Learning Where to Position Parts in 3D

Marco Pedersoli
Inria*†
marco.pedersoli@inria.fr

Tinne Tuytelaars
PSI-iMinds KU Leuven, Belgium
tinne.tuytelaars@esat.kuleuven.be

Abstract

*A common issue in deformable object detection is finding a good way to position the parts. This issue is even more outspoken when considering detection and pose estimation for 3D objects, where parts should be placed in a three-dimensional space. Some methods extract the 3D shape of the object from 3D CAD models. This limits their applicability to categories for which such models are available. Others represent the object with a pre-defined and simple shape (*e.g. *a cuboid). This extends the applicability of the model, but in many cases the pre-defined shape is too simple to properly represent the object in 3D. In this paper we propose a new method for the detection and pose estimation of 3D objects, that does not use any 3D CAD model or other 3D information. Starting from a simple and general 3D shape, we learn in a weakly supervised manner the 3D part locations that best fit the training data. As this method builds on a iterative estimation of the part locations, we introduce several speedups to make the method fast enough for practical experiments. We evaluate our model for the detection and pose estimation of faces and cars. Our method obtains results comparable with the state of the art, it is faster than most of the other approaches and does not need any additional 3D information.*

1. Introduction

For many practical applications, merely detecting the bounding box of an object of interest is not enough. A more precise localization of the object including also its 3D pose is needed. For instance, think of a robotic arm that needs to grasp an object. It needs to know not only where the object is, but also its 3D pose. For the sake of brevity, in the rest of the paper we use the term 3D detection to refer to object detection (*i.e*. find the object bounding box) and pose estimation (*i.e*. find the orientation or pose of the object). Note that this term refers to the model representation, not to the input data. We use ordinary 2D images as input, with no depth information available during either training or test time.

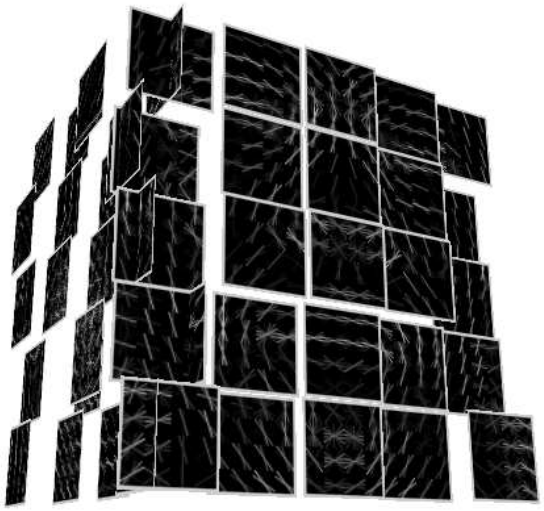

Figure 1. Our 3D model representation for the 'face' class. Starting from an RGB image (no depth), our model simultaneously learns appearance and depth of each HOG part.

In recent years several methods for 3D detection have been proposed. Among them, those based on extensions of deformable part models (DPM) [3] to 3D [5, 10, 17, 19] seem to be the best performing. However, they still have several issues. First, *the optimal part positioning* in 3D is an open issue. For the 2D case, the strategy proposed in the original DPM [3] based on selecting the location with higher discriminativity seems adequate. For 3D models the problem becomes more complex because we want to associate the same object part, seen from different viewpoints, to the same model location. For instance we would like to recognize the wheel of a car with the same model part, independently from the view.

Two main solutions have been proposed in the literature. Pepik *et al*. [17] associate parts seen from different viewpoint from 3D CAD models of the object class. It selects a set of parts on the 3D CAD model and then it places these on the training images using the ground truth orien-

*LEAR team, Inria Grenoble Rhone-Alpes, Laboratoire Jean Kuntzmann, CNRS, Univ. Grenoble Alpes, France.

†The author has done this work while he was with PSI/iMinds KU Leuven, Belgium

tation. This works quite well, but assumes availability of a 3D CAD model for the object category to be detected. Furthermore the CAD model should be general enough to represent all the instances of a certain class. If this is not the case, additional annotations are needed to associate the CAD subclass with the training samples.

Fidler *et al*. [5] instead use a simpler 3D structure of the object and parts are placed using a heuristic that does not depend on the specific class considered. More specifically, the object is represented with a cuboid whose dimensions are estimated from the training data. Then, parts are positioned with a strategy similar to DPM and are allowed to move only on the corresponding cuboid face. This approach does not use any 3D model or additional part annotations, but its representation is strongly limited to the similarity of the object to a cuboid. If the faces of an object are not flat, the location of the parts would be estimated wrongly and therefore the learned model would underperform.

An additional issue that limits the use of 3D models is their *high computational cost*. The main idea behind pose estimation is to evaluate several projections of the model with different poses and pick the best one as estimated pose. Thus, their computational cost is linear in the number of evaluated poses or views. Most of the methods, in practice, for an accurate pose estimation need more than 10 views. Thus, we can estimate that their computational cost is at least 10 times the cost of the used 2D models.

In this paper we present a method to tackle these two issues. We build a 3D deformable model as a composition of HOG parts that lie in 3D space. Each part is defined by its 3D position and orientation. At test time, the model is projected to a 2D image plane and used to detect the object. An example of a 3D model trained on a faces dataset is shown in Fig. 1. For a fast evaluation of the HOG parts on the image we adapt the technique proposed in [16] to deformable objects (Sec. 3.1). Similarly to DPM, we let the parts move to account for local distortion. However, our parts move in 3D space, with a quadratic displacement cost (Sec. 3.2). For a linear estimation of the quadratic cost, we extend the generalized distance transform to allow non axis-aligned deformation costs (also explained in Sec 3.2). As in [5], the initial shape of our model is a cuboid composed of HOG parts placed in 3D. However, during learning (Sec. 3.3) we simultaneously learn not only the appearance and deformation of each HOG part, but also their optimal depth (Sec. 3.4). Thus, the model iteratively adapts its 3D shape to the data. All together this results in a fast and robust 3D detector that can detect and estimate the fine pose of an object with similar or better accuracy than previous models (Sec. 4). All of this is performed in less than 30 seconds on a single CPU.

2. Related work

Many different methods have tackled the problem of 3D detection. Our focus is on those based on HOG features and deformable models. A recent trend is to use activations of deep convolutional neural networks, leading to excellent results. The HOG descriptors could be replaced by these. However, for simplicity and to allow for a fair comparison with our baselines, we left this as future work. Felzenszwalb *et al*. [3] introduced the deformable part model (DPM), which extends the HOG detector [1] by introducing moving parts as latent variables and a principled way to learn the model deformations. Our method is based on DPM, the main difference being that our model is fully 3D, where each part has full knowledge about its 3D location and orientation, given a certain object pose.

Thus, our model can be considered as a 3D generalization of DPM, which typically operates purely in 2D. DPM assumes that object part locations can slightly vary due to object deformations. However, in practice the main source of 2D part displacement is not the deformation of the object *per se*, but a viewpoint change[1]. If we consider the part location in 3D, the part displacement due to viewpoint is explicitly modeled. The deformation then only needs to handle the real displacement of the part, *i.e*. due to a non-rigid deformation or to instance-level differences. This results in smaller deformations, and therefore, one can expect a reduction of false positive detections.

For the part initialization DPM uses a greedy strategy sequentially placing each part at the 2D object location that is estimated as most discriminative. This approach works well, but cannot be generalized to 3D. For a fast computation of the optimal part deformations DPM uses a generalized distance transform that renders that computation linear in the number of part locations. In our approach we show how to use the fast generalized distance transform in 3D.

A possible way to detect an object and estimate its pose is to use a representation composed of multiple 2D templates, each for a specific view of the object. An example of such approach is proposed by Gu and Ren [8]. They learn multiple templates at the same time and use them to detect the location of the object and to predict its pose. A natural extension of this work to deformable templates is presented in [13], where it is also shown that using other poses as negative samples improves pose estimation at the cost of a reduction in detection accuracy. The same line of using multiple deformable templates for pose estimation can be found in [9, 18, 20, 21]. For instance, different variants of a model based on a tree of parts were used for human pose estimation [20], human face pose estimation and facial point localization [21], and 3D car viewpoint estimation [9].

[1]For instance, see the principal component analysis applied to deformations shown in [21].

Pepik *et al*. [18, 17] explicitly associate each moving part of different templates to a 3D landmark. The model and the deformation of the parts is then learned from 3D CAD samples, where the exact location of each part is known. In contrast, in our model we do not know anything about the parts location. Unlike previous methods that learn an independent template for each view, Xiang and Savarese [19] use a single 3D representation. Their model is a composition of planar surfaces or aspects that are estimated from a set of 3D CAD models. Similarly, our model uses a single 3D representation of the object, but without the need of a 3D CAD model. Additionally, in their approach the projection of the 3D model to the 2D plane is done by distorting the original image, while in our method it is effectuated at feature level, which is much faster. Another interesting approach is the 3D object recognition by object reconstruction [10], where object detection and pose estimation are effectuated by synthesizing a view of the object in the HOG space from a set of given 3D object representations. As the number of possible views is very high, similar views are clustered.

All of the aforementioned methods that estimate the 3D shape of an object class require some sort of 3D information as input, either in form of 3D CAD models or object instance landmarks. Our method instead handles the 3D of the object without any specific 3D information, but directly from the RGB data. We assign a reasonable initial 3D location of the parts and then we refine their location during training. The only method that builds a 3D model without using an explicit CAD model is [9]. In this case however the 3D detection is divided in two stages. The first one is based on a 2D multi-view detector. The second builds a 3D model to refine the pose estimation. However, for building the 3D model the method needs 2D part annotations, which are expensive to collect.

Fidler *et al*. [5] have proposed a method that has many similarities with ours. Both methods are based on a 3D cuboid composed of deformable parts. However, for the detection of non-frontal parts Fidler *et al*. apply the distortion at image level, as in [19]. More importantly, while Fidler *et al*. use a cuboid representation, in our model the cuboid is only used as initialization of a weakly supervised learning procedure used to refine the 3D parts location.

3. Our 3D model

We define a 3D model for object detection such that a single model can be used to represent any possible view of the object class. Consequently, we define our model as a set of 2D patches placed in a 3D world. Each patch has a well-defined 3D location $l^O = (l_x, l_y, l_z)$ and orientation $n^O = (n_x, n_y, n_z)$ with respect to the object reference location $o = (o_x, o_y, o_z)$ and rotation $\theta = (\theta_x, \theta_y, \theta_z)$ as shown in Fig. 2. A patch is a 2D representation of a small region of a 3D object surface, whose appearance varies with

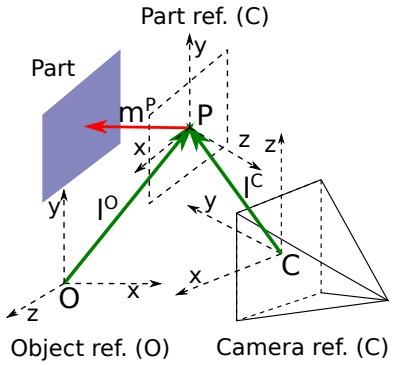

Figure 2. Illustration of the three reference systems used in the paper, one attached to the object O, another attached to the camera C and one attached to the part location at resting position P.

the viewpoint from which the object is seen. In this sense it has many similarities with what in computer graphics is called a texture. However, as in this work we use these patches for object detection in a similar way as in DPM, where patches actually represent parts of an object, from now on we will call them *parts*, as in DPM nomenclature. Still, the reader should consider that these parts are quite different from DPM parts, especially because our parts are placed and can be moved in a 3D environment.

During inference, we project the learned model on the image plane at any location o and with any possible pose θ, defined as three sequential rotation angles of the model. Detections with a certain location and pose are then ranked based on their score. Notice that in principle, and in contrast to many 2D based methods, with this model we can have a continuous estimate of the object pose. The final accuracy will depend on the number of model evaluations, which also influences the computational cost of the model. In Sec. 3.1 we present a strategy to make the evaluation of many possible poses computationally fast.

During training, as the location and pose of the object are known, we project the 2D appearance of each training sample on the corresponding parts of the 3D model. In practice, for each training sample, a specific set of parts is filled up and the rest is set to zero. This representation is converted into feature descriptors for support vector machine learning. Negative samples are extracted from images not containing the object and with randomly sampled poses. As the number of possible negatives is too high to fit into memory, a negative mining strategy is used.

In the following we present in detail our model formulation for inference and learning. We first define how to compute the score of an object as a sum of 3D parts, explaining the choice of the orthographic camera model, and how the extraction of HOG features is performed and sped-up. Then, we extend the model considering that the object parts can be displaced in 3D from their resting position with a quadratic cost function. Next, we show how to use the dis-

tance transform for 3D deformations in a fast way. Finally, we present the learning formulation and extend it to also learn the 3D part locations.

3.1. Inference

Scoring function The score of an object part i (whose appearance is represented by a set of learned weights w) depends on its relative location l^C and orientation n^C with respect to the camera:

$$< w, \phi(I, l^C, n^C) >, \tag{1}$$

where ϕ is the feature representation (discussed in detail later).

Using basic geometry the transformation from part positions l^O and orientations n^O with respect to the object center to part positions l^C and orientation n^C with respect to the camera is:

$$l^C = R_\theta(l^O + o), \qquad n^C = R_\theta n^O \tag{2}$$

where R_θ is the rotation matrix composed as sequential application of $(\theta_x, \theta_y, \theta_z)$ around the corresponding axes. The score of a part is then:

$$scr^p(w, l^O, n^O, o, \theta, I) =< w, \phi(I, R_\theta(l^O + o), R_\theta n^O) >,$$

while the entire object scoring function becomes:

$$scr(W, L, N, o, \theta, I) = \sum_i scr^p(w_i, l_i, n_i, o, \theta) \tag{3}$$

where $L = \{l_0, l_1, ..., l_P\}$ and $N = \{n_0, n_1, ..., n_P\}$ are part locations and orientations with respect to the object reference system o and $W = \{w_0, w_1, ..., w_P\}$ are the corresponding learned parameters.

Camera parameters We have defined $\phi(I, l^C, n^C)$ as a function that extracts image features at the 3D location and orientation of a part. However, our image is 2D and we should now define a camera model to project the 3D points into the 2D image. As we search for objects at multiple scales, we assume the z-coordinate of a part l^C_z to be proportional to its scale. However, we want to be able to detect objects in uncalibrated camera images where the factor that associates the distance of an object to its scale (i.e. the camera focal length f) is unknown. Thus, we should estimate the factor f for each image. Trying several values of f and selecting the best one would be computationally too expensive. Instead, we assume an orthographic projection (i.e. $f = +\infty$) for a single object. In practice, for the same object we assume that all parts have the same size and their location is (l^C_x, l^C_y, o_z). For objects far enough from the camera the orthographic projection is a good enough approximation of the object viewpoint which makes the estimation of camera parameters unnecessary. Considering that the 3D structure of the model is a coarse approximation of the real 3D shape, using a better projection model would not help much; small errors in part localization can be absorbed by part deformations as explained in Sec. 3.2. Still, we use the z-coordinate of the object center location o_z as a scale factor when searching for the object at multiple scales. In this case, the absolute value of o_z is not the real distance of the object from the camera, but for our application that is not really important. We also assume that an object will mostly be observed in its standing pose ("upright" assumption, common in object recognition) and therefore we do not need to consider part rotations on the camera plane (z axis). This limits our 3D model to fully rotate only over one axis (in our case y) and partially over the other (in our case x). We leave a detailed estimation of the camera parameters and the extension to camera plane rotations as future work.

Feature extraction In this work we use HOG features [1] for describing the appearance of each object part. However, in principle the method can be applied to any dense feature representation like bag-of-words [12] or activations of convolutional neural networks layers [11]. Given an image I the corresponding HOG feature is constructed as:

$$\phi(I, l^C, n^C) = \mathcal{V}(\mathcal{H}_{l^C_z}(\mathcal{T}_{n^C}(I), l^C_x, l^C_y)), \tag{4}$$

where $\mathcal{H}_s(I, x, y)$ is a function that extracts from image I HOG features (as (x, y, d), with (x, y) being spatial dimensions and d being the feature dimensionality, i.e. the gradient orientations for HOG) of a part at scale s and location (x, y). $\mathcal{T}_{n^C}$ is a transformation that accounts for the perspective distortion of the part appearance when observed at a certain orientation n^C from the camera. Finally, $\mathcal{V}$ converts the HOG features into a flat vector that is suitable for multiplying with w. Assuming again an orthogonal projection, and defining (η_x, η_y, η_z) as the (x, y, z) components of the part orientation vector n^C in the camera reference system C, $\mathcal{T}_{n^C}$ reduces to:

$$\mathcal{T}_{n^C}(I(x, y)) = \begin{cases} I(\eta_x x, \eta_y y) & \text{if } \eta_z > 0 \\ \vec{0} & \text{if } \eta_z \leq 0, \end{cases} \tag{5}$$

which is an affine transformation if the part faces the camera ($\eta_z > 0$) and an image of zeros otherwise. Note that this simple trick allows the method to reason about parts self occlusion. This formulation gives satisfactory results, but it is slow because a different image distortion and new HOG features should be computed and evaluated for each part orientation. In the following we propose a much faster solution.

Since the HOG feature representation still maintains a geometrical description of the appearance of a part, we can move the geometrical transformation due to the part orien-

tation $\mathcal{T}_{n^C}$ from image level to feature level:

$$\mathcal{H}_{l_z^C}(\mathcal{T}_{n^C}(I), l_x^C, l_y^C) \approx \begin{cases} \mathcal{H}_{l_z^C}(I, \eta_x l_x^C, \eta_y l_y^C) & \text{if } \eta_z > 0 \\ \vec{0} & \text{if } \eta_z \leq 0. \end{cases} \quad (6)$$

As shown in Fig. 3, this new formulation avoids applying the transformation $\mathcal{T}_{n^C}$ at image level and therefore also avoids computing different features for different transformations. As the HOG features have coarser granularity than pixels, this transformation introduces a higher quantization error. To reduce this error, rather than using an interpolation algorithm, we use hand-tuned linear HOG combinations as defined in [16]:

$$\mathcal{H}_{l_z^C}(I, \eta_x l_x^C, \eta_y l_y^C) \approx \sum_{x,y} \alpha_{x,y,\eta_x,\eta_y} (\mathcal{H}_{l_z^C}(I, l_x^C + x, l_y^C + y)),$$

where $\alpha_{x,y,\eta_x,\eta_y}$ are coefficients that best approximate the transformation $\mathcal{T}_{n^C}$.

Considering the dot product linearity, we can compute the score of the parts as in Eq. 3 on the non-distorted HOG features. In this way we can compute the score of each HOG cell of the frontal view (without distortion) for each part. Afterwards, the score of any other view can be obtained as a linear combination of frontal views. In practice, considering that the used HOG features have 32 dimensions per cell, computing from scratch the score of a part with size 4×4 HOG cells at a given location requires $4 \times 4 \times 32$ multiplications of the feature vector with the corresponding parameter vector w. Instead, by pre-computing the score of each HOG cell for the frontal appearance of the part, for any other view we only need to calculate the linear combination of 2 pre-computed HOG cell scores. Therefore, the new total cost is $4 \times 4 \times 2$ with a neat gain of $16\times$. The final gain is reduced because we still need to pre-compute the HOG cell scores for the frontal appearances of the parts. However, especially if we want a precise estimation of the object pose, a high number of possible views should be computed. The pre-computation cost is then distributed over more views and the final gain approximates the ideal case of 16.

3.2. 3D Deformations

Scoring function So far, we have considered that the part position is defined only based on the 3D location o and pose θ of the object. However, from DPM and max pooling strategies, we know that allowing parts to have a small neighborhood over which to search for the highest score can highly enhance their discriminativity. Thus, in our model we allow parts to move in the three dimensions x, y, z. We define the deformation cost of a part i as a quadratic function over the part displacement $m^P = (m_x, m_y, m_z)$ with respect to the part reference system defined by the part location l and orientation n:

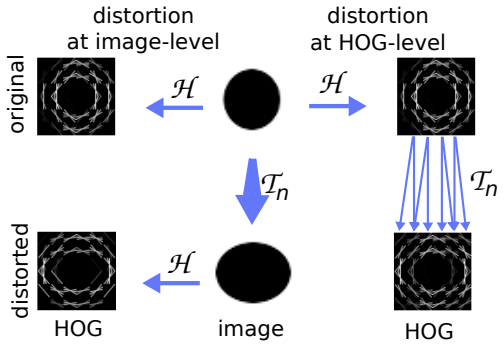

Figure 3. Example of a transformation $\mathcal{T}_n$ in the image space and in the feature space. Applying the transformation in the feature space (right) is faster because you can avoid having to transform the image and then compute the HOG features again (left).

$$m^{PT} \begin{bmatrix} d_x & 0 & 0 \\ 0 & d_y & 0 \\ 0 & 0 & d_z \end{bmatrix} m^P = m^{PT} D m^P, \quad (7)$$

where (d_x, d_y, d_z) defines the deformation costs that should be learned and T as super-index of the vector m denotes the transpose. Now, we can convert the local reference system attached to part location and pose to the camera reference system C: $m^C = R_\theta R_n m^P$. We first convert the local reference system to the global object reference system through R_n which is the rotation obtained from the part orientation n. Then we transform this new vector to the camera coordinate system through R_θ, which is the previously defined object rotation matrix. See Fig. 2 for reference. We can apply these transformations directly on the quadratic function as:

$$m^{CT} R_\theta R_n D R_n^T R_\theta^T m^C = m^{CT} \begin{bmatrix} q_x & q_{xy} & q_{xz} \\ q_{yz} & q_y & q_{yz} \\ q_{xz} & q_{yz} & q_z \end{bmatrix} m^C = m^{CT} Q_{\theta,n} m^C. \quad (8)$$

The new scoring function considering also the parts deformation is:

$$scr(W, L, N, o, \theta, I) = \sum_i \max_{m_i} \left(scr^p(w_i, l_i + m_i, n_i, \theta, I) - m_i^T Q_{\theta,n_i} m_i \right) \quad (9)$$

where m_i is the vector that defines the 3D deformation of part i in the camera coordinate system (we skipped the super-index C for clarity).

As we work with an orthographic projection, changing z-location of a part deformation m_z does not affect the appearance seen by the part, although it affects its deformation cost. In practice the location in m_z of a part is chosen to minimize the deformation cost:

$$m^T Q_{\theta,n} m = \min_{m_z} m^T Q_{\theta,n} m, \quad (10)$$

which can be solved in closed form and yields $m_z = -(q_{xz} * m_x + q_{yz} * m_y)/q_z$. The 2D deformation matrix is then:

$$Q^{2D}_{\theta,n} = \begin{bmatrix} q_x - q^2_{xz}/q_z & q_{xy} - q_{yz}q_{xz}/q_z \\ q_{xy} - q_{yz}q_{xz}/q_z & q_y - q^2_{yz}/q_z \end{bmatrix}. \quad (11)$$

In this way we can convert the 3D quadratic function into the corresponding 2D function which is then used for the 2D distance transform without any additional cost.

Notice that this formulation is similar to the one used in [17], where the authors assume that the 2D deformation cost for a certain view is the orthogonal projection of the 3D deformation cost, implicitly assuming $m_z = 0$. In contrast, we also optimize over m_z which leads to a better estimation of the deformation cost.

Generalized Distance Transform As we transform the 3D quadratic form defined by $Q_{\theta,n}$ to a quadratic form in the 2D camera plane $Q^{2D}_{\theta,n}$, we obtain a 2D function that still has a quadratic form in 2D, but in general not aligned to the main image axes. In this case the generalized distance transform algorithm [4] which is linear in the number of locations cannot be applied on each dimension independently as in [4]. As we need to apply distance transform to each view, it is important to compute it as fast as possible. For doing that, we find the eigenvalues of $Q^{2D}_{\theta,n}$ so that we find the principal axes of the deformation, rotate the score image to align it to $Q^{2D}_{\theta,n}$, apply the standard linear 1D distance transform algorithm on each axis and then rotate back the result. This proves to be faster and simpler than designing a specific 2D distance transform algorithm.

3.3. Learning

For each sample s we are given a tuple $(I_s, y_s, o_s, \theta_s)$ composed of an image, a label that defines the presence of the object, the location and the pose of the object (if present). Our goal is to find the parameters W that minimize the following objective function:

$$|W|^2 + C \sum_s \max(0, 1 - y_s scr(W, L, N, o_s, \theta_s, I_s)),$$

where C is the commonly used trade-off factor between regularization and loss.

Now, due to the maximization over latent parameters (the deformation of the object parts), the objective function is not convex. Thus, for its optimization we use the strategy defined in [3] where we optimize Eq. 9 using coordinate descent. First, we optimize W fixing the value for the latent variables for the positive samples. Then, with the obtained W we estimate the new value for the latent variables. This procedure is repeated until convergence, which is guaranteed because each steps leads to a reduction of the objective function. To be able to deal with an exponential number of configurations generated by all the possible locations, poses and values for the latent variables we use a caching procedure for the negative samples mining as in [3].

3.4. Shape estimation

In latent SVM the resting location of the parts is defined at the beginning of the learning procedure and maintained constant for the entire training procedure. We instead want to update the location of the part. Note that this is a weakly supervised task because no annotations of the correct parts location are used. More specifically, we want to move each part along the z component of the normal vectors n of a part to best represent the object class. Moving parts along x and y could also be useful. For example, it could be a refinement of the initialization of the parts locations used in [3]. This would allow the part location to move towards the most discriminative location during training. However, without any additional constraints, different parts could end up at the same spatial location, which is to be avoided.

Instead, moving the part only along z makes more sense because parts cannot overlap by construction and, more importantly, learning the correct z-location of each part produces a better estimation of the 3D shape of the object. By allowing the part to learn its best resting location tends to reduce the errors in the localization of the object parts and therefore it is likely to produce fewer false positive detections and a better pose estimation. We show this in Sec. 4.

Estimating the shape of the object class corresponds to finding the most appropriate translation in the z direction of the part orientation vector n, *i.e.* the part depth. We call this value t_z. We can find t_z as the value that minimizes the sum of the costs of the part displacements on the training data:

$$t_z = \arg\min_t \sum_{s=1}^{S} d_{z,s} |m_{z,s} - t|^2 = \frac{d_z}{S} \sum_{s=1}^{S} m_{z,s}. \quad (12)$$

Thus, in the learning algorithm, after estimating the part locations $m_{z,p}$ for the positive samples, we update the resting position of the part to $l^O_s = l^O_s + t_z n_z$. Note that this procedure effectively minimizes the objective funtion. In fact, for positive samples, t_z is optimized to maximize their scoring function, which corresponds to minimizing their loss. For negative samples, the new t_z would reduce their scores because it increases the costs of the parts displacements, but this, for negative samples corresponds to decreasing their loss.

4. Experiments

We validate the performance of our method for the task of object detection and pose estimation and compare it with state-of-the-art methods on two challenging datasets. We also study the importance of the different components of our method to observe their effect on the process of creating a

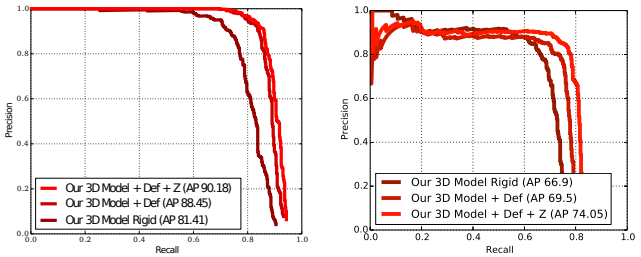

Figure 4. Precision-Recall on AFW for detection (left) and pose estimation (right) using different configurations of our 3D model.

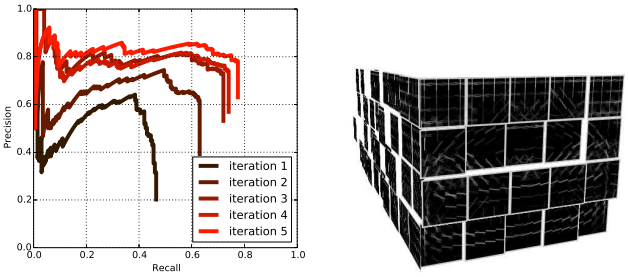

Figure 5. (left) Pose Estimation Precision-Recall on EPFL car over training iterations: note the improvement obtained in the latent SVM iterations. (right) 3D model trained on EPFL car.

robust and efficient 3D detector. For implementation details and additional details about the training and evaluation see the supplementary material.

AFW We evaluate our method for object detection and pose estimation on the annotated faces in the wild (AFW) dataset [21]. For object detection, we evaluate in terms of average precision (AP), where a detection is considered correct if it overlaps (intersection over union) more than 50% of the ground truth. For pose estimation, we use the pose estimation average precision (PEAP). We also evaluate the pose estimation in terms of pose accuracy at $15°$ and $30°$ (*i.e.* a pose is correct if the error is less than $15°$ or $30°$, as defined in [21]). In Table 2 we compare our model with state-of-the-art methods for detection and pose estimation. We evaluate our method with the same training protocol as in the original paper [21] which consists of training with 900 samples from MultiPIE [7]. Among these methods, ours reaches the top performance in detection and is the second best in pose estimation. For completeness in the table we also include top-performing detection methods that have been trained on different and much more training data.

In pose estimation our method obtains an accuracy close to the one reported in [6] for DeCAF features which are extracted from a pre-trained convolutional neural network. Considering that our model does not use any additional data (whereas DeCaf features are trained on 1 million images and 1000 classes from ImageNet), and it mainly learns only 3 views of the model (whereas HOG templates and TSM use 13 different views), the obtained result is quite impressive.

Method	MPPE8	MPPE16	MPPE36
MDPM [13]	73.7	66.0	-
3D^{2}PM [18]	77.9	69.1	**53.5**
Fisher [6]	76.6	**72.2**	51.8
DeCaf [6]	80.6	67.8	45.9
Our 3D model	**81.5**	56.4	36.8

Table 1. Results on EPFL car dataset. Our method works well for coarse pose estimation.

Method	Detection	Pose 15	Pose 30
multiview HOG [21]	75.5	74.6	85.0
3D model from [16]	78.8	71.4	-
TSM [21]	88.0	81.0	89.0
TSM shared [21]	76.2	76.9	87.0
Fisher [6]	88.3	78.6	90.6
DeCaf [6]	88.3	**86.5**	**93.4**
Our 3D model	**90.18**	85.9	92.1
DPM from [14]	97.21	-	-
HeadHunter [14]	97.14	-	-

Table 2. Results on the AFW dataset. Our model outperforms all the methods based on HOG features and is very close to the results obtained with more training data by DeCaf.

Effect of deformation When dealing with deformable models, the importance of deformations has been questioned [2]. As shown in Fig. 4 (left), in our model the introduction of deformation is fundamental to obtain good detection performance. By letting the parts of our model deform with a learned quadratic cost we boost the AP from 81% to 88%. For this specific dataset this means moving from the bottom to the top of the methods ranking! Considering pose estimation (Fig. 4 (right)), the improvement is also relevant moving from 66.9% to 69.5% PEAP.

Part positioning In Fig. 4 we report a quantitative evaluation of the effect of estimating the location of each HOG part in terms of detection and pose estimation. In both evaluations, it improves performance. While the improvement is relatively small for detection (especially if compared with the effect of deformation), for pose estimation the effect is quite relevant. This was to be expected, as for a good estimation of the 3D location of an object it is important to know the 3D location of its parts.

In Fig. 1 we show the learned 3D model for face on AFW. We can see that the learned depth of each part makes sense. In fact, starting from a cuboid, parts have adapted to a more spherical shape that better represents a face model. Also, we notice that the nose is more prominent than the other parts of the face. All of this has been learned in a weakly supervised way, without any information about the correct location of the parts, neither in 3D nor in 2D. Instead, their location is estimated during learning as explained in Sec. 3.4. To the best of our knowledge, this is the first method that produces a rough estimate of the 3D shape of an object class using only the bounding box loca-

tion and pose of the annotated objects in the training data. In Fig. 1 we can also see that the depth estimation of some parts seem wrong. For instance, the two parts at the left and right of the nose are also prominent. We belief that this is due to the fact that the nose is actually represented by those parts when the face is partially rotated. Thus, their depth makes sense too, although it is not really what one would expects for a 3D representation of a face. In this sense it is important to consider that the shape estimation is learned in an unsupervised way as side product of better detection and pose estimation. When some parts cannot learn a discriminative appearance, their 3D can be wrong. An example of that in Fig. 1 is the part at the bottom right. As this part is actually learning mostly background (note its appearance), its estimation is wrong. In future work we will consider some possible strategies to recognize and remove the parts that are not useful.

Computational Cost By using the part approximation explained in Sec. 3.1 the method without deformations is relatively fast and scales well with the number of views. For instance, it can evaluate using a single CPU the face model on an image of 640×480 pixel size, over 13 views in around 10 seconds. The deformable model is around three times slower due to the computation of the distance transform for each view. However, the current version of the code is not optimized, so a faster version should be feasible. Still, our method is faster than most of the HOG based methods that perform integrated detection and pose estimation. These methods are generally based either on distorting the image based on the viewpoint and computing and evaluating the HOG features in all views like [5, 19], or on applying a different model for each view [13, 18, 17]. Both approaches are much slower than ours.

EPFL cars We evaluate the fine-grained pose estimation on the EPFL car dataset [15]. As in [15], we use the first 10 videos for training and the other 10 for test. Detection is relatively easy in this dataset as cars are big, centered in the image and most of modern methods, includeing ours, obtain a detection rate of around 99%. Pose estimation is a more challenging task due to the high variability of the car models and the reduced number of training models (just 10). In table 1 we compare our method for the discrete pose estimation in terms of Mean precision Pose Estimation (MPPE) with different numbers of bins (8,16 and 36). Our model performs well for the coarse pose estimation, but it performs worse than the other evaluated methods for the finer pose estimation. We believe that this is due to the structure of the model. In our model, we represent the entire model always with 4 views while the other methods use 8,16 or 36 views, depending on the number of views needed in evaluation. The reduced number of learned parameters produces a better generalization of the model which allows it performing better for the 8 bins MPPE. Note that for other methods based on deformable HOG models, like [13, 17], using a high number of views make the detector very slow.

In Fig. 5 (left) we report the pose estimation precision-recall curves for several iterations of the latent SVM learning. It is noticeable how the pose estimation improves over iterations. This shows the power of a weakly supervised method where the deformation of the parts and their depth are learned on the data. Without this procedure the performance of the pose estimation would remain as at the first learning iteration. In Fig. 5 (right) we also show the learned 3D model for car. The estimated shape of the car is not as good as the face model. In particular, parts representing the frontal glass of the car remain on their initial position and do not really move much to the real 3D position of the frontal glass. We believe that this is due to the initialization, which is based on a cuboid. As the optimization is non convex, it finds a local minimum of objective. In this sense, the parts expected to move to the car frontal glass are probably initialized too far from their actual 3D position and remain stuck.

Limitations The proposed model aims to extend 2D HOG based approaches to 3D without the need of additional annotations besides the object pose. Even though the method attains very competitive results on the evaluated datasets, in some aspects it is still more limited than 2D based approaches. In particular, when the variability of the object class is high, as in PASCAL VOC, additionally to multiple views, multiple appearances for the different subclasses are needed. For multiview 2D HOG models there is no distinction between views and subclass appearances so multiple views can deal with multiple subclass appearances. In contrast, our 3D model (as other 3D HOG based models) assumes a multiview but single subclass representation. Thus, when dealing with subclasses with different appearance the estimation of the depth of the parts is wrong and therefore poor detection and classification are obtained.

5. Conclusions

In this paper we have presented a new and computationally efficient model for 3D object class detection and viewpoint estimation. The model is a composition of parts located in the 3D space. During training the appearance of the parts, their deformation cost and their location are learned in a weakly supervised manner without the need of any additional annotation or 3D information. Results show that the approach attains results comparable with state-of-the-art in both detection and pose estimation, but with less prior knowledge of the object class, and with a faster detection.

Acknowledgements

This work was supported by IWT-SBO project PARIS as well as KU Leuven GOA project CAMETRON.

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005. 2, 4

[2] S. K. Divvala, A. A. Efros, and M. Hebert. How important are "deformable parts" in the deformable parts model? In *European Conference on Computer Vision. Workshops and Demonstrations*, pages 31–40, 2012. 7

[3] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *Transactions on Pattern Analysis and Machine Intelligence*, pages 1627–1645, 2010. 1, 2, 6

[4] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, 2004. 6

[5] S. Fidler, S. Dickinson, and R. Urtasun. 3d object detection and viewpoint estimation with a deformable 3d cuboid model. In *Advances in Neural Information Processing Systems*, pages 611–619, 2012. 1, 2, 3, 8

[6] A. Ghodrati, M. Pedersoli, and T. Tuytelaars. Is 2d information enough for viewpoint estimation? In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. 7

[7] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker. Multi-pie. *Image and Vision Computing*, 28(5), 2010. 7

[8] C. Gu and X. Ren. Discriminative mixture-of-templates for viewpoint classification. In *Proceedings of the European Conference on Computer Vision*, 2010. 2

[9] M. Hejrati and D. Ramanan. Analyzing 3d objects in cluttered images. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, pages 602–610, 2012. 2, 3

[10] M. Hejrati and D. Ramanan. Analysis by synthesis: 3d object recognition by object reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014. 1, 3

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014. 4

[12] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bag of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2006. 4

[13] R. Lopez-Sastre, T. Tuytelaars, and S. Savarese. Deformable part models revisited: A performance evaluation for object category pose estimation. In *ICCV, 1st IEEE Workshop on Challenges and Opportunities in Robot Perception*, 2011. 2, 7, 8

[14] M. Mathias, R. Benenson, M. Pedersoli, and L. Van Gool. Face detection without bells and whistles. In *ECCV*, 2014. 7

[15] M. Ozuysal, V. Lepetit, and P. Fua. Pose estimation for category specific multiview object localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009. 8

[16] M. Pedersoli and T. Tuytelaars. A scalable 3d hog model for fast object detection and viewpoint estimation. In *3DV*, 2014. 2, 5, 7

[17] B. Pepik, P. Gehler, M. Stark, and B. Schiele. 3dpm - 3d deformable part models. In *European Conference on Computer Vision*, October 2012. 1, 3, 6, 8

[18] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3d geometry to deformable part models. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2012. 2, 3, 7, 8

[19] Y. Xiang and S. Savarese. Estimating the aspect layout of object categories. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2012. 1, 3, 8

[20] Y. Yang and D. Ramanan. Articulated human detection with flexible mixtures-of-parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99:1, 2012. 2

[21] X. Zhu and D. Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2879–2886. IEEE, 2012. 2, 7