A Study of 3D Deformable Parts Models for Detection and Pose-Estimation

University of Bern

Simon Jenni

March 31, 2015

Abstract

This work covers the implementation of a 3D Deformable Parts Model and provides experimental evaluations of several design choices, model parameters and training-set configurations. A comparison to similar models then shows a performance improvement compared to other 3D DPMs. It also covers the difficulties and methods for incorporating features obtained by Convolutional Neural Networks into a 3D DPM and shows that without fine-tuning of the feature weights, there is no performance gain on cars compared to the standard HOG features.

Acknowledgements

Foremost I would like to thank my supervisor, Professor Paolo Favaro. His guidance, feedback, constructive criticism, and ideas helped me greatly throughout the work on this project. It was his lecture on Machine Learning that got me interested in this field and I'm very glad for the opportunity he gave me to write my Bachelor's thesis on this exciting topic. I would also like to thank Attila Szabó who always provided me with great help and advice when I needed it and also taught me many new things. The meetings with Professor Favaro and Attila were always a source of inspiration and motivation and offered me many new insights into the field.

I would also like to thank my family and friends who supported me and provided me with pleasant distractions from the work on this thesis. I'm especially grateful for the moral and financial support of my father without whom I would not have been able to do this.

A special thanks goes to my very understanding and patient girlfriend Céline who managed to keep up with me during this time and helped me greatly throughout the writing of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

One of the main objectives in computer vision is that of recognising instances of an object category (e.g. cars, pedestrians, faces,...) in images or videos. This has many useful applications like cameras that automatically bring faces into focus, video surveillance systems that track certain objects, traffic monitoring, and roboter vision to name just a few. When given an image the goal of a typical object detection system is to list whether one or several instances of a pre-trained object class are present, and to indicate its position in the image via a rectangular bounding box, tightly covering the object instance (see figure 1.1).

Object category recognition remains one of the most challenging goals of computer vision. While for us humans it is mostly a trivial and effortless task, teaching a computer to recognise objects is very challenging. Given an image, all the computer initially "sees" are numbers referring to the RGB values of each image pixel. Today, the arguably most important step towards teaching machines how to see is to translate these pixel values into more meaningful and task-suited representations (features) of an image.

An object recognition system has to deal with a lot of variation between object classes (inter-class variability) as well as inside of object classes (intra-class variability). Cats for example are very different from cars in that they are non-rigid, have a furry texture, etc. and the system should be able to identify and discriminate between both. A recognition system for cars has to be able to detect cars of all sorts of different sizes and types and should

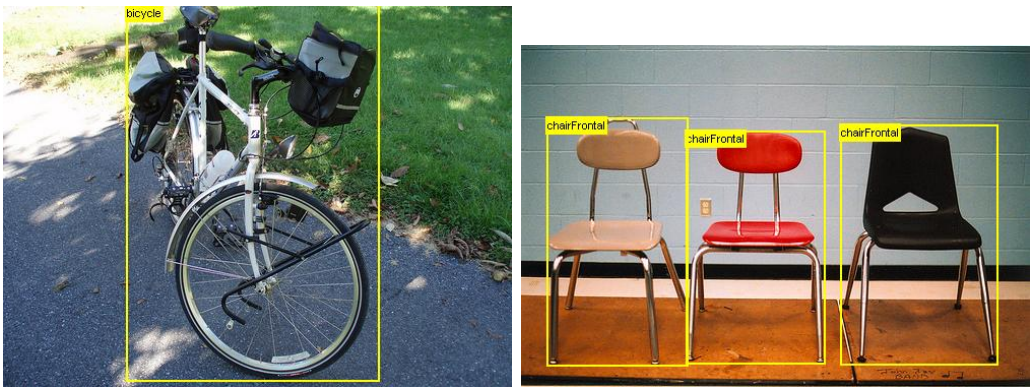


Figure 1.1: Examples from the Pascal Visual Object Classes (VOC) challenge 2012. [6]

also be able to recognise them from any viewpoint.

One of the most widely used and successful approaches to object recognition in the last decade was the Deformable Parts Model (DPM) [10]. It combines effective scale-invariant Histograms of Oriented Gradients (HOG) features [4] with a deformable geometric model and mixture models that allow it to model intra-class variation effectively for many object classes. Current state of the art results in object detection are achieved by systems based on Convolutional Neural Networks (CNN) [15], a technique that made an impressive comeback thanks to the possibilities of GPU-programming (which made efficient training possible) and the existence of large datasets [21].

Very often, more information than just the presence and location of an object would be useful. To arrive at a better scene understanding, we might, for example, like to know the viewpoint that an image of an object was taken under. For this reason, there was a recent growth of interest for 3D object representations. The arguably most successful 3D models today are based on the DPM and have delivered impressive results in viewpoint recognition while upholding and in some cases even increasing 2D bounding box localisation [28, 29]. The extension of a DPM to 3D is usually done by anchoring object parts in three-dimensional space and modelling a discrete number of viewpoints. Modelling objects under a set of discrete viewpoints requires to have viewpoint information provided in the training data and introduces additional supervision to the training procedure. It is interesting to note however that a normal 2D DPM very often learns objects under specific

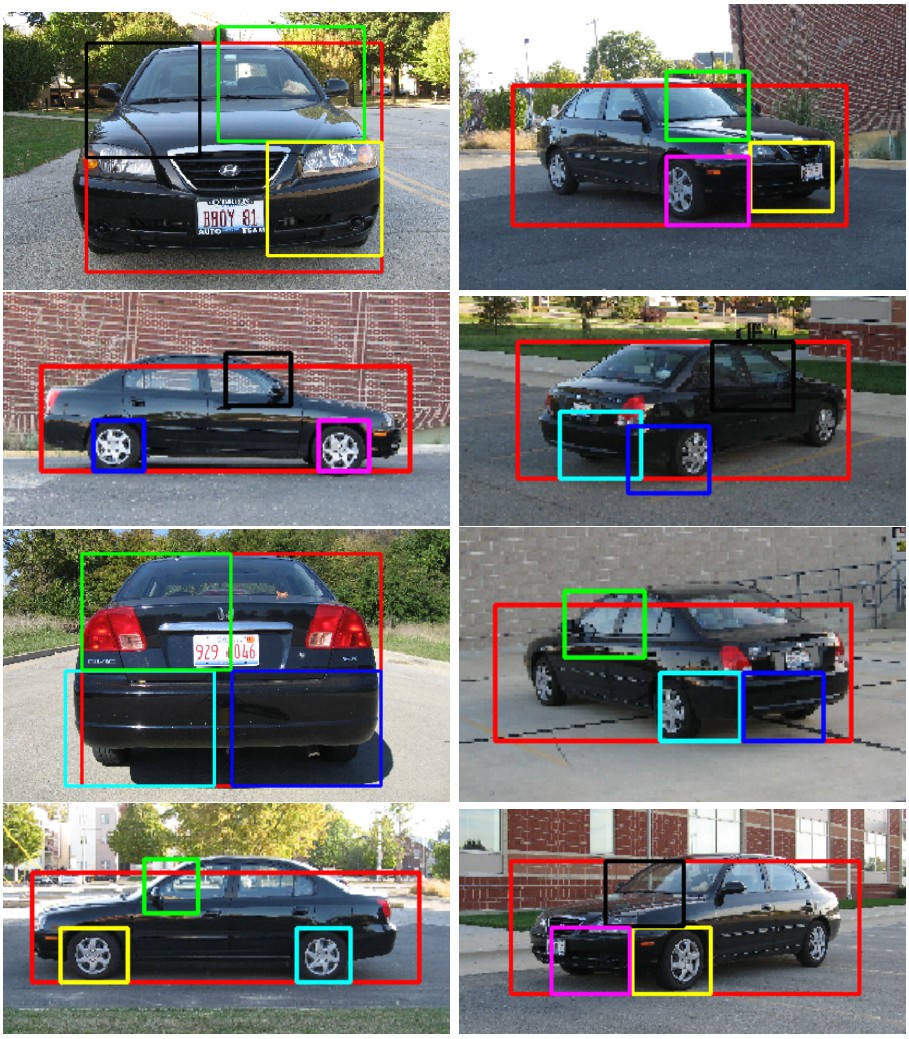


Figure 1.2: This figure shows detections of an eight-viewpoint-3D-DPM with six 3D-parts and three parts per viewpoint. The red boxes indicate the root filter placement and the other boxes indicate part placements. Each 3D part is assigned a separate colour to help illustrate their consistent 3D placement. Note that this model was trained with a small number of parts for illustration purposes. The images are taken from the 3D Objects Dataset of [32].

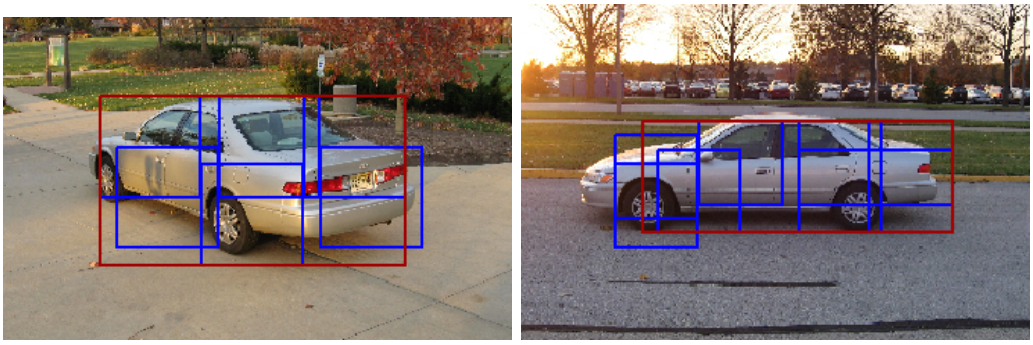


Figure 1.3: This figure shows detections of a normal 2D-DPM. The red boxes indicate the root filter placement and the blue boxes indicate part placements. Note that there is no apparent connection between parts of the two different views.

viewpoints, therefore justifying the approach of the 3D model. The mapping of the 3D part position to the 2D image plane during test-time is done via a viewpoint dependant projective mapping. Part appearances are learned for each model viewpoint separately, with the constraint that they have to be consistent across views. This is important as otherwise the 3D information of the model reduces to basically just a number of different object views. Figure 1.2 illustrates the consistent 3D learning of part appearances on detections of the 3D-DPM implemented in this project. Detections for the same car are shown for eight viewpoints that the model was trained on. In contrast to detections of a usual 2D-DPM (see figure 1.3), which models parts independently for different views, we observe that the parts detected with the 3D-DPM have a 3D geometric relationship. To enforce the 3D constraints on the model, exact viewpoint information for several views of the same object instance are needed. This information is usually provided by renderings of Computer Aided Design (CAD) models.

The aim of this project is to implement and study a 3D deformable parts model for the detection and pose-estimation of cars. The implementation will be based on the successful and openly available implementation of the original DPM [10]. The study will cover the use of new annotated training data, implementation details, an evaluation of model and training parameter choices, experiments on well known and challenging datasets and a comparison to other models. Furthermore, it will also cover the incorporation of

features obtained from Convolutional Neural Networks into the model and how these affect the training procedure and the performance of a 3D-DPM.

1.2 Outline

This thesis is structured into three main chapters. Chapter 2 provides an overview of the research field at hand and gives a review of related work. Also, a brief introduction to the two image-descriptors (features) used in the project will be given. This chapter sets the context for the research in this thesis.

In chapter 3, the model implemented in this project will be introduced. In section 3.1 a review of the original DPM (on which it is based on) will be provided and a formal framework for the model will be set up. Section 3.2 will then explain the modifications made to the DPM to arrive at a 3D object model. Finally, section 3.3 explains the steps taken to replace the HOG features of the original DPM with features obtained from Convolutional Neural Networks.

Chapter 4 shows and discusses experimental results obtained on several datasets and puts them into context with other work. It also discusses the influence of several model parameters on the model's performance on bounding box localisation and viewpoint estimation.

Finally, chapter 5 will conclude the thesis by providing a review of the results and insights gained throughout the project. It will also discuss ideas for future research work building on this thesis.

Chapter 2

Background

In recent years, the task of multiple-view object recognition has gained increasing interest in the computer vision community. The combination of object detection and pose-estimation helps to develop a deeper 3D scene understanding which is crucial to several vision goals.

2.1 A Bit of History

Already in the early days of computer vision, techniques to solve the detection problem were based on 3D modelling. Some of the earliest works in this field were restricted to polyhedral blocks on uniform backgrounds, and approaches relied on edge detectors, line fitting algorithms, and projective projections of feature groupings [30]. The apparent short comings of modelling the world as a "world of blocks" for real world scene reasoning (curves, illumination, complex backgrounds,...) drove forth attempts to model non-polyhedral objects as assemblies of generalised cylinders [1].

The trend in the 1970s was to model objects as collections of 3D volumetric part primitives. These approaches would often be combined with an elastic arrangement model called "pictorial structures" [11] which served as inspiration to the DPM.

The 1980s were dominated by exact 3D shape models inspired by CAD models [24]. A view-based scheme to represent 3D objects gained momentum during this time: aspect graphs [36], a network of distinct 2D views of an object.

In the 1990s a paradigm shift from 3D shape models to image based ap-

pearance models happened [26]. These view-based models made recognition of arbitrary complex objects possible for the first time (only specific objects that the system was trained on of course). The models were still not scale nor viewpoint invariant though.

To overcome these limitations there was a shift from global view appearance representations to localised representations (parts) that are invariant to scale, translation, rotation, illumination, etc. in the 2000s. These methods were made possible by the development of local scale-invariant features like SIFT [25] and HOG [4].

2.2 3D Approaches

Attempts to solve multi-view object class detection can roughly be broken down into three groups. The first group builds upon existing 3D object models (CAD for example [12]). In [39] a 3D feature model is constructed by holographically mapping 2D features from training views to a 3D CAD shape model. [18] follows a similar approach using a rough 3D model which gets assigned parts that are consistently localised across views and then learning appearance over these parts. Another approach based on 3D CAD models comes from [23]. Here, CAD models are used to extract features via a filtering procedure which are then used to construct 3D-filter maps to represent the objects appearance and 3D structure.

The second group consists of 2D object detectors that are combined into a multiview detection system. A first step in this direction was taken by [35]. Those 2D object detectors became increasingly powerful through the introduction of new learning procedures and the use of object part relations and multi scale pyramids [9]. The DPM [10] can be seen as a member of this group, although the connection from mixture component to viewpoint is not explicit. A well performing example of this group is given by [28] (DPM-VOC+VP) where the mixture components of the DPM are connected to discrete viewpoints by training the model on renderings of viewpoint annotated CAD models. In [27], viewpoints are estimated after fine-tuning an initial bounding box assumption and eventually, a classifier is evaluated on both, the estimated bounding box and viewpoint.

The third group consists of methods to build 3D representations dynamically based on initial viewpoint annotated training data. In the work of [32] and [33] object categories are modelled by viewpoint invariant parts that are

connected through homographic constraints.

2.2.1 Related work

The 3D Deformable Parts Models (3D-DPM) studied in this thesis can be seen as a mix between group 1 and 2. They build upon one of the most successful 2D object detector to date, the DPM [10]. This way, they are already predestined to perform well in 2D bounding box localisation. Different viewpoints are modelled by a discrete number of different mixture-components. Where they differ from examples of group 2 is in that they model the parts in 3D and place 3D constraints on them.

The first 3D-DPM was introduced in [28] (DPM-3D-Constraint). They place parts in 3D and map them to 2D via an orthographic projection. By relying on CAD data during training, they learn part-appearances consistently across views. The part-deformations are modelled independently for each viewpoint but are of course also consistently learned thanks to the 3D constraints. The learning problem is rephrased as a structured output prediction problem and the Latent-SVM (L-SVM) formulation of [10] replaced by a Structured-SVM (SSVM) objective aimed at optimising for both bounding box overlap and correct viewpoint estimation.

The model of [29] builds upon [28]. They introduce a 3D deformation model for the parts, modelling the deformation by a 3D gaussian distribution. While making the model slightly more compact through a reduction of parameters that have to be learned, this showed to slightly decrease performance compared to [28]. In order to recognise viewpoints at a finer scale, [29] interpolate between the filters of the model at test time. The interpolation factors are proportional to the angular distance between the viewpoint that is being tested and the viewpoints the model is trained on.

The model implemented during this project is most similar to [28]. It also relies on CAD data during training but uses photo-realistic renderings instead of gradient-based, non-photorealistic ones as in [28] and [29]. During training, it also makes use of viewpoint annotations on Pascal VOC 2012 provided by [38]. Instead of an SSVM objective, the model uses the L-SVM formulation of [10] and introduces a penalty term during latent-positive search. Also, this project experimented with the use of Convolutional Neural Network features as a replacement for the HOG features, usually used in the DPM.

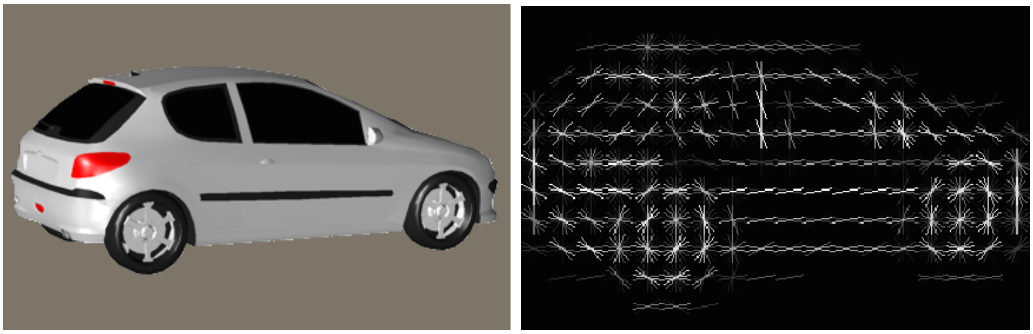


Figure 2.1: Visualisation of a HOG feature map (right) computed from the rendered car image on the left.

2.3 Features

This section gives a brief introduction to the two image features used in this project.

2.3.1 HOG

Histograms of Oriented Gradients (HOG) have been introduced in 2005 by Dalal and Triggs [4] and showed remarkable results in pedestrian detection. HOG descriptors divide the image into rectangular cells, compute gradient directions of all the pixels in the cell, and compile a histogram of gradient directions after binning the gradient-votes into typically nine orientation bins. The weight of each gradient vote is given by the gradients magnitude. This accumulation of pixel gradient votes into cells already ensures a small degree of translation-invariance.

The computed cells are then grouped into overlapping blocks of 3×3 cells and local normalisation is performed in each of the blocks. This block normalisation makes the HOG features invariant to changes in contrast and illumination. Each cell leads to a 36-dimensional feature vector. A visualisation of a HOG feature map computed on a rendered CAD car model can be seen in figure 3.4.

HOG features, with some modifications, are the typical image descriptors used in DPMs. In [10] Felzenszwalb et al. proposed a modification of the HOG features which computes both, directed and un-directed gradients and

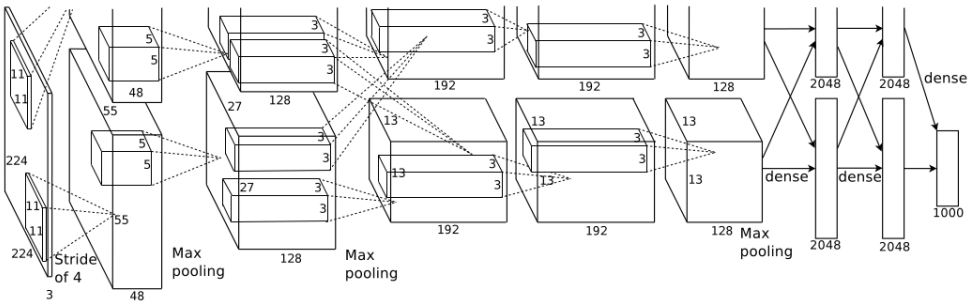


Figure 2.2: Figure showing the architecture of the CNN of Krizhevsky et al. [21]

reduces the dimensionality of the features by projecting them on a 31 dimensional subspace. This variant of the original HOG feature is the standard image-descriptor used in 3D-DPMs.

2.3.2 Convolutional Neural Networks

Recent state of the art performance in image classification and object detection have been reported by methods relying on Convolutional Neural Networks (CNN) [21][15]. CNNs are inspired by biological mechanisms of the visual system and were first introduced in 1980 by [13]. Their design has since been refined and first promising results on document recognition were reported by LeCun [22] in 1998 with the introduction of LeNet5. In 2012 Krizhevsky et al. [21] demonstrated the power of CNNs on image classification by showing a substantial improvement in accuracy compared to previous methods.

The architecture of the CNN from [21] can be seen in figure 2.2. CNNs are build out of several types of layers:

Convolutional Layers: These consist of several convolution kernels per layer (96 in the first layer of [21]). The image is convoluted with each kernel and the result passed on to the next layer. The convolution kernels are not fixed but learnt during training.

ReLu Layers: The ReLu layer introduces non-linearities into the CNNs and consists of a function that is applied to each of the layer's inputs. The

ReLu function is $f(x) = \max(0, x)$, which has shown to considerably reduce training time compared to other functions typically used to model a neuron's output (e.g. $f(x) = tanh(x)$).

Max Pooling Layers: These layers partition the input into a set of rectangular subregions and outputs the maximum value of each subregion, therefore ensuring small translation invariance of the features.

Fully Connected Layers: These are the last few layers of a CNN. In these layers all the layer's neurons are connected to all the next layer's neurons. Here is where the high level reasoning of the CNN happens.

As mentioned before, the filters of a CNN are trained rather than handcrafted. CNNs, as feed-forward networks, are trained using backpropagation. Backpropagation works by back propagating the partial derivatives of the error along the layers. First, the partial derivative of the error with respect to the last layers weights is computed and the weights updated before the partial derivative for the second-to-last layer is computed and so forth. Once partial derivatives for all the weights have been computed, gradient descent can be used to learn the network.

There have been very recent reports on the incorporation of features from Convolutional Neural Networks into DPMs by [16] and [31]. They both report a substantial improvement in mean average precision on VOC 2007. It is important to note however that cars are an exception to this rule, as CNN feature-based DPMs did not improve on HOG feature-based DPMs on cars. Both [16] and [31] use the first five layers of the network of Krizhevsky et al. [21] but use fine-tuned weights as provided by [15]. While [16] do not use the detection fine-tuned weights of [15], it is unclear which weights were used by [31].

One aim of this project is the incorporation of CNN features into a 3D-DPM, and to study how it affects the performance and training of the model. While the detection performance on cars is likely to decrease compared to a HOG based 3D-DPM as in [16], it is unclear how CNN features affect the model's performance on pose-estimation.

Chapter 3

The Deformable Parts Model

This chapter will give a review of the Deformable Parts Model as described in [10] and its implementation (VOC-release 3) on which this project builds upon. Section 3.2 will then introduce the modifications that were made to extend it towards 3D. Wherever possible notation used in [10] and [28] will be adopted.

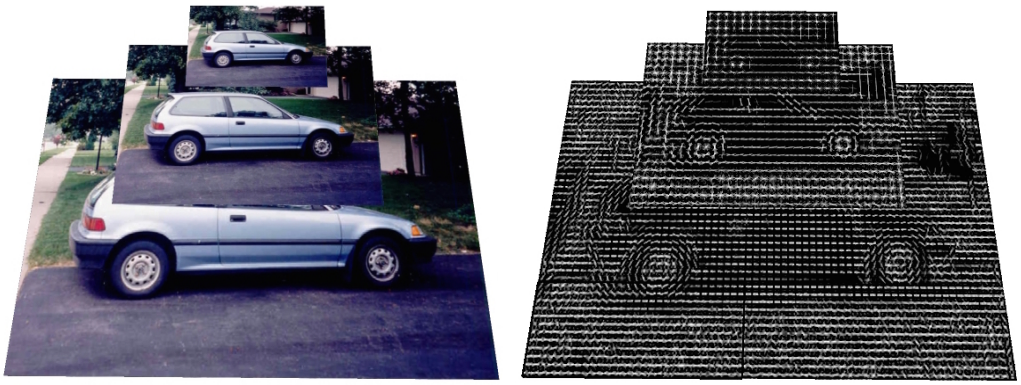
3.1 DPM Review

The DPM is a type of sliding window detector, consisting of a set of linear filters which are applied to a feature pyramid computed from an input image. Each DPM has a root filter which models the object appearance as a whole, and several higher resolution part filters which model smaller parts of the object.

3.1.1 Sliding Window Approach

The idea behind the sliding window approach is that, given an input image and a window (of smaller size than the image), the window is placed on all locations inside the image and for the region covered by the window to decide whether the object is present or not.

Rather than working with images directly, feature maps are computed from them. A feature map G is a two-dimensional array of d-dimensional feature vectors. One can think of each feature vector as representing a small image patch and the feature map as a grid of feature vectors computed from



(a) Image pyramid (b) HOG feature pyramid

Figure 3.1: These figures illustrate image and feature pyramids. The pyramids here span two octaves with one level each. Therefore each pyramid level downwards doubles the image resolution. Note that the cell-size of all the feature maps in the HOG pyramid is constant.

image patches. The features used in [10] are a modified version of the HOG features introduced in [4]. For now let us look at the case of a model consisting of a single root filter (e.g. a DPM without parts). A root filter F is a two-dimensional array of d-dimensional weight vectors. The root filter can be interpreted as the "window" that slides through each position on the feature map to detect objects.

To detect objects at different scales, a feature pyramid is used (see figure 3.1 and the appendix for a detailed description). The subsampling factor from level to level is constant and determined via a parameter λ which fixes the number of levels in an octave. A feature map an octave beneath in the pyramid is computed at twice the resolution.

Each placement of the filter in the feature pyramid is assigned a score via the "dot"-product of the filter and the corresponding sub-array of the feature map at the given level. More formally: given a filter F of size $w \times h$, a filter placement in the feature pyramid is defined by a three-tuple $p = (x, y, l)$ where (x, y) specifies the position of the upper-left corner of the filter in the l-th level of a feature pyramid H. Further let $\phi(H, p, w, h)$ denote the vector obtained by concatenating the feature vectors in H, belonging to a window of size $w \times h$ at position p in H. The score of this placement is then given

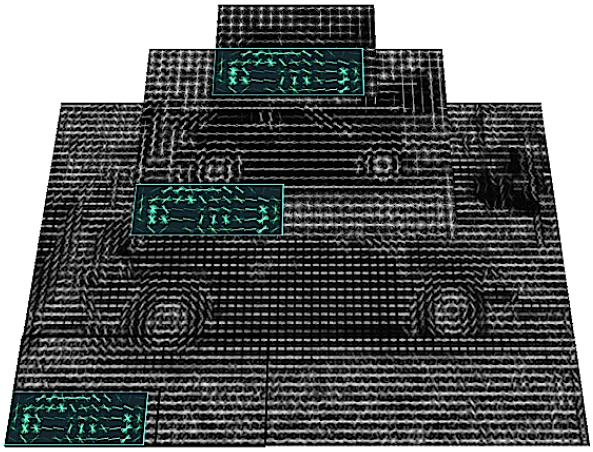


Figure 3.2: This figure illustrates the instantiations of a filter in a feature pyramid. The same filter is placed in the lower left corner of each pyramid level. To better distinct it from the feature maps, the filter is coloured.

by:

$$score(p) = F' \cdot \phi(H, p) \tag{3.1}$$

where F' denotes the concatenation of the weight vectors in F and $\phi(H, p)$ is a shorthand for $\phi(H, p, w, h)$, as w and h are implicit from F.

3.1.2 Deformable Parts and Mixture Models

We will now extend the root-only model with parts and a deformation model. Let us consider the case of a DPM model for cars. The root filter approximately covers all of the car and models the object appearance as a whole. It also serves as a detection window. Part filters cover smaller, more detailed parts of the car, e.g. the tires or windows, and model their appearance at a higher resolution. The part filters are always placed λ levels (an octave) below the root filter in the feature pyramid and are therefore computed at twice the resolution. Each part is placed at a fixed anchor-position relative to the root filter's position. This leads to a star-shaped model. To model intra-object class variations, parts are allowed to deviate from their anchor positions by a certain amount restricted by a deformation cost.

Formally, a model with n parts is defined by $(F_0, P_1, ..., P_n, b)$ where F_0 is

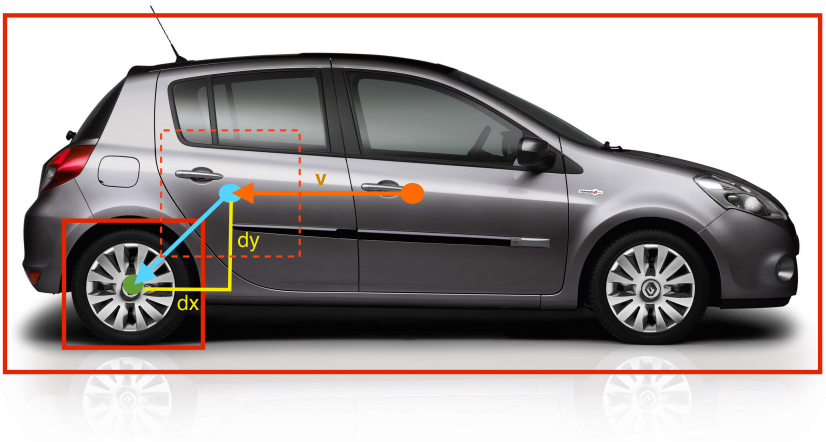


Figure 3.3: This figure illustrates the anchors and displacements of the parts. The anchor position v from the root centre to the anchor is coloured in orange. The displacement from anchor to the part position is coloured in light blue.

the root filter, P_i is a model for part i, and b is a bias term. P_i in term consists of (F_i, v_i, d_i) where F_i is the part filter, v_i specifies the anchor position, and d_i is a four-tuple of coefficients for a quadratic function modelling the deformation cost. A placement $z = (p_0, p_1, ..., p_n)$ of all the filters in the feature pyramid is called a hypothesis. The displacement of the parts relative to their anchor position is given by $(dx_i, dy_i) = (x_i, y_i) - (2(x_0, y_0) + v_i)$. Remember that the parts are computed at twice the resolution of the root, hence the factor of two. Figure 3.3 illustrates part anchors and deformations. Given the displacements, the deformation feature is set to $\phi_d(dx_i, dy_i) = (dx^2, dx, dy^2, dy)$. A hypothesis is assigned a score via

$$score(z) = \sum_{i=0}^{n} F_i' \cdot \phi(H, p_i) - \sum_{i=1}^{n} d_i \cdot \phi_d(dx_i, dy_i) + b. \tag{3.2}$$

Note that the first term of the sum is concerned with the appearance and the second term with the deformation.

To arrive at richer object models, several DPMs, as just described, are combined into one mixture model. A mixture model is an m-tupel $M = (M_1, ..., M_m)$ where M_c is the model for mixture-component c. An object hy-

pothesis for a mixture model is given by $z = (c, p_0, ..., p_{n_c})$ where n_c specifies the number of parts in M_c and the score for hypothesis h is simply the score of $z' = (p_0, p_1, ..., p_{n_c})$ for M_c.

By concatenating all the model-parameters of M_c into a vector

$$\beta_c = (F_0, F_1, ..., Fn, d_1, ...d_n, b) \tag{3.3}$$

the whole mixture model can be expressed through the vector $\beta = (\beta_1, ..., \beta_m)$. By similarly concatenating all the features of hypothesis z' into a vector

$$\Phi(H, z') = (\phi(H, p_0), ..., \phi(H, p_n), -\phi_d(x_1, y_1), ..., -\phi_d(x_n, y_n), 1) \tag{3.4}$$

a sparse feature vector for hypothesis h can be constructed by setting

$$\Phi(H, z) = (0, ..., 0, \Phi(H, z'), 0, ..., 0). \tag{3.5}$$

The score of the hypothesis h can then be expressed via the dot product

$$score(z) = \beta \cdot \Phi(H, z). \tag{3.6}$$

3.1.3 Inference

As the root filters act as detection windows, the problem of object detection reduces to finding high scoring root placements. The score of a root placement for a mixture component is given by

$$score(p_0) = \max_{p_1, ...p_n} score(p_0, p_1, ...p_n). \tag{3.7}$$

That is, for each root placement we want to find the best placements of all the parts. In practice, this is solved by first pre-computing the filter responses and storing them in an array $R_{i,l}(x, y) = F_i' \cdot \phi(H, (x, y, l)$ where i specifies the filter and l the level in the feature pyramid. The filter responses are then transformed via

$$D_{i,l}(x, y) = \max_{dx, dy}(R_{i,l}(x + dx, y + dy) - d_i \cdot \phi(dx, dy)). \tag{3.8}$$

$D_{i,l}(x, y)$ gives the maximum contribution of part i to a root filter placed in the feature pyramid so that the anchor position of part i is at (x, y, l). Given the array of filter responses, the maximum in equation 3.8 can be computed in linear time using the efficient distance transform of sampled functions described in [8]. The score 3.7 can now be expressed as

$$score(x_0, y_0, l_0) = R_{0,l_0}(x_0, y_0) + \sum_{i=1}^{n} D_{i,l_0-\lambda}(2(x_0, y_0) + v_i) + b. \tag{3.9}$$

Bounding Box Prediction

To increase the accuracy of the detected bounding boxes, a technique called bounding box regression is used. The idea is to make use of the additional information provided by the part placements to refine the predicted bounding boxes. For one, parts allow to model intra-class variability better (e.g. a long car vs a short car) and parts are matched more precisely as they are modelled at twice the resolution of the root filter.

After training a model, a separate bounding box predictor is trained. Let $g(z)$ be a feature vector obtained from the object hypothesis z containing the width of the root filter in image coordinates and the upper left corners of each filter in image coordinates. The bounding box predictor is a linear function $b(g(z)) = (x_1, y_1, x_2, y_2)$ of $g(z)$. The outputs of the function are the upper-left (x_1, y_1) and lower-right corners (x_2, y_2) of the predicted bounding box. The function b is learnt using least-squares regression.

Non-Maximum Suppression

Using the bounding box prediction procedure as it stands now, typically leads to multiple overlapping detections as the model would detect the same object on locations that are spatially close to each other and have a similar scale. As simply choosing the highest scoring detection for each image would discard several correct detections on images that contain multiple object instances, another method to reduce detections is needed.

With non-maximum suppression, detections are first sorted according to their score and the highest scoring detection is chosen. Greedily, more detections are chosen while skipping detections that overlap by more than 0.5 with a previously chosen detection.

3.1.4 Training

For training, a set of training examples $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ is given. The x_i's are images and $y_i = (y_l, y_b)$ are labels where the label $y_l \in \{-1, 1\}$ specifies whether the object is present or not and $y_b = (x_{min}, y_{min}, x_{max}, y_{max})$ specifies a bounding box around the object in image coordinates.

Let $Z(x, y)$ denote the set of possible latent hypothesis values for an example x and labels y. For positive examples, $Z(x, y)$ is restricted to hypotheses which yield a root placement that overlaps with the bounding box label by

at least 0.7. For negative examples or during testing, no restrictions are set on $Z(x)$. Given a model β, equation 3.6 allows to train a linear classifier

$$f_\beta(x) = \max_{z \in Z(x,y)} \beta \cdot \Phi(x, z) \tag{3.10}$$

using a formalism called Latent-SVM introduced in [10] which is mathematically similar to MI-SVM [2]. Latent-SVM minimizes the objective function

$$L_D(\beta) = \frac{1}{2}\|\beta\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i f_\beta(x_i)) \tag{3.11}$$

in analogy to standard linear SVMs with standard hinge loss. This optimisation problem is semi-convex that is, given fixed latent variables for positive examples, it becomes convex.

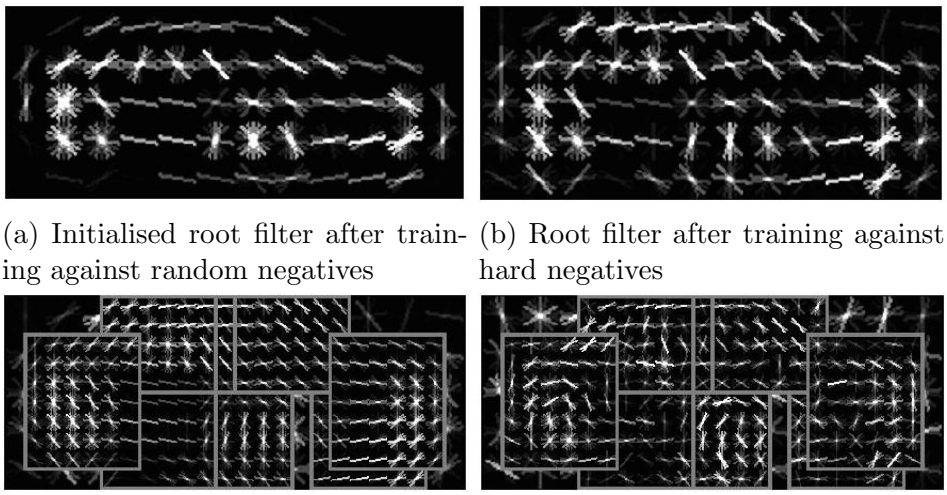
In practice equation 3.11 is solved by repeatedly:

1. Constructing a training set $D(Z')$ where Z' fixes latent variables for both positive and negative examples while keeping β fixed. Latent variables of positive examples are fixed by finding $\text{argmax}_{z \in Z(x,y)} \beta \cdot \Phi(x, z)$. For negative examples, hard negative mining is used. Concretely, we search for examples with $\beta \cdot \Phi(x, z) > -1.05$.

2. Updating β by solving 3.11 on the constrained set via a gradient descent algorithm using sub-gradients similar to [34].

3.1.5 Initialisation

An n-component model is initialised by first splitting the positive training examples according to aspect ratio statistics in n training-sets, and then training root filters for each component. Root filter dimensions are chosen in a way that their covered area is not greater than 80% of the positive bounding boxes. Root filters are trained by warping positive bounding boxes to filter dimensions and training against negative examples obtained by extracting features from randomly selected regions of negative training images. Once all the root filters are initialised, the model is trained using several iterations of hard negative mining. Note that at this stage, only the component variable and root placement is treated as latent.



(a) Initialised root filter after training against random negatives

(b) Root filter after training against hard negatives

(c) Mixture component after parts have been initialised

(d) Mixture component after training against hard negatives

Figure 3.4: This figure visualises the evolution of a mixture component of an eight viewpoint 3D DPM during the training procedure.

The part filters are initialised by placing them on high "energy" regions, that is, regions with high euclidean norm of positive root filter weights. Possible part shapes are chosen from a pool of shape-templates and greedily added to the model. Regions covered by a part are zeroed out in the energy map and the next part is chosen until a fixed number of k parts are added. In [10] symmetry constraints on both the root and parts, are placed. Root filters are constrained to be symmetric along the vertical axis, as are parts lying on the centre vertical axis. Parts not lying in the centre have a symmetric partner.

3.2 Extension to 3D

To extend the DPM to 3D, this project largely followed the approach of [28] and [29]. Training a 3D DPM relies highly on having viewpoint annotated training data. The model presented here is trained by using both real and synthetic training examples of cars. The synthetic examples were obtained by renderings of CAD models onto negative training images or backgrounds

Figure 3.5: A CAD example rendered on a negative training image including the annotated bounding box (left). And a CAD example rendered on a background of mean pixels (right). Both were rendered using perspective projection.

of mean pixel colour of the training images (see figure 3.5). Experiments with both, orthographic and perspective projections for the rendering were done. Both [28] and [29] rely on rendered CAD models as well, but they use non-photorealistic gradiant-based renderings and projective projections. For real training data, positive examples from the 3D object dataset [32] and examples from VOC2012 [6] with viewpoint annotations provided by [38] were used.

All the symmetry constraints placed on the root and parts from the original DPM [10] were eliminated as they do not allow to distinguish different views (mirrored left and right side views of a car look identical). Figure 3.4 shows the evolution of one model view during the training procedure.

3.2.1 Assigning Components to Viewpoints

The first modification to the model was to relate mixture components to specific viewpoints. To achieve this, the training data D was split according to viewpoint annotations into n_c component training sets $\{D_i\}_{i=1}^{n_c}$ and root filters were initialised on each of these sets by training against random negatives (see figure 3.4a). Three parameters control the splitting of the training set:

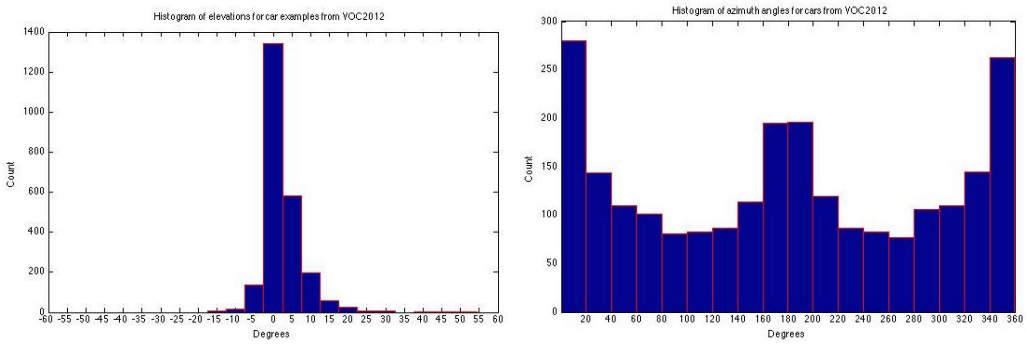


Figure 3.6: Histograms showing the distribution of elevations (left) and azimuth angles (right) across car examples from VOC2012 as annotated by [38].

- n_c: specifies the number of mixture components.
- n_a: specifies the number of different azimuth angles.
- n_e: specifies the number of different elevations.

Note that $n_c = n_a \cdot n_e$. Viewpoints are split evenly across components, i.e. a model with $(n_c, n_a, n_e) = (16, 8, 2)$ would have eight azimuth angles lying in $\{0, 45, 90, 135, 180, 225, 270\}$. In this project a setting of $n_e = 1$ was most often used while only varying n_c. Setting $n_e = 2$ doubles the complexity of the model while only giving slight, if any, improvement in performance. Also, viewpoint accuracy as measured in [38] only considers correct azimuth angle binning and example images from from VOC2012 [6] typically don't vary much in elevation (see figure 3.6).

During latent training, if a positive example falls within an angle $\Delta\alpha$ of a viewpoint assigned to a mixture component c, the latent component label is restricted to c. This has the nice side-effect of speeding up training as we do not have to search over all the components. Choosing $\Delta\alpha$ leads to a trade-off between precision and viewpoint accuracy. A value of $\Delta\alpha \in [10, ..., 20]$ was found to perform well.

Another way of enforcing the correct component label during training is to introduce a penalty for incorrect viewpoint predictions. Formally, the labels of positive examples are extended by a component label y_c leading to $y = (y_l, y_b, y_c)$. During latent positive search, latent variables would then be

set to:

$$z' = \underset{z \in Z(x,y)}{\operatorname{argmax}}(\beta \cdot \Phi(x, z) - l(y_c, c)) \tag{3.12}$$

where $l(y_c, c) = \mathbb{1}[y_c \neq c] * \gamma$ is the penalty term and the parameter γ controls the weight of correct viewpoint assignments. The value of γ was set to $\gamma = \max(1, \beta \cdot \Phi(x, z)/2)$. This way, very confident hypotheses are penalised more if they lead to wrong viewpoint estimates. Experiments showed that this generally doesn't decrease detection performance while increasing viewpoint accuracy.

3.2.2 Placing Parts in 3D

Going from 2D to 3D, anchor positions should now be placed in 3D object space rather than in the 2D root filter. As a reference coordinate system, a 3D object box is created and the origin is placed at the centre of the 3D box (see section 3.2.3 for details). Analogous to [28], parts are parametrised by part boxes $b_i = (s_{x_i}, s_{y_i}, s_{z_i})$ co-aligned with the object box. The new anchor position $v_i = (x_i, y_i, z_i)$, now in 3D, specifies the center of the part box in object coordinates.

Part filters remain 2D templates of weight vectors and are sized so that they tightly cover the 3D part box as seen from the viewpoint of each mixture component. Mapping the 3D anchor positions to root filter coordinates is done by using an orthographic projection P_c given by the viewpoint of component c.

To make sure that part filters are learned consistently across views, latent part positions have to be inferred in 3D and simultaneous for all views of a given car model. This is achieved by making use of the CAD training data. Concretely, positive CAD examples are labeled with a model-ID y_o and a set $D(i) = \{(x, y) \in D : y_o = i\}$ is created. As CAD examples come with perfect annotations, the root placement and component variable are fixed. The root placement p_c of component c is chosen such that it yields maximum overlap with the annotated bounding box y_b. An object hypothesis in 3D is now defined by $h = (q_1, \ldots, q_n)$ where $q_i = (x_i, y_i, z_i)$ specifies the position of part i in object coordinates. Note that for model component c we get $z = (c, p_c, P_c(q_1), \ldots, P_c(q_n))$, the connection to the 2D hypothesis. Parts are still fixed to be an octave below the root filter in the feature pyramid. Let $\Psi(x, y, h) = \Phi(x, z(y, h))$, where $z(y, h) = (y_c, p_c, P_c(q_1), \ldots, P_c(q_n))$. The

objective is now to find:

$$h' = \underset{h}{\operatorname{argmax}}(\sum_{(x,y) \in D(i)} \beta \cdot \Psi(x, y, h)) \tag{3.13}$$

Note that part displacements are still modelled in 2D (as in the model of [28]) and are only dependent on other views via equation 3.13. This is in contrast to [29] where displacements are modelled in 3D. While the 3D deformation model increases the models performance on ultra-wide baseline matching experiments, performance decreased for both detection and viewpoint estimation.

For each car model i the procedure to solve 3.13 is the following:

1. **Step** Construct a 3D grid of possible part placements by padding the 3D object box to all sides. Padding is used to allow parts to move out of the 3D object box.

2. **Step** Compute feature maps for root and part filters for all examples $(x, y) \in D(i)$. The feature maps are computed so that they cover the whole 3D grid as seen from viewpoint y_c and the fixed root placement yields maximum bounding box overlap.

3. **Step** Iterate each part p_j through each position q_i of the 3D grid and sum up the scores for placing part p_j at position q_i for all the views. For a view related to component c, the score is given by the score of placing it at $P_c(q_i) = (x_i, y_i)$ in root filter coordinates. That is, $score(p_j, q_i, c) = F'_j \cdot \phi(H, P_c(q_i)) - d_j \cdot \phi_d(dx_j, dy_j)$. Note that here H denotes the feature maps computed in step 2 and not a feature pyramid as in section 3.1.2. The total score for placing p_j at q_i is then given by $score(p_j, q_i) = \sum_c score(p_j, q_i, c)$.

4. **Step** Pick the position that yields the highest overall score for each part. We set $h' = (\operatorname{argmax}_q score(p_1, q), \ldots, \operatorname{argmax}_q score(p_n, q))$.

Positive training examples for which no model-ID is provided, are treated as before, i.e. projecting the 3D anchor position to 2D and then searching for the best hypothesis in 2D.

3.2.3 Initialising Parts in 3D

The initialisation of parts in 3D follows a similar heuristic as the 2D DPM by greedily placing them on high energy regions in 3D. For initialising a model with m parts, the procedure is the following:

1. **Step** First the 3D object box is created which will be used to set up a grid of possible part placements. The box is modelled as 3D array of zeros. The dimensions of the box are initialised from the dimension of front and right view root filters. The height is set to twice the height of the front filter. The front and right view dimensions are then warped to fit the height of the box while preserving the aspect ratios.

2. **Step** Next, a pool of possible 3D part boxes $b_i = (s_{x_i}, s_{y_i}, s_{z_i})$ is created. Each part box is sized so that its volume approximately covers $1/m$ of the object box volume, concretely: $V_{part} = 0.8 * \frac{V_{object}}{m}$.

3. **Step** Each part box is then convolved with the 3D object box to get a grid of possible part placements. The part box is then moved through the grid and each grid position is assigned an energy. The energy of a grid position $q = (x, y, c)$ is obtained by projecting q onto the root filters of all the components and then summing up the norm of positive root filter weights covered by the part box for all the components.

4. **Step** The highest scoring shape and position combination is then chosen as new 3D part. To get non overlapping parts, regions covered by the new part are set to -100 in the 3D object box. This way, convolving the object box with the part shapes in step 3 gives grid positions that lead to overlapping parts a negative energy.

Repeat Step 3 and 4 until m parts are chosen

As not all of the 3D parts are visible from all the views, only a maximum of k parts are assigned to each view. For each view, parts are chosen greedily based on both their energy on the root filter and their depth in the 3D object box as seen from this view (parts in front are preferred). When a 3D part gets chosen, the region it covers is zeroed out in the root filter energy and a next part is chosen until either all k parts are chosen or no more regions of the root energy can be covered by other parts (Note that different components might now have a different number of parts assigned to them). Part filters

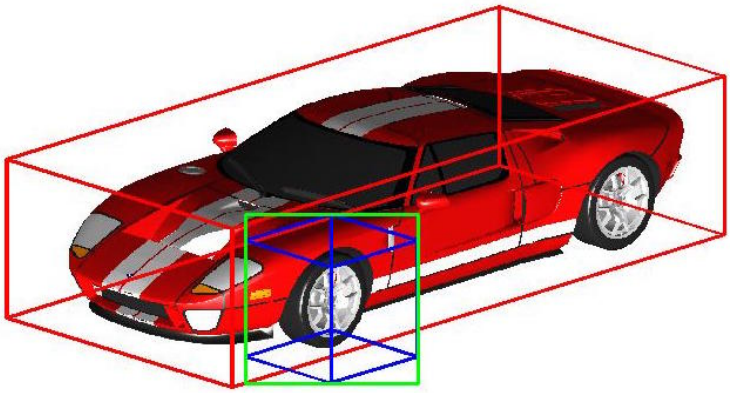


Figure 3.7: This figure illustrates the idea of the 3D object box and the part parametrisation. The blue box represents a 3D part box and the green 2D box represents the part filter template from the given viewpoint.

are chosen such that they tightly cover the 3D part box as seen from that view (see figure 3.7).

3.3 CNN Features

This section covers the steps taken to replace the HOG features of the original DPM with CNN features. The features used in this project are computed using the MatConvNet toolbox [37]. The concrete net used was the fast CNN-F architecture described in [3]. The net consists of 5 convolutional layers, some of which are followed by max-pooling layers, and 3 fully connected layers. The architecture used is similar to the one by Krizhevsky et al. [21] with the difference of having dense connectivity between convolutional layers in CNN-F and less convolution kernels in layers 1 (64 vs.96) and layers 3 and 4 (256 vs 384 each).

The standard input of CNN-F is a fixed sized $224 \times 224 \times 3$ image patch which is preprocessed by subtracting the mean training image. Constructing the feature pyramid requires to compute features for several rescaled versions of images of variable size. This project therefore used arbitrarily sized images as input to the CNN and approximated the normalisation by subtracting the

Layer	conv_1	pool_1	conv_2	pool_2	conv_3	conv_4	conv_5
#Filters	64	-	256	-	256	256	256
Size	11×11	3×3	5×5	3×3	3×3	3×3	3×3
Stride	4	2	1	2	1	1	1
Padding	5	1	2	1	1	1	1

Table 3.1: This table shows the architecture of the first five layers of the CNN-F network with the modified padding.

mean pixel value from the input image. This approach was also used and experimentally justified in [19] on the Caffe net [20]. CAD examples were rendered on a background of the same mean pixels (see figure 3.5 right).

Only the first five convolutional layers of the network are used to compute features (also discarding the last max-pooling layer). The output of the fifth convolutional layer given an input image of size $224 \times 224 \times 3$ is a feature map of size $13 \times 13 \times 256$. To get a more convenient mapping from image coordinates to feature map, the inputs of all the convolutional and max-pooling layers with kernel size k are padded by $\lfloor k/2 \rfloor$ with zeros to all sides (see table 3.1). The output of a $224 \times 224 \times 3$ is then a $14 \times 14 \times 256$ feature map and coordinates (x, y) in the feature map, map to $(16x, 16y)$ in image coordinates (16 is the stride of the features computed by the fifth convolutional layer). For HOG features, the mapping was given by $(8x, 8y)$ where 8 is the HOG cell size. This approach is similar to the one of [16] and was found to work slightly better than simply padding the feature maps to obtain the same mapping (used in [19] and [31]).

The feature vectors obtained by the fifth convolutional layer are further standardised, i.e. the mean feature vector is subtracted and an element-wise division by the standard deviation of the feature vectors is performed. This has shown to be very important for the SVM training (see section 4.4). Also, it was found to be important to adjust the SVM parameter C for CNN features. While HOG features work well with a value of $C = 2 \cdot 10^{-3}$, a value of $C = 2 \cdot 10^{-5}$ was found to work better with CNN features. This lower value of C places more focus on maximising the margin of the classifier and reduces the risk of overfitting to the training data.

By observing the detections during latent positive search, another necessary adjustment to the models parameters became apparent. The quadratic

deformation parameters of HOG based DPMs are lower bounded by a value of 0.01 ensuring that the deformation model doesn't become too "flat". This lower bound showed to be too low for CNN features however, as the parts tend to move away very far from their anchor positions (sometimes placed on other cars in the scene) and also hindered the convergence of the hard negative training. This can be explained by the lower influence the deformation features/parameters have on the overall score of a hypothesis. The lower bound was therefore raised to 0.2, which fixed the issues.

Switching from HOG to CNN features brings four significant changes:

1. The subsampling factor from image to feature map is doubled from 8 to 16. If the root filter dimensions are kept unchanged, this implies that the smallest objects that can be detected by a CNN-DPM are twice as large as the smallest detectable objects for a HOG-DPM. To counteract this, the oversample factor of the first level of the feature pyramid is increased from 2 to 4, which makes the feature pyramids equivalent with respect to the root filter sizes (doubled root and feature map sizes).

2. HOG features describe scale invariant local gradient statistics while CNN features describe large, highly overlapping 163×163 image patches. This might be one reason why it is beneficial to use poorly localised positive examples as negative training examples as reported by [16] and [15]. Poorly localised in this case means examples that lead to a ground truth bounding box overlap of less than 0.3.

3. The feature vector dimension increases from 31 to 256. This of course slows down training significantly as well as increases the time needed to compute the convolutions of the model filters with the feature maps during inference considerably. One possibility to counteract the slower training would be to employ the Linear Discriminative Analysis (LDA) based acceleration to DPM training [17]. In this project however, the SVM training procedure has been preserved (allowing a better comparison to HOG features) and training iterations have been reduced as CNN models have been found to converge faster.

4. The feature computation takes much longer on a CPU. This makes the computation of the feature pyramid considerably slower, typically increasing the computation time by a factor of four. In this project

therefore the number of levels per pyramid (λ) was reduced from five to two during training, sacrificing some performance in favour of faster training and inference. [16] reported that this modification still provided good enough feature localisation and experiments conducted in this project also found that increasing λ for CNN features only slightly increases performance. This of course also speeds up inference as much less convolutions are needed.

3.4 Performance Optimisation

As the model presented here is based on DPM VOC-release 3, there is an inherent performance gap to newer versions (e.g. DPMver4 [7] or DPMver5 [14]). To close this gap, several improvements introduced in DPMver4 have been adopted to the model described here.

To speed up training:

- Positive and hard negative search has been implemented in parallel, and convergence criteria were added. Training stops if relabelling doesn't reduce positive loss by much and hard negative mining stops and skips to the next latent positive search if the negative loss on the full negative training set doesn't increase the negative loss by much.

- The C++ training code has been updated to the one used in DPMver4. It features better learning statistics and adds convergence criteria. A new stopping criteria, where training stops if the sum of positive and negative loss don't exceed a certain value ϵ, has also been added. This is only affecting the CNN training however, and should also prevent overfitting to a certain degree.

- For viewpoint annotated data with viewpoints inside a range of α of a model's viewpoint angle, latent positive search has been restricted to examples with the correct component label.

Section 4.1 gives an experimental evaluation of several model parameters and design choices on the models detection and pose-estimation performance. Two of the most significant performance boosts come from adding an extra octave to the feature pyramid and the use of a truncation feature.

Figure 3.8: Figures illustrating problems with the bounding box annotations on Pascal VOC. The red bounding boxes are the annotations and the blue bounding box illustrates the discrepancy to a real car bounding box.

3.4.1 Handling Truncated and Occluded Objects

In experiments on the Pascal VOC 2012 dataset which contains several truncated and occluded objects, the importance of handling truncated and occluded objects effectively became apparent (see figure 3.8). In this project two approaches were tested to improve detection of truncated objects.

To detect truncated objects, all levels of the feature pyramid are padded with zeros by half the root size. In DPMver3 the computation of the bounding box overlap requirement during latent search (0.7 overlap) is computed using the root filter box. This discards a lot of truncated examples or leads to wrong component labels as the correct root placement is prohibited by the overlap requirement. An easy fix for this is to clip the root box of truncated examples to image boundaries before computing the overlap. It showed that detection performance can be improved further by padding the feature pyramid with the mean hog features computed on positive training examples. An even better performing approach is to augment the feature vectors with an additional truncation bias term (used in [7]). This feature term is set to 0 if the feature vector is placed within image boundaries and to 1 if it is outside the image boundaries. This way, the model actually learns which filter regions often are truncated.

Another observation made on occluded examples was that the minimum bounding box overlap requirement of 0.7 during latent positive search would often prohibit the correct viewpoint from being chosen. For occluded examples (indicated through annotation) the minimum bounding box requirement was therefore reduced to 0.5. As un-occluded and non-truncated object in-

stances should be expected to have "good" bounding boxes in the sense that the model's detection should lead to high bounding box overlap, the penalty term from equation 3.12 has been extended with an overlap penalty term leading to:

$$l(y, z) = (0.5 \cdot \mathbb{1}[y_c \neq c] + 0.5 \cdot \frac{A_z \cap A_y}{A_z \cup A_y}) * \gamma \tag{3.14}$$

Where A_z is the area covered by the hypothesised root filter placement and A_y is the area covered by the annotated bounding box. The intuition behind this was that this enforcing of good bounding box localisation has a similar positive effect as the structured output formulation of [28], while being more flexible as the bounding box overlap is only penalised if the object is in full sight.

Chapter 4

Experiments and Results

4.1 Influence of Model Parameters

This section presents experimental results concerning the influence of several model parameters. The model presented in this work has a variety of different parameters that can have an influence on its performance. These parameters concern:

Training data:

- The amount of training data
- Synthetic vs. real vs. mixed data
- Orthographic vs. perspective CAD rendering
- Backgrounds of rendered CAD examples (negative examples vs. mean pixels)
- Angle α for which an example's viewpoint is fixed during training

Model:

- Number of views/mixture components
- Number of 3D parts
- Number of visible parts per viewpoint
- Rootfilter dimension/area
- Number of elevations the model is trained on

Learning:

- L-SVM parameters C and J (cost weighting factor for positive examples)
- Influence of incorrect viewpoint penalty

Features:

- HOG vs. CNN
- Truncation feature
- Standardisation of CNN features
- Extra octave for feature pyramid

To test the influence of some of these parameters and maximise performance of the model, experiments were conducted on the VOC 2012 validation data. For most of the experiments, all but one parameter were kept fixed and a model was trained with different values for the given parameter. After evaluating the performance of the model on the validation set, the best performing parameter value was chosen and fixed during experiments with other parameters. Both average precision as measured by the VOC criteria (a minimum of 0.5 bounding box overlap) and viewpoint accuracy (0.5 overlap and correct viewpoint) proposed by [38] were used as performance measures. Tables 4.1-4.3 show the results for some parameters as relative performance loss or gain for different values.

Table 4.1a illustrates the influence of the constraints on the model's root filter areas in feature dimensions. We can observe that the performance decreases for areas of sizes between 57 and 67 or 67 and 77 compared to root filter areas of sizes between 62 and 72. Smaller root areas lead to filters that do not capture enough detail and therefore might miss some meaningful details of the object's appearance. As root filter areas have a direct influence on the size of the smallest perceivable object instances, bigger root filters lead to fewer detections of small objects.

The influence of the number of mixture-components/views can be observed in table 4.1b. Note that viewpoint accuracy (VA) decreases significantly when increasing number of views from 8 to 16. This is expected as more views lead to a finer binning of viewpoints and neighbouring viewpoints therefore get more difficult to discriminate. We can also observe that

Root Area	AP	VA
57-67	0	0
62-72	+1.1	+1.4
67-77	+0.4	+0.2

(a) Relative AP and VA for different root area constraints (min-max)

#Components	AP	VA
4	0	0
8	+1.2	+0.6
16	-0.7	-6.2

(b) Relative AP and VA for different numbers of mixture components

#3D parts	Max parts per view	AP	VA
12	8	0	0
16	10	+0.4	-0.3
20	12	+1.2	+0.8

(c) Relative AP and VA for different numbers of 3D-parts and parts per view (PpV)

#elevations	AP	VA
1	0	0
2	-0.9	-0.6

(d) Relative AP and VA for different numbers of elevations

Extra pyramid octave	AP	VA
without	0	0
with	+5.2	+3.1

(e) Relative AP and VA for models with or without an extra pyramid octave

Table 4.1: Tables showing the influence of several model parameters on average precision (AP) and average viewpoint accuracy (VA).

detection performance increases when going from 4 views to 8. This can be explained by a more complex model and an increased variety of aspect ratios covered by the root filters. The slight decrease in performance when going from 8 views to 16 could be explained by the fact that more model components result in a reduction of training examples for each component.

The influence of different amounts of parts can be seen in table 4.1c. We see that increasing the number of parts generally increases the performance of the model. Note that by construction (section 3.2.3) more 3D parts lead to smaller 2D part-templates. Also, the model's complexity doesn't increase by much as part filters are only added until no more root filter area can be covered by new parts. Therefore the model effectively just grows from the additional deformation parameters.

Surprisingly, training models on more than one elevation didn't lead to better performance (table 4.1d). A possible explanation for this observation could be found in the distribution of object viewpoints in the data sets (see figure 3.6). As there are not many examples having an increased elevation there are not a lot of training examples for the new components and the training data of the more important (ground-level) views is reduced.

One of the most significant boosts in performance was observed when training the model with an extra octave added to the feature pyramid (table 4.1e). The highest resolution feature maps computed on an extra octave pyramid are computed at four times the resolution of the original image. This modification leads to more training examples, as some examples were just too small to be used with the normal pyramid, and also allows the model to detect smaller object instances at test time. It should be noted however that this performance gain come at the cost of increased detection and training time.

In table 4.2a we can observe the influence of different training data configurations. Note that training a model on only real training data implies that no 3D-constraints can be enforced on the model and parts are learnt fully independent across views. This can be one explanation why there is an improvement in performance compared to training on a combination of synthetic and real images, as parts can be learnt optimally for each separate view. Also, the importance of using perspective projections for rendering becomes apparent. While orthographic projections are a reasonable approximation for far away objects, real images of cars often are taken from a rather close distance. As table 4.2c shows, uniform backgrounds showed to work better than backgrounds of negative training images.

Training data	AP	VA
Real only	0	0
Real+CAD persp.	-1	-0.6
Real+CAD ortho.	-1.9	-2

(a) Relative AP and VA for different training data

Fixation angle α	AP	VA
20	0	0
45	-1	+0.2
5	-0.5	-0.4

(b) Relative AP and VA for different viewpoint fixation angles.

CAD background	AP	VA
Negative training images	0	0
Mean pixels of CNN features	+0.8	+2.3

(c) Relative AP and VA for different backgrounds of the CAD training examples

Table 4.2: Tables showing the influence of training data parameters.

Penalty term l	AP	VA
without	0	0
with	+0.5	+2.8

(a) Relative AP and VA influence of penalty term.

Truncation feature	AP	VA
without	0	0
with	+7.9	+7.5

(b) Relative AP and VA influence of penalty term.

Table 4.3: Tables showing the influence of penalty term and truncation feature.

The influence of the angle α which defines the range of viewpoints that are going to be assigned to a fixed mixture component during training is shown in table 4.2b. The choices for α here are listed for a four-component-model and of course $\alpha = min(\alpha, 360/(2 * \#Comp))$ holds. As expected a value of $\alpha = 45$, which means in this case that all examples are fixed to their correct viewpoint, increases viewpoint accuracy but decreases detection performance significantly. More surprising is that an angle of 5 decreases performance in both tasks compared to an angle of 20.

As we can see from table 4.3a the penalty term l, introduced to softly enforce the correct viewpoints (equation 3.12), does not only deliver a better viewpoint-estimation performance but also slightly raises detection performance. Again, we observe that correct viewpoint assignments favour the detection performance. Table 4.3b shows the significant performance boost obtained by the truncation feature and the related adjustments described in

3D-DPM	3D-DPM+	DPMver3 [10]	DPMver4 [7]	DPM-VOC [28]	DPM 3D-Const. [28]	3D2PM [29]
53.9	63.4	50.2	57.9	66	63.1	61.2

Table 4.4: Average Precision (AP) on cars of the VOC 2007 [5] test set. Results for the model in this project are given in the first two columns and results of other models are provided for comparison.

section 3.4.1.

4.2 Bounding Box Localisation

To test the model's detection performance the model was evaluated on the well known and challenging Pascal VOC 2007 [5] dataset, the VOC 2012 [6] 'val' set, and the 3D object dataset [32]. The performance measure used is the standard VOC measure of a minimum ground truth bounding box overlap of 50%.

Table 4.4 reports the results on VOC 2007 of the model implemented in this project (3D-DPM and 3D-DPM+) and also lists results for different versions of the DPM and similar models. 3D-DPM is an eight-view-model which was trained on perspective CAD examples and real training images from VOC 2007 'trainval'. 3D-DPM+ was trained on additional training data from VOC2012 'train', the first five car models of 3D object dataset and uses an extra octave pyramid.

We see that the 3D-DPM outperforms DPMver3 on which it is built upon and doesn't quite perform as good as DPMver4. The performance improvements with respect to DPMver3 can largely be attributed to the addition of the truncation feature, as well as the higher number of components. A possible explanation for the performance difference to DPMver4 could lie in the fact that VOC 2007 examples come without viewpoint annotations and therefore the model is initialised on CAD examples only (which might lead to sub-optimal filter dimensions and initialisations). Using additional examples from VOC 2012 in fact showed to increase the performance to around the same value as DPMver4 achieves. Another explanation can be found in the 3D constraints, which have already showed to have a negative impact on detection performance.

Compared to the other 3D DPM implementations DPM 3D-Constraints [28] and 3D2PM [29], 3D-DPM performs worse. On the other hand, the per-

#views	3D-DPM	3D-DPM+	VDPM [38]	DPM-VOC+VP [28]
4	38.2/30.2	47.4/37.3	37.2/20.2	45.6/36.9
8	39.4/30.8	49.4/35.5	37.3/23.5	47.6/36.6
16	37.5/24.0	47.5/29.5	36.6/18.1	46.0/29.6

Table 4.5: Average precision and average viewpoint accuracy (AP/VA) on cars of the VOC 2012 'val' set with annotations by [38].

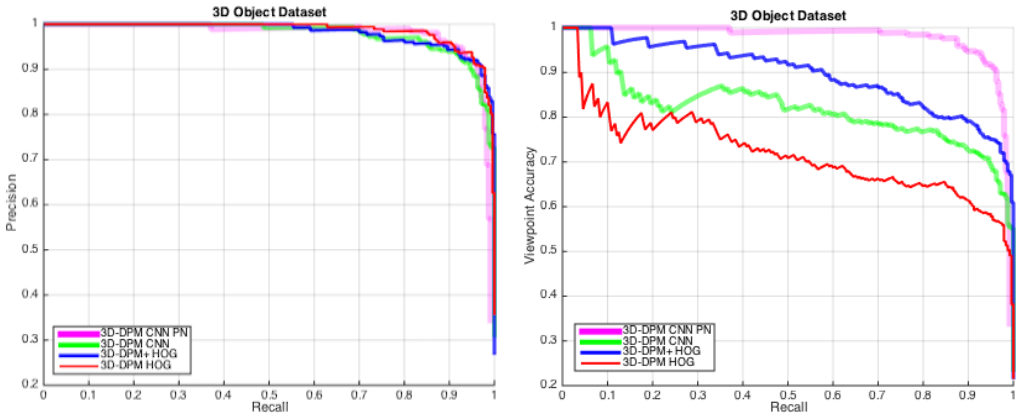
3D-DPM	3D-DPM+	DPM 3D-Const. [28]	3D2PM [29]	DPM-VOC+VP [28]
99.1/96.7	98.2/93.3	99.7/96.3	99.6/95.8	99.9/97.9

Table 4.6: Average precision and viewpoint estimation in MPPE on cars of the 3D Object classes dataset [32].

formance boost of 3D-DPM+ obtained by the use of an extra pyramid octave lifts its performance above the ones reported for the other two 3D models [29] [28]. This suggests that these models most probably also made use of an extra octave pyramid, an implementation detail not mentioned in their publications. As [28] report an average precision of 63.3 on VOC 2007 cars without the SSVM formulation and without the addition of synthetic training data, no other design choice seems to justify the performance difference to 3D-DPM or DPMver4.

Table 4.5 lists the AP of 3D-DPM and 3D-DPM+ on the VOC2012 'val' dataset. It also shows results for DPM-VOC+VP [29] and VDPM which is a modification of DPMver4 in which root filters are initialised on training sets that are obtained by binning the annotated VOC2012 test data of [38] into k viewpoint bins. After the initialisation on the correct viewpoints, training is identical to the standard DPM training of DPMver4. Here, we observe that 3D-DPM achieves consistently higher average precision compared to VDPM (DPMver4) and considerably worse than DPM-VOC+VP. On the other hand, 3D-DPM+ achieves the highest average precision of all the listed models. Some of the performance gain compared to DPM-VOC+VP might be due to the higher amount of training data.

Finally, the results on the 3D object classes dataset [32] are shown in table 4.6. This is a rather easy dataset as all the cars are in full sight and the viewpoints exactly match the viewpoints an eight-view-model is trained on. For comparison the results for other models are listed as well. Note however, that for these models it is unclear which of the examples in the dataset were



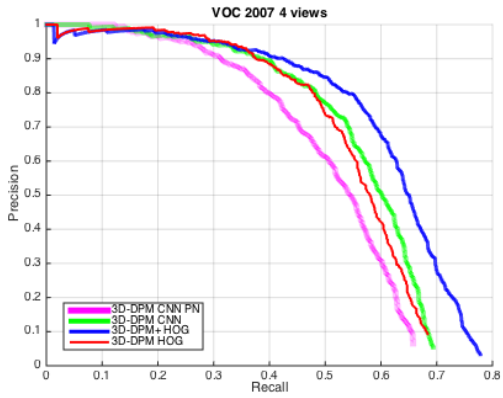
(a) 3D-DPM (b) 3D-DPM+

Figure 4.1: Precision/Recall (left) and Viewpoint-Accuracy/Recall (right) curves on cars of the 3D object dataset [32] of models described in this project. All the models were trained using all the training data (CAD, VOC2007, VOC2012, 3D Dataset). 3D-DPM+ uses an extra octave pyramid and 3D-DPM PN uses positive training data with wrong viewpoint labels as negative training examples.

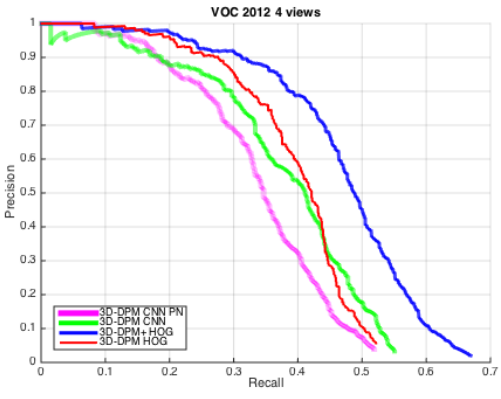
used for testing. While they state that a subset of the examples was used for training, the exact assembly and size of the subset is unknown, making the comparison difficult. 3D-DPM is trained on CAD renderings, examples from VOC2012 and the first five (of ten) car models of the 3D object dataset. The last five car models were used for testing. It is interesting to notice that 3D-DPM+ performs slightly worse than 3D-DPM on this occasion. As the dataset is also relatively small (only 480 car examples in total), these performance differences should be taken with a grain of salt. Overall, the performance is comparable to the other models.

4.3 Viewpoint Estimation

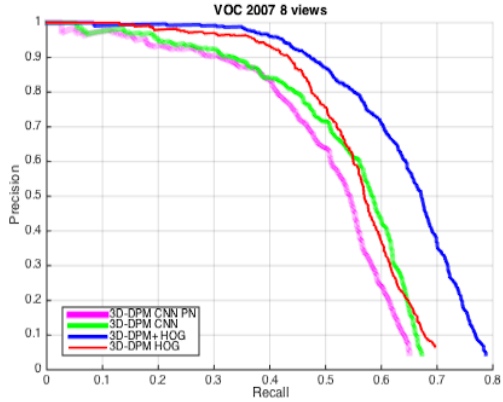
To test the model's performance on pose-estimation, experiments on the 3D object dataset as well as the VOC 2012 'val' set with the viewpoint annotations from [38] were conducted. The performance measures used were viewpoint accuracy (VA) and mean precision in pose-estimation (MPPE).



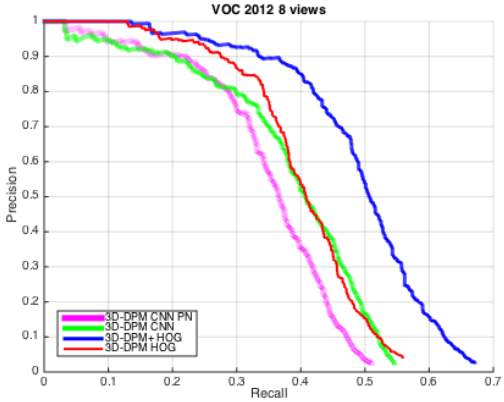
(a) VOC 2007 4 views



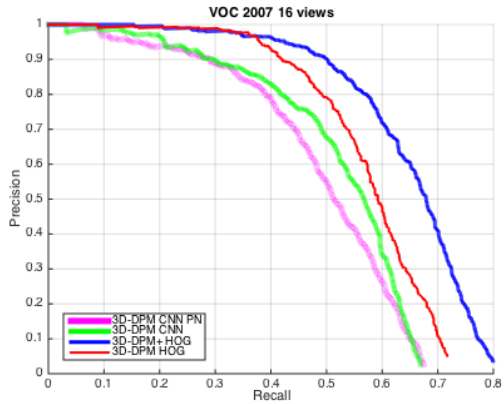
(b) VOC2012 4 views



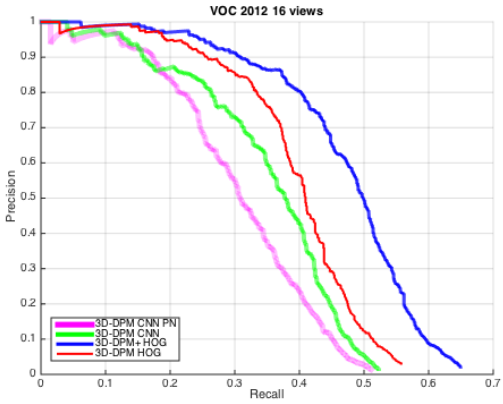
(c) VOC 2007 8 views



(d) VOC2012 8 views

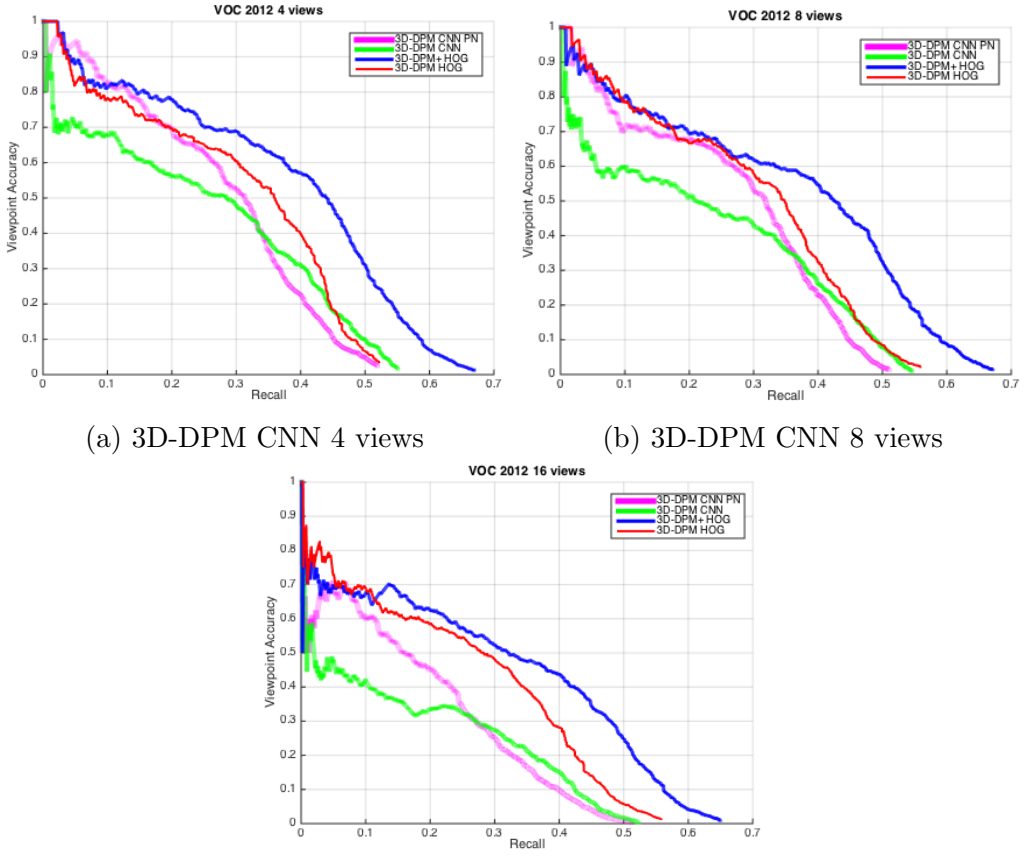


(e) VOC 2007 16 views



(f) VOC2012 16 views

Figure 4.2: Precision-Recall curves on VOC 2007 'test' (left) and on VOC 2012 'val' (right). All the models were trained using all the training data.



(a) 3D-DPM CNN 4 views

(b) 3D-DPM CNN 8 views

(c) 3D-DPM CNN 16 views

Figure 4.3: Viewpoint-Accuracy-Recall curves on VOC 2012 'val' of models described in this project. Only examples with bounding-box overlap of at least 0.5 and correct viewpoint label are considered correctly classified. All the models were trained using all the training data (CAD, VOC2007, VOC2012, 3D Dataset). 3D-DPM+ uses an extra octave pyramid and 3D-DPM PN uses positive training data with wrong viewpoint labels as negative training examples.

Viewpoint accuracy only counts detections with a minimum bounding box overlap of 50% and a correct viewpoint assignment as valid detections, therefore measuring the combination of both, object localisation and viewpoint estimation. MPPE is defined as the average along the diagonal of the viewpoint confusion matrix, therefore ignoring correct bounding box localisation altogether.

Table 4.5 lists results on the 3D object category dataset in MPPE. 3D-DPM was trained using CAD renderings, examples from VOC 2012 and the first five cars of the 3D object category dataset. 3D-DPM+ again used all the training examples and makes use of an extra octave pyramid. Results for other 3D Deformable Parts Models [28] [29] and DPM-VOC+VP [28] are listed for comparison. Note however that the test sets might differ (see section 4.2). The MPPE achieved by 3D-DPM is slightly higher than the results reported by the other 3D models and not quite as good as the results obtained by DPM-VOC+VP. Interestingly however, 3D-DPM+ performs worse than the other models on this dataset.

The viewpoint accuracy results on VOC 2012 are listed in table 4.5. For comparison the results reported for DPM-VOC+VP and VDPM (DPMver4 initialised on correct viewpoints) are shown as well. By comparing the results of 3D-DPM to VDPM (which both use the same pyramid), we can observe the drastic improvement in viewpoint accuracy obtained by the continuous enforcement of correct viewpoints during training through component-fixation and the penalty term. The results for 3D-DPM+ show a significant improvement in viewpoint accuracy compared to 3D-DPM. This improvement is mainly due to the better detection performance and does not imply that 3D-DPM+ obtains better viewpoint estimation given a correct detection. The results achieved by 3D-DPM+ are nearly identical to the results obtained by DPM-VOC+VP [28].

4.4 CNN vs. HOG Features

This section will present the influence of several design choices on the performance of a CNN based model, report results obtained with such models and compare the performance of a CNN based model to HOG based models.

Table 4.7 lists the relative performance gain on the VOC2012 'val' set achieved through feature standardisation and table 4.8 shows the influence of the increased deformation lower-bounds. The importance of these modifi-

Feature standardisation	AP	VA
without	0	0
with	+22	+21.5

Table 4.7: Relative AP and VA with or without feature standardisation

Lower bounds on deformations	AP	VA
$(0.01, 0, 0.01, 0)$	0	0
$(0.2, 0, 0.2, 0)$	+7.6	+8.4

Table 4.8: Relative AP and VA for different lower-bounds

cations is obvious from the experiments. As mentioned in section 3.3, feature standardisation mainly improves the SVM training while the increased lower-bounds are important to prevent parts from moving too far away from their anchor positions.

Results for a CNN-based model on VOC 2007 'test' are listed in table 4.9. For comparison, the results of a HOG-based 3D-DPM are also shown. Both these models were trained using all the training data. 3D-DPM CNN is trained using badly localised positives as negatives in addition to the normal negative training examples. Only positive examples with annotated bounding box overlap of less than 30% count as badly localised and care is taken to only consider images where not more than one car is present to extract negative examples from. We can observe that the CNN model only performs better with a four-viewpoints-model. Note how the performance of 3D-DPM

#views	3D-DPM CNN	3D-DPM HOG
4	55.3	54.3
8	53.1	54.7
16	51.8	57.1

Table 4.9: Comparing the detection performance of a CNN based model to a HOG based model using average precision on cars of the VOC 2007 'test' set.

#views	3D-DPM CNN	3D-DPM CNN-PN	3D-DPM HOG
4	38.0/24.4	34.1/28.0	38.2/30.2
8	38.0/23.8	35.1/26.8	39.4/30.8
16	35.2/14.6	30.9/17.8	37.5/24.0

Table 4.10: Comparing the performance of CNN based models to a HOG based model using average precision and average viewpoint accuracy (AP/VA) on cars of the VOC 2012 'val' set with annotations by [38].

CNN decreases with an increased number of model viewpoints, while the performance of the HOG model improves. This observation was also made on other datasets and probably has to do with the reduction of training data for each individual model viewpoint.

In table 4.10 the performance of two CNN based models on VOC2012 'val' are shown in comparison to the HOG based 3D-DPM. The HOG based model doesn't make use of an extra octave, therefore making the feature pyramids of all the models comparable relative to their root size. We can observe that the detection performance of 3D-DPM CNN is only slightly below the performance achieved by 3D-DPM HOG. This is in accordance with recent work combining CNN features with DPMs [16] where the car category has shown to be one of only two categories where AP did not increase with CNN features while significantly improving the mean average precision (mAP) over all object categories (from 33.7 for HOG to 44.4 for CNN on VOC2007). While average precision is about on par with HOG based models, we can see that the viewpoint accuracy (VA) of 3D-DPM CNN is several points below the VA achieved by 3D-DPM HOG. This is due to a tendency observed in CNN based models to assign examples that are not viewpoint annotated to only a few model components. This results in detections that are concentrated on only those few components. As VA is arguably a better measure for the performance of a 3D-DPM as it directly measures the two main objectives of a 3D-DPM (object localisation and pose-estimation) we would like to increase the models VA performance. 3D-DPM CNN-PN uses positive examples with wrong viewpoint assignments as additional negative training examples. The intuition is that the model in this case learns to better distinguish between the different viewpoints of the object. As can be seen in table 4.10 the VA clearly improves with this training procedure

3D-DPM CNN	3D-DPM CNN-PN	3D-DPM HOG
97.9/82.8	97.6/97.5	99.14/93.0

Table 4.11: Comparing the performance of CNN based models to a HOG based model using average precision and average viewpoint accuracy (AP/VA) on cars of the 3D object dataset [32].

while AP significantly decreases, and VA still not matching the performance of 3D-DPM HOG.

It is interesting to note however the results achieved on the 3D object dataset [32] (listed in table 4.11). This dataset contains only cars in full sight, i.e. no truncated or occluded cars. Here 3D-DPM CNN-PN achieves the highest VA of all the models, also clearly outperforming the HOG based model. It is also noteworthy that AP and VA are nearly identical for 3D-DPM CNN-PN, therefore achieving the highest VA/AP ratio of all the models (see figure 4.1).

Overall, the observation is that off-the-shelf CNN features do not lead to better results for 3D Deformable Parts Models on cars. It should be noted that the features used here were trained for image classification and that fine-tuning the parameters for object detection usually results in rather drastic performance improvements.

Chapter 5

Conclusion

During this project a 3D Deformable Parts Model was implemented and carefully evaluated with respect to different design choices and model parameters. The result of this work is a 3D model whose performance on object detection and pose-estimation is on par and in many cases even better than results reported for other 3D-DPMs [29] [28]. Through the evaluation of several design choices and model parameters, deeper insights into the training of 3D-DPMs and DPMs in general have been gained. Among the factors that turned out to be important, were the construction of the feature pyramid where an extra octave (to detect small objects) showed significant performance improvements. The way how truncated and occluded objects are handled turned out to be crucial as well, and it was found that performance improves significantly by the use of a truncation feature and effective handling of bounding-box overlap requirements during training. It has also been shown how different types of CAD examples affect the performance of the model. Here, perspective renderings on uniform backgrounds have proofed to work best. The introduction of a penalty term during latent positive search has turned out to be an effective way to increase the viewpoint accuracy of the 3D-DPM.

While the model has shown to perform well on viewpoint estimation, the estimations at the moment are still limited to a set of discrete viewpoints the model was trained on. While it is in theory possible to train the model on arbitrarily many discrete viewpoints (provided that sufficient CAD renderings are generated), this is clearly not an efficient solution. An idea for a new and alternative approach is outlined in section 5.1.

The incorporation of features obtained by Convolutional Neural Networks

has been found to be more difficult than expected. Feature standardisation, an adjustment of the deformation lower-bounds and a modification of the training procedure have proofed to be essential to get a comparable detection performance to HOG features. Also some compromise between performance and training time had to be made in order to make experiments feasible (reducing the number of pyramid levels mainly). While the detection performance of HOG features can nearly be matched by CNN based models using these modifications, the viewpoint estimation capabilities of CNN based models showed to be significantly worse. By using wrong viewpoint assignments as negative examples during training, I managed to increase the pose-estimation performance with CNN features such that they even exceed the performance of HOG based models on datasets where objects are not occluded or truncated.

5.1 Future work

One idea of future research work on the model aims to increase the model's viewpoint accuracy, and to also allow for finer viewpoint estimation. Rather than using interpolation between filters at test-time to allow finer viewpoint estimation as in [29] (which increases detection time), the idea is to use a voting scheme to determine the viewpoint. Given an image of a car, there are typically going to be several overlapping object hypotheses with different viewpoint assignments and scores around the correct object localisation. Rather than just applying non-maximal suppression on all these hypotheses, the non-maximal suppression could only be applied to groups of possible detections with the same viewpoint-assignment only and the resulting set of hypotheses used to arrive at a better, more complete object hypothesis. The most confident (highest scoring) hypothesis could be used as an initial guess for the viewpoint and location of the object and hypotheses with neighbouring viewpoints as well as a certain overlap with the initial guess could then be used to refine the initial guess by voting for other viewpoints (with voting-weights being proportional to the hypothesis score). This approach would also not lead to a significant increase in detection time as there are not more convolutions of the filters with the feature maps needed.

Another path for future work could be the improvement of the CNN based models by fine-tuning the network weights to the given task of object detection with Deformable Parts Models. As such fine-tuning has shown

to be very effective by [15], this could result in much better performance. To adapt the fine-tuning procedure to the intended use with DPMs, region proposals with a bounding box overlap of more than 0.5 as well as smaller regions totally inside the bounding box (related to possible parts) could be used as positive examples.

It would also be interesting to test the models performance on other object classes, especially with CNN features as they have shown to improve over HOG features for other object classes than cars. This of course means that training data in the form of CAD examples would first have to be created.

Bibliography

[1] Gerald J Agin. Representation and description of curved objects. Technical report, DTIC Document, 1972.

[2] Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning.

[3] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014.

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

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[7] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Discriminatively trained deformable part models, release 4. http://people.cs.uchicago.edu/ pff/latent-release4/.

[8] Pedro Felzenszwalb and Daniel Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell University, 2004.

[9] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[10] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, Sept 2010.

[11] Martin A Fischler and Robert A Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, 22(1):67–92, 1973.

[12] Patrick J Flynn and Anil K Jain. Cad-based computer vision: from cad models to relational graphs. In *Systems, Man and Cybernetics, 1989. Conference Proceedings., IEEE International Conference on*, pages 162–167. IEEE, 1989.

[13] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[14] R. B. Girshick, P. F. Felzenszwalb, and D. McAllester. Discriminatively trained deformable part models, release 5. http://people.cs.uchicago.edu/ rbg/latent-release5/.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv preprint arXiv:1311.2524*, 2013.

[16] Ross Girshick, Forrest Iandola, Trevor Darrell, and Jitendra Malik. Deformable part models are convolutional neural networks. *arXiv preprint arXiv:1409.5403*, 2014.

[17] Ross Girshick and Jitendra Malik. Training deformable part models with decorrelated features. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3016–3023. IEEE, 2013.

[18] D. Hoiem, C. Rother, and J. Winn. 3d layoutcrf for multi-view object class recognition and segmentation. In *Computer Vision and Pattern*

Recognition, 2007. CVPR '07. IEEE Conference on, pages 1–8, June 2007.

[19] Forrest N. Iandola, Matthew W. Moskewicz, Sergey Karayev, Ross B. Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *CoRR*, abs/1404.1869, 2014.

[20] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[23] Joerg Liebelt, Cordelia Schmid, and Klaus Schertler. Viewpoint-independent object class detection using 3d feature maps. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[24] David G Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial intelligence*, 31(3):355–395, 1987.

[25] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[26] Hiroshi Murase and Shree K Nayar. Visual learning and recognition of 3-d objects from appearance. *International journal of computer vision*, 14(1):5–24, 1995.

[27] M. Ozuysal, V. Lepetit, and P. Fua. Pose estimation for category specific multiview object localization. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 778–785, June 2009.

[28] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3d geometry to deformable part models. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3362–3369, June 2012.

[29] Bojan Pepik, Peter Gehler, Michael Stark, and Bernt Schiele. *3D2PM – 3D Deformable Part Models*, volume 7577, pages 356–370. Springer Berlin Heidelberg, 2012.

[30] Lawrence G. Roberts. *Machine Perception of Three-Dimensional Solids*. Outstanding Dissertations in the Computer Sciences. Garland Publishing, New York, 1963.

[31] Pierre-André Savalle, Stavros Tsogkas, George Papandreou, and Iasonas Kokkinos. Deformable part models with cnn features. In *3rd Parts and Attributes Workshop, ECCV*, volume 8.

[32] S. Savarese and Li Fei-Fei. 3d generic object categorization, localization and pose estimation. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, Oct 2007.

[33] Silvio Savarese and Li Fei-Fei. View synthesis for recognizing unseen poses of object classes. In *Computer Vision–ECCV 2008*, pages 602–615. Springer, 2008.

[34] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[35] Alexander Thomas, Vittorio Ferrar, Bastian Leibe, Tinne Tuytelaars, Bernt Schiel, and Luc Van Gool. Towards multi-view object class detection. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1589–1596. IEEE, 2006.

[36] Stephen A Underwood and Clarence L Coates. Visual learning from multiple views. *Computers, IEEE Transactions on*, 100(6):651–661, 1975.

[37] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. *CoRR*, abs/1412.4564, 2014.

[38] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2014.

[39] Pingkun Yan, Saad M Khan, and Mubarak Shah. 3d model based object class detection in an arbitrary view. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–6. IEEE, 2007.

Appendix A

Image and Feature Pyramid

Sliding window approaches make use of feature pyramids to detect objects at different scales and locations in an image. To compute a feature pyramid, first an image pyramid is constructed. Image pyramids are constructed via repeated subsampling of the original image. The subsampling factor between any two consecutive pyramid levels is constant and determined via a parameter λ. This parameter λ defines the number of levels in an octave. An octave in turn is defined as the number of levels that have to be stepped down in a pyramid until an image of twice the resolution is reached. Figure 3.1a visualises an image pyramid with two octaves and $\lambda = 1$. The bottom level of an image pyramid is usually at twice the resolution of the original image. In case an extra-octave pyramid or CNN features are used, the original image is resized to twice the resolution before the construction of the image pyramid, which results in a bottom level that has four times the resolution of the original image. Note that only the parts will be placed in the higher resolution bottom octave of the pyramid. After the image pyramid is constructed, feature maps are computed for every pyramid level which results in the desired feature pyramid (see figure 3.1b).