int’l Conf. Robotics and Automation*, 2011, pp. 860–867.

[11] P. Dollár, C. Wojek, B. Schiele, and P. Perona, “Pedestrian detection: An evaluation of the state of the art,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743–761, 2012.

[12] P. Viola, M. Jones, and D. Snow, “Detecting pedestrians using patterns of motion and appearance,” in *Proc. IEEE Int’l Conf. on Computer Vision*, 2003, pp. 734–741.

[13] B. Wu and R. Nevatia, “Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors,” *International Journal of Computer Vision*, vol. 75, no. 2, pp. 247–266, 2007.

[14] S. Maji and A. C. Berg, “Max-margin additive classifiers for detection,” in *Proc. IEEE Int’l Conf. on Computer Vision*, 2009, pp. 40–47.

[15] W. R. Schwartz, A. Kembhavi, D. Harwood, and L. S. Davis, “Human detection using partial least squares analysis,” in *Proc. IEEE Int’l Conf. on Computer Vision*, 2009.

[16] P. Dollár, Z. Tu, P. Perona, and S. Belongie, “Integral channel features,” in *Proc. British Machine Vision Conference*, 2009.

[17] B. Leibe, E. Seemann, and B. Schiele, “Pedestrian detection in crowded scenes,” in *Proc. IEEE Int’l Conf. on Computer Vision and Pattern Recognition*, vol. I, 2005, pp. 878–885.

[18] J. Wu, W.-C. Tan, and J. M. Rehg, “Efficient and effective visual codebook generation using additive kernels,” *Journal of Machine Learning Research*, vol. 12, pp. 3097–3118, Nov 2011.

[19] S. Paisitkriangkrai, C. Shen, and J. Zhang, “Incremental training of a detector using online sparse eigendecomposition,” *IEEE Trans. on Image Processing*, vol. 20, no. 1, pp. 213–226, 2011.

[20] C. Shen, S. Paisitkriangkrai, and J. Zhang, “Efficiently learning a detection cascade with sparse eigenvectors,” *IEEE Trans. on Image Processing*, vol. 20, pp. 22–35, 2011.

[21] J. Gall, A. Yao, N. Razavi, L. V. Gool, and V. Lempitsky, “Hough forests for object detection, tracking, and action recognition,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 33, no. 11, pp. 2188–2202, 2011.

[22] C. Wojek, S. Walk, and B. Schiele, “Multi-cue onboard pedestrian detection,” in *Proc. IEEE Int’l Conf. on Computer Vision and Pattern Recognition*, 2009.

[23] R. Benenson, M. Mathias, R. Timofte, and L. V. Gool, “Pedestrian detection at 100 frames per second,” in *Proc. IEEE Int’l Conf. on Computer Vision and Pattern Recognition*, 2012, pp. 2903–2910.

[24] C. H. Lampert, M. B. Blaschko, and T. Hofmann, “Efficient subwindow search: A branch and bound framework for object localization,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2129–2142, 2009.

[25] R. Zabih and J. Woodfill, “Non-parametric local transforms for computing visual correspondence,” in *Proc. European Conf. Computer Vision*, vol. 2, 1994, pp. 151–158.

[26] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[27] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[28] Y. Mu, S. Yan, Y. Liu, T. Huang, and B. Zhou, "Discriminative local binary patterns for human detection in personal album," in *Proc. IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, 2008.

[29] J. Wu, "Efficient HIK SVM learning for image classification," *IEEE Trans. on Image Processing*, vol. 21, no. 10, pp. 4442–4453, 2012.

[30] S. Walk, N. Majer, K. Schindler, and B. Schiele, "New features and insights for pedestrian detection," in *Proc. IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, 2010.

[31] V. Jain and E. Learned-Miller, "Online domain adaptation of a pre-trained cascade of classiers," in *Proc. IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, 2011, pp. 577–584.

[32] T. Sim, S. Baker, and M. Bsat, "The CMU pose, illumination, and expression database," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 25, no. 12, pp. 1615–1618, 2003.

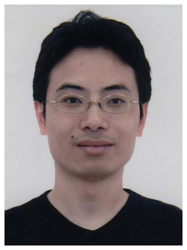
[33] J. Li, T. Wang, and Y. Zhang, "Face detection using SURF cascade," in *Proc. ICCV 2011 BeFIT workshop*, 2011.

[34] S. Agarwal, A. Awan, and D. Roth, "Learning to detect objects in images via a sparse, part-based representation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1475–1490, 2004.

[35] B. Leibe, A. Leonardis, and B. Schiele, "Robust object detection with interleaved categorization and segmentation," *International Journal of Computer Vision*, vol. 77, no. 1-3, pp. 259–289, 2008.

[36] J. Mutch and D. G. Lowe, "Multiclass object recognition with sparse, localized features," in *Proc. IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, 2006, pp. 11–18.

[37] A. Lehmann, P. Gehler, , and L. V. Gool, "Branch&rank: Non-linear object detection," in *British Machine Vision Conference*, September 2011, pp. 8.1–8.11.



Jianxin Wu received his BS and MS degrees in computer science from Nanjing University, and his PhD degree in computer science from the Georgia Institute of Technology. He is currently a professor in the Department of Computer Science and Technology at Nanjing University, China. Previously he was an assistant professor in the Nanyang Technological University, Singapore. His research interests are computer vision and machine learning. He is a member of the IEEE.

Nini Liu received her BS degree in information engineering from Shanghai Jiao Tong University, China, in 2007, and her PhD degree in information engineering from Nanyang Technological University, Singapore, in 2012, respectively. She was a research staff at Nanyang Technological University. Her research interests include biometrics, pattern detection and recognition.

Christopher Geyer is a Senior Principal Research Scientist at iRobot Corporation. Dr. Geyer started his career in computer vision in the GRASP Lab at the University of Pennsylvania, where he received his B.S.E. and Ph.D. in Computer Science in 1999 and 2002, respectively. As a post-doctoral researcher at the University of California, Berkeley from 2002 to 2005, he led a team to develop an autonomous landing capability for unmanned rotorcraft for the U.S. Defense Advanced Research Projects Agency (DARPA). From 2005 to 2008, he lead research in perception for aerial vehicles at Carnegie Mellon University (CMU), and developed technology for sensing and avoiding general aviation aircraft for unmanned aerial vehicles (UAVs). While at CMU, he was also a member of the CMUs Tartan Racing team, the team that won first place in the DARPA Urban Challenge. He joined iRobot in 2008 where he leads research and development in perception for unmanned and robotic systems. His interests include computer vision, robotics, human machine interaction, and autonomy.

James M. Rehg received his PhD degree in Electrical and Computer Engineering from the Carnegie Mellon University. He is a Professor in the College of Computing at the Georgia Institute of Technology. He is a member of the Graphics, Visualization, and Usability Center and co-directs the Computational Perception Lab. His research interests are computer vision, robotics, machine learning, and computer graphics. He is a member of the IEEE.