

# OBJECT DETECTION VIA BOOSTED DEFORMABLE FEATURES

Mohamed Hussein<sup>†</sup>, Fatih Porikli

Mitsubishi Electric Research Labs  
Cambridge, MA 02139

Larry Davis

University of Maryland  
Dept. of Computer Science  
College Park, MD 20742

## ABSTRACT

It is a common practice to model an object for detection tasks as a boosted ensemble of many models built on features of the object. In this context, features are defined as subregions with *fixed* relative locations and extents with respect to the object's image window. We introduce using deformable features with boosted ensembles. A deformable feature adapts its location depending on the visual evidence in order to match the corresponding physical feature. Therefore, deformable features can better handle deformable objects. We empirically show that boosted ensembles of deformable features perform significantly better than boosted ensembles of fixed features for human detection.

**Index Terms**— Human Detection, Boosting, Deformable Features

## 1. INTRODUCTION

Human detection is one of the most challenging tasks in computer vision with a long list of applications in several domains, such as intelligent vehicles, video surveillance, and interactive environments. Approaches for human detection can be categorized based on how the human body is modeled. In one category, a holistic model is used, where the human body is modeled as a whole without being divided into smaller parts [1, 2, 3]. In a second category, a part-based model is used, where models for parts of the body are learnt, possibly along with global constraints that need to be satisfied [4, 5, 6]. Part-based models, in general, deliver better performance than holistic models since they can better handle partial occlusion. However, their drawback is that the number of parts and their locations has to be manually set. A third category of approaches addresses this problem by modeling the body as an ensemble of *features*. A feature in this context is a subregion of an object's image window identified by

\*©COPYRIGHT 2009 IEEE. PUBLISHED IN THE IEEE 2009 INTERNATIONAL CONFERENCE ON IMAGE PROCESSING (ICIP 2009), SCHEDULED FOR NOVEMBER 7-11, 2009 IN CAIRO, EGYPT. PERSONAL USE OF THIS MATERIAL IS PERMITTED. HOWEVER, PERMISSION TO REPRINT/REPUBLISH THIS MATERIAL FOR ADVERTISING OR PROMOTIONAL PURPOSES OR FOR CREATING NEW COLLECTIVE WORKS FOR RESALE OR REDISTRIBUTION TO SERVERS OR LISTS, OR TO REUSE ANY COPYRIGHTED COMPONENT OF THIS WORK IN OTHER WORKS, MUST BE OBTAINED FROM THE IEEE. CONTACT: MANAGER, COPYRIGHTS AND PERMISSIONS / IEEE SERVICE CENTER / 445 HOES LANE / P.O. BOX 1331 / PISCATAWAY, NJ 08855-1331, USA. TELEPHONE: + INTL. 908-562-3966.

<sup>†</sup>The first author is a graduate student at the University of Maryland, and was an intern at Mitsubishi Electric Research Labs while doing most of this work.



**Fig. 1.** An illustration of the desired behavior of a deformable feature for the head. The feature's initial location is marked by a dotted rectangle and the desired final location is marked with a solid rectangle. Notice how the initial location is often not aligned with the actual location of the physical feature (head).

its relative position and size. Typically, boosting techniques [7] are used to select the best features among all possibilities [8, 9, 10].

The common drawback of most part and feature based models is the difficulty of handling deformation since feature and part locations are fixed. However, in deformable objects, physical parts/features are hardly fixed in location. Consider for example the head part/feature in the human images in Figure 1, from the INRIA Person dataset [3]. Felzenszwalb et al. [11] proposed deformable part models to handle this problem. However, as other part models, this work lacks the flexibility of automatically determining the number, locations, and sizes of parts. In this paper, we introduce deformable features, instead of deformable parts, to be used in boosting ensembles. To the best of our knowledge, our work introduces deformation to feature based models for the first time.

The rest of the paper is organized as follows: Section 2 introduces deformable features. Boosting of deformable features is explained in Section 3. Details of our implementation and experimental results are provided in Sections 4 and 5. Finally, the paper is concluded in Section 6.

## 2. DEFORMABLE FEATURES

In the context of feature-based models for object detection, we define a deformable feature (d-feature) as a feature that is not bound to a fixed location in the object's model. Rather it can move (translate) in a small neighborhood around a central location. We would like a d-feature to be able to locate the physical feature it represents within this neighborhood. Figure 1 illustrates the desired behavior of a d-feature that represents the head of a human. Starting from an initial (typical) location for the physical feature, illustrated as a dotted rectangle, the feature moves to a better location to capture the

```

procedure DEFREFINE( $\mathbf{F}, \mathcal{X}$ )
   $\triangleright \mathbf{F}$  is a feature,  $\mathcal{X}$  is a set of  $N$  training examples
   $\forall \mathbf{x}^i \in \mathcal{X}, \mathbf{z}_0^i \leftarrow \mathbf{z}_0$ 
  for  $j = 0$  to  $k$  do
    Estimate  $\theta_j$  based on  $\mathbf{z}_j^i, i = 1..N$ 
     $\mathbf{z}_{j+1}^i \leftarrow \arg \max_{\mathbf{z} \in \mathbf{Z}} \theta_j(\Delta_{\mathbf{F}}(\mathbf{x}^i, \mathbf{z}), \mathbf{z}), \forall i$ 
  end for
end procedure

```

Fig. 2. Pseudo-code for the d-feature model refinement procedure.

physical feature. In this section, we explain how to train a model for a d-feature. In section 3, we explain how to combine models for individual d-features to build an ensemble that represents the object as a whole.

### 2.1. Learning Deformable Features

The main advantage of feature-based models is the automatic selection of representative features from a very large pool. We do not even need to know what the underlying physical features are. Therefore, our framework has to be able to automatically learn d-features based solely on the image data.

Let  $\mathbf{F} = (\mathbf{s}, \mathbf{z}_0, \mathbf{Z})$  be a d-feature identified by its size  $\mathbf{s}$ , its initial location  $\mathbf{z}_0$  and a neighborhood  $\mathbf{Z}$  relative to  $\mathbf{z}_0$  in which the feature is allowed to move. Let  $\Delta_{\mathbf{F}}(\mathbf{x}, \mathbf{z})$  be a descriptor of the feature’s appearance in an example  $\mathbf{x}$  at location  $\mathbf{z} \in \mathbf{Z}$ , *e.g.* a HOG descriptor [9]. For simplicity, we will omit the variable  $\mathbf{x}$  when confusion is not expected. Let  $\theta(\Delta_{\mathbf{F}}, \mathbf{z})$  be a scoring function that measures the likelihood of an example being positive given the appearance of the feature  $\mathbf{F}$  at location  $\mathbf{z}$ , *i.e.*  $p(\mathcal{O} | \Delta_{\mathbf{F}}(\mathbf{z}), \mathbf{z})$ . Note that,  $\theta$  depends on both  $\Delta_{\mathbf{F}}(\mathbf{z})$  and  $\mathbf{z}$ . This allows us to model the case when the prior probability of  $\mathbf{z}$  is not uniform.

On one hand, to learn the scoring function  $\theta$ , we need to know the locations of the feature  $\mathbf{F}$  in the training examples. On the other hand, to estimate the location of the feature in a given example, we need an objective function (scoring function) to optimize (maximize) over the feature’s neighborhood  $\mathbf{Z}$ . To break this cycle, we can start with an approximation to the scoring function by assuming the feature’s location in all training examples to be the initial location  $\mathbf{z}_0$ . Let  $\theta_0$  be the initial estimate for the scoring function obtained based on this assumption. Recall our prior assumption that features move within a small neighborhood around their initial (typical) locations. If we further assume also that typically the feature is close to its initial location, then the initial model  $\theta_0$  is expected to capture the rough appearance of the feature. Therefore, we can use  $\theta_0$  to estimate the feature location in a given example by maximizing the function over the neighborhood  $\mathbf{Z}$ . Given these estimated locations, we can learn a better estimate for the scoring function  $\theta$ . We can keep iterating over these two steps to reach a refined estimate for the scoring function  $\theta$ . This procedure is illustrated in Figure 2.

To visualize the effect of refining the d-feature’s model, consider the toy classification task illustrated in Figure 3. In this task, all images are  $40 \times 40$ . Positive samples contain circles with the same radius of 8 pixels. The circles can be at random locations in the  $20 \times 20$  central square of the image. Negative images contain random points in the same central square. We trained a Linear Discriminant Analysis model on the raw binary pixel values of the internal  $20 \times 20$  squares in all images. In the bottom row of Figure 3, we show the obtained weight vectors after 0, 1, and 2 refinement iterations. We can observe that the more we refine the model, the better it matches

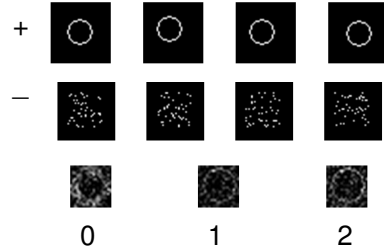


Fig. 3. A toy classification task to illustrate the effect of refining d-feature’s model. Sample positive and negative images are in the first the second rows. The learned weight vector after 0, 1, and 2 refinement iterations, Figure 2, are in the bottom row. Refinement enhances the match to the shape of the positive object.

the shape of the object we are training for, which is a circle in this case.

### 2.2. Classification With Deformable Features

We explained how a d-feature learns its best location on each training example and its object likelihood function through iteratively refining both in alternation. On a testing example, we select the feature location,  $\mathbf{z}^*$ , to be the location that maximizes the scoring function, and then consider the score at that location to be the object likelihood, equations 1 and 2.

$$\mathbf{z}^* = \arg \max_{\mathbf{z} \in \mathbf{Z}} \theta(\Delta_{\mathbf{F}}(\mathbf{z})) \quad (1)$$

$$\theta^* = \theta(\Delta_{\mathbf{F}}(\mathbf{z}^*)) \quad (2)$$

This procedure is equivalent to finding the Maximum Likelihood (ML) estimate of the feature location and using the corresponding object likelihood value as the score of the feature for the given test sample. This is similar to the way parts are deformed by Felzenszwalb et al. [11].

## 3. BOOSTED DEFORMABLE FEATURES

A boosting algorithm forms a strong classification ensemble out of weak classifiers. It adds ensemble members incrementally so that each newly added member performs the best in the training samples that are poorly learned by the current ensemble. In feature-based detectors, each weak classifier is built on a single feature, and the boosting algorithm selects one feature to add to the ensemble in every iteration. There are several variants of boosting. We experimented with the LogitBoost algorithm [7]. For completeness of presentation, the algorithm is reproduced in Figure 4 with the necessary modifications to fit with our framework. The only change is in the fitting of  $z_i$  to  $x_i$ , where  $z_i$  is computed by the algorithm, and  $x_i$  in our case is the  $\Delta_{\mathbf{F}}$  descriptors. In the case of d-features, we do not apply one step of least squares regression. Instead we use the iterative procedure in Figure 2 to allow the feature to find its best location. An important point to make here is that values of the  $\theta$  function used to update  $F(x)$  in the final step of the for loop of LogitBoost must be based on the estimates of the best locations  $\mathbf{z}$  computed in the final iteration of DefRefine in Figure 2. It is tempting to skip computing the  $\mathbf{z}$  values in the final iteration, since they are not used to update the model for  $\theta$  again. However, they are used

**procedure** LOGITBOOST( $\mathcal{F}, \mathcal{X}$ )  
 $\triangleright \mathcal{F}$ : set of  $M$  features,  $\mathcal{X}$ : set of  $N$  examples  
 $\forall \mathbf{x}^i \in \mathcal{X}, w_i = \frac{1}{N}, p(\mathbf{x}^i) = \frac{1}{2}, F(\mathbf{x}^i) = 0$   
**for**  $k = 1$  to  $K$  **do**  
    Compute the working response and weights  

$$z_i = \frac{y_i^* - p(\mathbf{x}^i)}{p(\mathbf{x}^i)(1 - p(\mathbf{x}^i))}$$

$$w_i = p(\mathbf{x}^i)(1 - p(\mathbf{x}^i))$$
     $\forall \mathbf{F} \in \mathcal{F}$  fit the function  $\theta_{\mathbf{F}}$   
    by a weighted least-squares regression of  $z_i$  to  $\mathbf{x}^i$   
    with weights  $w_i$  using the procedure in Figure 2.  
    Update  $F(\mathbf{x}) \leftarrow F(\mathbf{x}) + \frac{1}{2}f_k(\mathbf{x})$ , and  
     $p(\mathbf{x}) \leftarrow e^{F(\mathbf{x})} / (e^{F(\mathbf{x})} + e^{-F(\mathbf{x})})$ ,  
    where  $f_k(\mathbf{x})$  is  $\theta_{\mathbf{F}}$  that minimizes the residual.  
**end for**  
Output the classifier  $sign[F(\mathbf{x})]$   
**end procedure**

**Fig. 4.** Pseudo-code for the LogitBoost algorithm on d-features. Note that  $y_i^*$  is set to 0 for a negative example and to 1 for a positive example.

to compute the object likelihood scores, which, in turn, are used to update  $F(x)$  of LogitBoost.

#### 4. IMPLEMENTATION DETAILS

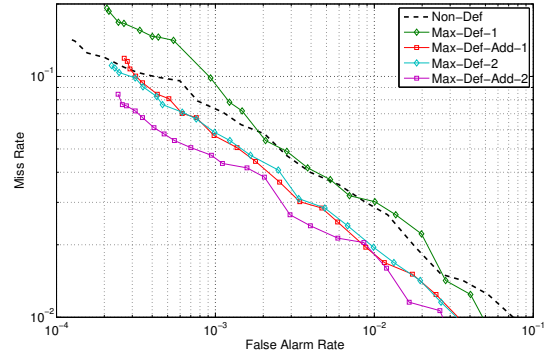
We use the HOG descriptor [9] to represent the features. The HOG descriptor of a feature is a concatenation of four histograms, each built on one quadrant of the feature. Each histogram contains 9 bins representing 9 ranges of orientation directions. Each pixel contributes to two bins of the histogram by linear interpolation. Each pixel also contributes to the 4 quadrants with bilinear interpolation. Computing these descriptors is very fast using kernel integral images [12].

The deformation neighborhood is made to be double the size of the feature in both dimensions, with a maximum of 16 pixels away from the feature’s boundary. On searching for the best feature location, we use 5 steps in each dimension. We use two types of scoring functions. One is based only on the descriptor and the other is based on the descriptor and location together. In the latter version, similar to Felzenszwalb et al. [11], we concatenate  $\delta = \mathbf{z}^* - \mathbf{z}_0$  and its element-wise square to the descriptor and estimate a function  $\theta$  based on the concatenated descriptor. Since the function  $\theta$  in our case is a linear function, the concatenation of  $\delta$  values to the descriptor is equivalent to decomposing  $\theta$  as  $\theta_{descriptor} + \theta_{displacement}$ . Therefore, this is equivalent to using an additive penalty term in the scoring function. This is also equivalent to learning a non-uniform prior for the feature location.

We use a rejection cascade [13] of 30 layers of LogitBoost classifiers. Each layer is adjusted to produce detection rate of 99.8% at false alarm rate of 65%.

#### 5. EXPERIMENTAL RESULTS

We trained and tested all our classifiers on the INRIA Person dataset [3]. In this dataset, training and testing positive images are resized so that the human body is around 96 pixels high. A



**Fig. 5.** DET curves for cascaded boosted HOG features classifiers on INRIA Person dataset with and without d-features. Using d-features helps reduce the miss rate by up to 30% at false alarm rate of  $3 \times 10^{-4}$ , and reduce the false alarm rate by 66% at the miss rate of 8%.

margin of 16 pixels is added to the top and the bottom to make the height 128 pixels and the width 64 pixels. The negative testing images are scanned with this window size ( $64 \times 128$ ) with a step of 8 pixels in both dimensions, to create close to a million sample negative images.

In this section, we refer to the variant of d-features that uses an additive penalty term in the scoring function (Section 4) by Max-Def-Add, and the variant without penalty as Max-Def. We experimented with the two variants with number of refinement steps 1 or 2, along with the conventional Non-Def features (0 refinements). We use DET (Detection Error Tradeoff) curves to present the detection results, figures 5, where the plots are generated by changing the number of cascade layers. In these plots the number of refinements appears at the end of the legend, when applicable. As the figure shows, only the Max-Def-2 and the Max-Def-Add-2 consistently outperform the Non-Def classifier. Max-Def-Add-1 compares favorably over most of the false alarm rate’s range. Max-Def-1 is inferior to the Non-Def classifier beyond false alarm rates of  $2 \times 10^{-3}$ . Max-Def-Add-2 is the clear winner among all. At a false alarm rate  $3 \times 10^{-4}$ , Max-Def-Add-2 reduces the miss rate compared to Non-Def by 30%, from 10% to 7%. At the miss rate of 8%, it reduces the false alarm rate to about one third, from  $8 \times 10^{-4}$  to  $2.5 \times 10^{-4}$ . These results highlight the value of d-features and the importance of performing multiple refinement iterations during training.

In Figure 6, examples of detection errors obtained using Non-Def that are successfully corrected using Max-Def-Add-2 are shown. To produce these images, each classifier is applied to the image using a sliding window approach, where the search step is set to 5% of the size of the search window in each dimension. The search sizes are selected based on knowledge of ground truth annotations. The resulting detection windows are then grouped using the mean shift algorithm on the location and height of the windows. For each searching size, the image is resized so that we always search using the size used in training.

#### 6. CONCLUSION

We introduced deformable Features (d-features in short) and showed how they can be used to enhance the performance of boosted feature-based object detectors. The advantage of d-features over the regular ones is their ability to search for the locations of the corre-



**Fig. 6.** Sample detections by the cascade boosted HOG features classifiers. The top image of every pair has the results from the Non-Def classifier (without d-features). The bottom image has the results with d-features with 2 refinement iterations and with a penalty term.

sponding physical features before computing their matching scores. This property makes them able to better handle complicated object structures and deformations than fixed location features. We experimented with d-features on human detection in a cascaded boosting framework. Our experiments showed a consistent enhancement in performance when using d-features.

We use brute force search in our current implementation, which makes training and testing classifiers using d-features slow. However, the distance transform techniques [14] can be used to make it more efficient. This approach can be extended in many other ways. We can apply the d-features using other common descriptors, such as the covariance descriptors [10]. Other objects, rigid and non-rigid, can benefit from the approach.

## 7. REFERENCES

- [1] D.M. Gavrila and V. Philomin, "Real-time object detection for smart vehicles," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, Fort Collins, CO, 1999, pp. 87–93.
- [2] P. Papageorgiou and T. Poggio, "A trainable system for object detection," *Int'l J. of Computer Vision*, vol. 38, no. 1, pp. 15–33, 2000.
- [3] Navneet Dalal and Bill Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, San Diego, CA, 2005, pp. 886–893.
- [4] A. Mohan, C. Papageorgiou, and T. Poggio, "Example-based object detection in images by components," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 23, no. 4, pp. 349–360, 2001.
- [5] K. Mikolajczyk, C. Schmid, and A. Zisserman, "Human detection based on a probabilistic assembly of robust part detectors," in *Proc. European Conf. on Computer Vision*, Prague, Czech Republic, 2004, vol. 1, pp. 69–81.
- [6] Zhe Lin, Larry Davis, David Doermann, and Daniel DeMenthon, "Hierarchical part-template matching for human detection and segmentation," in *Proc. 10th Intl. Conf. on Computer Vision*, Rio de Janeiro, Brazil, 2007.
- [7] Jerome Friedman, Trevor Hastie, and Robert Tibshirani, "Additive logistic regression: A statistical view of boosting," *Annals of Statistics*, vol. 28, pp. 337407, 2000.
- [8] P. Viola, M. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, New York, NY, 2003, vol. 1, pp. 734–741.
- [9] Qiang Zhu, Shai Avidan, Mei-Chen Yeh, and Kwang-Ting Cheng, "Fast human detection using a cascade of histograms of oriented gradients," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, New York, NY, New York, June 2006, vol. 2, pp. 1491 – 1498.
- [10] Oncel Tuzel, Fatih Porikli, and Peter Meer, "Human detection via classification on riemannian manifolds," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.
- [11] Pedro Felzenszwalb, David McAllester, and Deva Ramanan, "A discriminatively trained, multiscale, deformable part model," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, , Anchorage, AK, 2008, pp. 1–8.
- [12] Mohamed Hussein, Fatih Porikli, and Larry Davis, "Kernel integral images: A framework for fast non-uniform filtering," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, , Anchorage, AK, June 2008, pp. 1–8.
- [13] Paul Viola and Michael Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, Kauai, HI, 2001, p. 511518.
- [14] Pedro Felzenszwalb and Daniel Huttenlocher, "Pictorial structures for object recognition," in *Int'l J. of Computer Vision*, 2005, vol. 61, pp. 55–79.