OBJECT DETECTION WITH CAD-TRAINED STATISTICAL MODELS

FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
GUSTAV LARSSON

ADVISOR: YALI AMIT

CHICAGO, ILLINOIS
MARCH 2014
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ABSTRACT</strong></td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Objective</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1</td>
<td>Object class instantiation</td>
<td>2</td>
</tr>
<tr>
<td>1.1.2</td>
<td>Object detection</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Model testing</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Training from synthetic data</td>
<td>5</td>
</tr>
<tr>
<td>1.4</td>
<td>Outline</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>IMAGE REPRESENTATION</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>Feature extraction</td>
<td>8</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Overview</td>
<td>8</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Edge extraction</td>
<td>9</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Edge spreading</td>
<td>11</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Parts model</td>
<td>12</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Part spreading</td>
<td>14</td>
</tr>
<tr>
<td>2.1.6</td>
<td>Subsampling</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>Training parts</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Visualizing parts</td>
<td>16</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Post processing</td>
<td>16</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Optimization</td>
<td>20</td>
</tr>
<tr>
<td>2.3</td>
<td>Latent polarity/orientation</td>
<td>20</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Polarity</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Orientation</td>
<td>21</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Training</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>OBJECT DETECTION</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Training from real images</td>
<td>25</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Training</td>
<td>25</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Handling degenerate probabilities</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Training form CAD models</td>
<td>28</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Introduction</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Training</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Testing</td>
<td>30</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Classification</td>
<td>30</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Standardization</td>
<td>31</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Key points</td>
<td>33</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Minimizing correlation between features</td>
<td>34</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Detection</td>
<td>34</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Bounding box</td>
<td>35</td>
</tr>
<tr>
<td>3.3.7</td>
<td>Non-maximal suppression</td>
<td>36</td>
</tr>
<tr>
<td>3.3.8</td>
<td>Detection of small objects</td>
<td>36</td>
</tr>
</tbody>
</table>
ABSTRACT

Using 3D CAD models to train computer vision systems is an attractive premise that reconnects us with earlier work within the field. The prospects of an ample source of noiseless and automatically pose-annotated training data sounds too good to be true. Perhaps it is, since so far attempts made by the community have not been close to matching those of natural data, making the supposed benefits moot.

In this work, we argue that the performance discrepancy is not necessarily due to the difference in image statistics. Instead, one of the main problems is that objects in natural images follow a heterogeneous camera pose distribution, favoring certain angles. Training from data where this distribution naturally occurs, infuses the model with a strong prior that will be beneficial at testing time. However, current attempts of using CAD models render the objects from a uniform camera pose distribution, within a certain set of constraints. By selecting a camera pose distribution with only a few hand-picked nonzero points, we show that detection rates are greatly improved over the uniform alternative, even with much less training data.

The paper also explores using simple generative models for building the detector. Algorithmically, our models are trivial compared to our baseline of choice, the linear SVM. However, that does not mean the problems we face are trivial, such as how to deal with correlated features when the model assumes independence. In a sense, these problems are solved automatically by the SVM. We believe that exposing them and trying to solve them explicitly can be a great source of knowledge and eventually lead to better models. By carefully addressing some of these problems, we show that we can achieve results not far behind the baseline.

In the same vein, we do not use any natural images as part of the training process, even though methods that mix synthetic and natural training data have been fairly successful. Training solely from CAD data forces us to face the problems associated with this approach and exposes the real reasons behind the discrepancy in error rates.
CHAPTER 1
INTRODUCTION

1.1 Objective

In broad strokes, the various goals of computer vision can all be phrased as learning a function $x \mapsto y$, where $x$ is visual data and $y$ is a high-level description of $x$. Depending on how this task is concretized, various problems within the field emerge, each with its own definition of “high-level description” and usually with further constraints on $x$. It is this high-level reasoning that sets the field aside from image processing, as well as making it one of the central fields in the broader area of artificial intelligence.

Examples of tasks in computer vision include (1) optical character recognition (OCR): detecting written characters in images. This could help read a hand-written zip code in an image or parse the text of old books that have been digitally scanned. (2) structure from motion: given a video, trying to understand and reconstruct the 3D geometry of the scene (3) segmentation: Given an image of a specific object, determine which pixels belong to the object and which belong to the background. (4) classification: Given an image of an object, determine which class the object in the image belongs to, given a finite set of object classes from which to choose (5) facial detection: Locating human faces in images.

The last task mentioned is already in active use in consumer products. Digital cameras will identify faces to better adapt focus and exposure time to optimize quality locally over those regions. When users upload photos to social networking services, faces are sometimes automatically highlighted so that they can be linked to the corresponding user’s account.

This body of work focuses on the more general task of object detection, where the object class is not necessarily the class human face. Particular focus will be put on broad object classes, such as car and pedestrian.

Before further specifying this task, let us explore the space of object instantiations in images.
1.1.1 Object class instantiation

The object classes that we are interested in, such as car, or more specifically car in image, are extremely diverse for several reasons. First, the class includes many different types of cars, everything from compact cars to station wagons. Depending on the data set, it may even include vans and pickup trucks. Each category has various brands, models, colors and other variations.

Furthermore, an object as portrayed in an image can have any given object pose, by which we primarily mean its translation and orientation in 3D space. When detecting a single object, we prefer to equivalently consider that the object has a canonical pose, whereby it is the camera that has a translation and orientation. These are called the extrinsic camera parameters and the task of object pose estimation is thus also referred to as extrinsic calibration. For some classes, especially the pedestrian class, object pose can also include an articulated body pose, in which case the object is considered a collection of joints and rigid parts. This could also be applied to cars, giving degrees of freedom to turn the wheels and open doors.

Then there are the intrinsic camera parameters, such as focal length, image format and lens distortions. Together the extrinsic and intrinsic camera parameters define a projection transform that we will commonly refer to as the camera pose. If we add lighting, an image representation of the object is established. When doing object detection, the background and foreground of the object will also play important roles, especially if they occlude the object of interest.

This is a coarse overview of the instantiation of an object class in a 2D image. There are many aspects omitted here, but this covers the greatest sources of variation to which a competitive object detector must be invariant.

In the following sections, it will be useful to imagine that an instantiation is fully determined by a set of instantiation parameters, even though we will not explicitly enumerate them.
Figure 1.1: Example of how the ground truth for detections are defined as tight bounding boxes around the instances, regardless of occlusion patterns. (Source: PASCAL VOC 2007 [10])

1.1.2 Object detection

In object detection, after we find all instances of an object class in an image \( (x) \) we need to formalize the information about them \( (y) \). This information will be a subset of the instantiation parameters mentioned above, or derivatives thereof. For instance, an object detector could report only the pixel location of an object instance’s center of gravity. This would correspond to the image projection of the object’s translation parameter, if the object is centered as such. The detector could also report a much more detailed account of the object, such as its 3D pose, appearance characteristics, etc.

A common format, that the community has settled with, is to report the tight bounding box of each object instance. This is defined as the smallest axis-aligned rectangle that covers all pixels belonging to the object instance (see fig. 1.1).

Suppose there exists an oracle \( f : \mathcal{X} \to \mathcal{Y} \), such that for all images \( x \in \mathcal{X} \), its return value \( f(x) \in \mathcal{Y} \) is a correct list of bounding boxes of all instances from a specific object class. We make the assumption that this task always has a correct and unambiguous answer. This boils down to a machine learning problem, both in terms of the training and testing procedure.
The central goal of this research topic is to devise an estimate of \( f \), denoted \( \hat{f} \). To determine how well \( \hat{f} \) captures \( f \), an evaluation procedure is needed.

### 1.2 Model testing

The evaluation method of an estimate \( \hat{f} \) that is described here is taken from the PASCAL Visual Object Classes (VOC) Challenge [10]. This is not the only data set that we will report results on, but it uses the most common evaluation method.

Given a testing set

\[
D_{\text{test}} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\},
\]

for each \((x, y) \in D_{\text{test}}\), we can compare \( \hat{y} = \hat{f}(x) \) to \( y \), to see how well they match. Since \( \hat{f} \) will in most cases not produce a pixel-perfect bounding box around the object, a bounding box \( A \in \hat{y} \) compared to the correct \( B \in y \) is considered a true positive (TP) if

\[
\text{overlap}(A, B) = \frac{|A \cap B|}{|A \cup B|} \geq \frac{1}{2},
\]

where \(|\cdot|\) takes the area of a bounding box. For each bounding box in \( y \), at most one can be matched as correct in \( \hat{y} \); the rest are deemed false positives (FP). If a bounding box in \( y \) has no match at all in \( \hat{y} \), it accounts for a false negative (FN).

From this, precision (\( \frac{TP}{TP+FP} \)) and recall (\( \frac{TP}{TP+FN} \)) can be calculated. In order to better capture the trade-off between precision and recall for a detector \( \hat{f} \), the detector augments each predicted bounding box with an arbitrarily scaled confidence value. The threshold that will decide which detections to keep can now be varied to form a precision/recall curve. This curve can then be summarized by the average precision (AP), which is the integral of the precision over the recall.
1.3 Training from synthetic data

Each public data set provides a set of training images that can be used in the learning process. Instead of using this training data, the focus of this paper is to use 3D computer-aided design (CAD) models to train the detector. See fig. 1.2 for a few examples of CAD models rendered as 2D images.

The benefits are many. First of all, CAD models offer a compact way of representing an object class. Given a CAD model, an object can be rendered using numerous poses, camera intrinsics and lighting conditions. For broad object classes, such as car, a single CAD model is not enough. Luckily, in recent years, CAD models of common object classes are easily acquired.

Not only does it offer us the ability to render as much data as we need, but it does so without noise and outliers. Too idiosyncratic data, such as an image of a car with an occluding mailbox in the foreground, is highly unlikely to strengthen the object model. This will make for a cleaner and stronger object model. However, synthetic data does not preclude us from rendering images with certain occlusion patterns or noise, if we for some reason want to include that explicitly in the learning process.
Knowing the 3D geometry of objects also allows a deeper scene understanding, which is crucial for tasks such as robot navigation. It can also greatly benefit viewpoint estimation, segmentation and ultra-wide baseline matching. We will not demonstrate these benefits here, but it is still part of the motivation for why we should care about training with CAD models. Also, bear in mind that the techniques that we develop can equally be applied to CAD models that were automatically created using structure from motion. Imagine a situation where a robot observes an object from a range of viewpoints, learns its geometry, and then applies that knowledge to various computer vision tasks.

Naturally, training from synthetic data comes with its own set of challenges that need to be addressed. Even though an occluding mailbox is too idiosyncratic to strengthen the model, there are other aspects that are far more common. Most often, detection algorithms train on a selection of rectangular photographs of the object, without specifying exactly which pixels belong to the object. If the object is consistently placed in a certain context, the model can learn the context as part of the object model. For instance, cars are more often than not on road, which could get absorbed into the object model. If the cars in the testing set are also frequently on road, this could actually help detection rates.

It is a common conception that that the principal challenge of using CAD models is that synthetic data does not look exactly like the data in the testing set. Looking at fig. 1.2, you can tell that they are 3D renderings and not photographs, not only because they are perfectly isolated, but because of subtle image statistics. Rendering techniques are getting better and better, and I guarantee that everyone has seen a computer-generated car in a movie without realizing it. However, such rendering perfection does not mean it will perfectly match the image statistic of your particular problem domain, and more concretely, your testing set.

One of the main conclusions of this work is that this aspect alone is over-emphasized and that there are other differences that are also of crucial importance. For instance, consider all cars in 2D images as samples from a distribution over the instantiation parameter space. This means that some camera poses will be more common than others. For instance, a car
shown in profile will be far more common than a car from underneath. Training from only CAD models, there is no data from which to estimate the distribution over these parameters. Similarly, there is also a distribution over car models that must be manually estimated by selecting the right set of CAD models.

Simply put, synthetic data provides a source of clean data, but it introduces a systematic error, as well as leaving us without any prior knowledge of instantiation parameters. This paper’s main objective is to explore this premise and learn how to best use the lack of noise, while coping with the systematic error.

1.4 Outline

Learning a robust detector is a challenging task that requires good image representation, object model and hypothesis testing. We will begin by covering the image representation in chapter 2 and then move on to the object model and detection in chapter 3. Related work on CAD-based training is surveyed in §3.5. Experimental results are presented in chapter 4, followed by conclusion and future work in chapter 5.
CHAPTER 2

IMAGE REPRESENTATION

2.1 Feature extraction

The raw data are assumed to be grayscale and sometimes with an opacity ($\alpha$) channel. Both intensity and opacity are values in $[0, 1]$. If the images instead are color, then we convert them to intensity by taking $\frac{R+G+B}{3}$, where $R$, $G$ and $B$ are the color channels. This incurs an information loss that is ideally avoided in a more pragmatic setting, but accepted here in favor of interpretability.

Let $I \in \mathcal{I}$ denote an image, where $\mathcal{I} = [0, 1]^{W \times H \times 2}$, in the case that it has an opacity channel. Let $\phi : \mathcal{I} \to \mathcal{X}$ represent a mapping to a binary feature space $\mathcal{X} = \{0, 1\}^{W' \times H' \times F}$. Let $F$ denote the dimensionality of the descriptor that can be found at $L = W' \times H'$ spatial locations. Given a sample $X \in \mathcal{X}$, let $X_{l,f}$ denote the feature $f$ at spatial location $l$.

2.1.1 Overview

The feature extraction process, $\phi$, discussed here has several steps. Note that $\phi$ ignores the $\alpha$ channel and images will be assumed be completely opaque before extracting features.

**Edge extraction** At each pixel and for a set of orientations, a binary edge is determined by intensity differences from a small local neighborhood. A value of 1 means an edge is present.

**Edge spreading** Edges are spread to nearby neighbors.

**Parts coding** At each location, a local neighborhood of edge features is matched against a codebook of parts. The feature descriptor at that location is taken to be an indicator vector of the maximum likelihood part.

**Parts spreading** Parts are spread to nearby neighbors.
**Subsampling** The size of the model is reduced by subsampling.

The edges and the parts are similar in nature: they both describe the structure of a local neighborhood. Feature spreading is done to promote local invariance and improve robustness. This is reminiscent of a 2-layer deep neural network that propagates at each level according to localized receptive fields (e.g. convolutional neural network [13, 18]). The main difference is that instead of supervised training using back-propagation, we use unsupervised training using EM.

In the spirit of this layered structure, we will denote the edge space as $\mathcal{X}^{(1)}$ and the parts space before subsampling as $\mathcal{X}^{(2)}$. After subsampling, the feature space will be denoted simply as $\mathcal{X}$. To avoid notational clutter, we may use $X$ to denote a feature vector at any level of the hierarchy.

### 2.1.2 Edge extraction

The binary edge features described here are taken from [1, p. 81], but restated in full with minor changes.

For an image space $I_{W \times H}$, there will be a corresponding edge space $\mathcal{X}^{(1)}_{W \times H} = \{0, 1\}^{W \times H \times E}$, where $E$ is the number of edge orientations at each pixel and a value of 1 indicates the presence of an edge. We will consider edges along the cardinal and diagonal directions, giving us $E = 8$ types of edge orientations.

For each pair $(y, z)$ of neighboring pixel locations, an edge between them is considered. Both immediate and diagonal neighbors are considered and the angle between the pixels determines the edge orientation. Let $\Delta I_{y,z} = |I(z) - I(y)|$ denote the intensity difference between the two pixels and $R_z$ a 90° rotation operation around $z$. Let $k$ be defined for each pair as follows (see fig. 2.1)

$$k(z, y) = \sum_{i=1}^{3} 1\{\Delta I_{y,z} > |I(z) - I(R_z y)|\} + \sum_{i=1}^{3} 1\{\Delta I_{y,z} > |I(y) - I(R_y z)|\}.$$
Figure 2.1: Intensity comparisons when calculating $k$. An edge is present if the gradient marked in red is greater than $k$ of the 6 gradients marked in black. The resulting edge is associated with the highlighted pixel at $z$, causing a half-pixel misalignment. The two figures are examples of two different edge directions out of four tested.

The value of $k(z, y)$ counts how many other local intensity differences that $\Delta I_{y,z}$ is greater than. An edge is deemed present at $z$ if $k(z, y) \geq k_{\text{min}}$, where $k_{\text{min}}$ is typically 5 or 6. This makes edges represent not just gradient activity but local gradient maxima.

For each $z$, this is performed for four neighboring $y$ covering $180^\circ$. The polarity of an edge, meaning the direction of its gradient, is then determined by $\text{sgn}(I(z) - I(y))$, giving us a total of 8 edges covering $360^\circ$. In certain problem domains, the polarity of the edge may not be useful and can undesirably bias the model. Skipping this step leaves us with polarity insensitive edges, in which case $E = 4$. Another possibility is to use the full $E = 8$ and achieve polarity insensitivity at a higher level (see §2.3).

To avoid unwanted edge activity in areas with extremely low contrast, the condition that $\Delta I > \kappa$ is added, where $\kappa$ is called the minimum gradient threshold.
2.1.3 Edge spreading

The act of spreading binary features to local neighbors is called \textit{dilation} in mathematical morphology. Let $A, B \subset \mathbb{Z} \times \mathbb{Z}$ be sets of pixel locations where edges are present. The dilation of $A$ by $B$ is then defined as

$$ A \oplus B = \bigcup_{b \in B} \{a + b : a \in A\}, $$

where $A$ is the source binary image and $B$ is a structuring element (kernel) that describes the neighborhood in which to spread. Since our binary features are defined as 0’s and 1’s and not sets of pixel locations, we will abuse notation and allow the left operand of “$\oplus$” to take a dense representation as well, in which case the dilated image should follow the same format.

If $X \in \mathcal{X}^{(1)}$ is a set of edge features before spreading, let $X_{:,e}$ denote the binary image describing only edge type $e$. An edge map $X^{(spr)}$ after spreading is then defined as

$$ X^{(spr)}_{:,e} = X_{:,e} \oplus K_e \quad \forall e = 1, \ldots, E. $$

The structuring element $K_e$ may depend on $e$, meaning that edge spreading may depend on the orientation of the edge. For all experiments in this work, edges are spread with a single-pixel line along the gradient, in other words orthogonal to the contour. This can be expressed as

$$ K_e = \{p_{e \mod 4}(x) : x \in \{-r_{edge}, \ldots, r_{edge}\} \subset \mathbb{Z}\} $$

where $r_{edge}$ is the edge spreading radius and

$$ p_1(x) = (x, 0), \quad p_2(x) = (x, x), \quad p_3(x) = (0, x), \quad p_4(x) = (-x, x). $$

The edge spreading radius $r_{edge}$ is left as a parameter. This process is easier to understand...
Figure 2.2: Example of edges extracted from a small image. Opposite facing edges are shown together with different colors. Edges activated through spreading ($r_{\text{edge}} = 2$) are shown in a lighter color. Edges in opposite directions are mutually exclusive without spreading, but can co-occur with spreading (shown in purple). Notice that the orthogonal edge spreading kernel means that horizontal lines in the image are spread vertically. (Source: PASCAL VOC 2007 [10])

by looking at the example in fig. 2.2.

2.1.4 Parts model

Following is a description of Bernstein and Amit’s part-based model [4]. This model describes a sparse, but densely coded, mid-level feature descriptor. The computer vision literature sometimes means object-specific components when referring to “parts”, such as in the deformable parts model [11]. However, in our case, parts describe generic structure in a small spatial neighborhood. The descriptor is by design sparse to allow for a great deal of spatial spreading, which makes the descriptor more invariant to spatial deformations.

Similar to the edges, the descriptor occurs densely and is composed of binary features. The descriptor is modeled as a mixture model, where each location has a latent variable $Z$ that determines which part appears from a codebook of parts. Parts represent generic local structure, such as a certain type of contour or junction. A codebook of parts is shared between object classes and could to a lesser extent be shared between problem domains.

Let $N_{S}(c)$ denote the set of all spatial locations in an $S \times S$ rectangle centered around the spatial location $c$. Let $X^{c} \in \{0, 1\}^{S^{2} \times E}$ denote the edge map of spread edges at each location in $N_{S}(c)$. 
The parts model at location $c$ describes a probability distribution over the edges $X^c$. The distribution is modeled as a Bernoulli mixture model, where the edge probabilities are conditionally independent given a latent variable $Z$, which indicates which part is present. For $F$ distinct parts, this gives a codebook of parts parameterized by $\theta = [0, 1]^{F \times S^2 \times E}$.

Each patch is independently considered. However, notice that two neighboring patches will have overlapping support and thus both will describe the probability of a single edge. Unlike in Amit and Trouvé’s POP model [2], this is not explicitly resolved and consequently a complete generative model on the edge level is not described.

The model in the patch around $c$ is summarized as follows

$$X^c_{l,e}|Z = f \sim \text{Bern}(\theta_{f,l,e}) \quad \forall l \in \text{Ne}_S(c), \ e \in \{1, \ldots, E\}, \ f \in \{1, \ldots, F\}.$$ 

At each location, the part with maximum log likelihood is considered

$$\hat{z} = \arg\max_z \log p(X|Z = z; \theta)$$

$$= \arg\max_z \sum_l \sum_e \left[ X^c_{l,e} \log \theta_{z,l,e} + (1 - X^c_{l,e}) \log(1 - \theta_{z,l,e}) \right].$$

(2.1)

Since we are only interested in parts with some level of edge activity in the center of the patch, we also leave room for a null hypothesis (there is no interesting part at location $c$) and reject it only if

$$\rho(X^c) > \rho_{\text{min}},$$

which sets a lower bound on the edge density

$$\rho(X^c) = \frac{1}{|E|_{\text{Ne}_{S'}(c)}} \sum_{l \in \text{Ne}_{S'}(c)} \sum_{e=1}^{E} X^c_{l,e}.$$ 

The value $S' \leq S$ is the size of an inner frame from which the edge density is calculated. A distinction is made between $S$ and $S'$, because it can be valuable to require parts to represent
a certain level of edge activity in the center of the patch. We call $\rho_{\text{min}}$ the minimum edge density threshold. In order to keep optimal values of $\rho_{\text{min}}$ fixed while changing the edge spreading radius $r_{\text{edge}}$, we may instead calculate the edge density from unspread edges. Let us call that threshold $\rho_{\text{min}}^{(\text{unspread})}$.

If the null hypothesis was rejected and we obtained $\hat{z}$, a descriptor is formulated from its indicator vector. That is, a binary vector of length $F$ where only the $\hat{z}$:th element is set to 1. If the null hypothesis was not rejected, a zero-vector of length $F$ is used instead. This gives us a densely coded local descriptor, where at each location at most one part is indicated. The formulation of using indicator vectors allows us to spread features to local neighborhoods.

### 2.1.5 Part spreading

Similar to edge spreading, coded parts are also spread in local neighborhoods. It is technically identical to edge spreading, except that for all parts $f$ a single square structuring element is used

$$K = N_{e2r_{\text{part}}+1},$$

where $r_{\text{part}}$ is the parts spreading radius. Notice that a spreading radius of $r_{\text{part}} = 2$ means a single part will be dilated to a $5 \times 5$ area.

Since the unspread descriptors are by design sparse, we can allow a large amount of part spreading, without running the risk of saturating the descriptors.

### 2.1.6 Subsampling

After spreading, neighboring descriptors are largely identical which means the feature map is filled with redundancy. Without much loss of expressiveness, we can subsample the feature map along its spatial axes. Since these will be used as part of sliding window detectors, this greatly reduces the computational overhead.
Spreading and subsampling can be equivalently rephrased as an OR (max) pooling operation centered at each subsampling location, with a pooling rectangle the same size as the spreading rectangle. This formulation is common in the literature and lends itself to faster implementations, since spreading is a costly operation compared to pooling.

2.2 Training parts

We turn our attention to how to acquire an appropriate codebook of parts $\theta$.

Take a set of images relevant to the problem domain and apply the edge extraction and edge spreading routines. Select a random set of patches of size $S \times S$ from the resulting edge maps, giving a set of training patches

$$X = X^{(1)}, \ldots, X^{(N)} \in \{0, 1\}^{S^2 \times E}.$$ 

As mentioned, the parts model is a mixture model, with a latent variable for each sample that describes which part is present

$$Z = Z^{(1)}, \ldots, Z^{(N)} \in \{1, \ldots, F\}.$$ 

Since $Z$ is unknown, a complete likelihood model cannot be calculated. Instead, the following incomplete likelihood is considered

$$L(\theta; X) = \Pr(X; \theta) = \sum_Z \Pr(X, Z; \theta),$$ 

where the summation is over all possible values of $Z$. Ideally, the codebook should be determined by the maximum likelihood estimate (MLE) of $\theta$, which is

$$\hat{\theta} = \arg \max_\theta L(\theta; X).$$
This contains the sum over all values of $Z$ and it thus intractable. An approximate solution is given using the expectation-maximization (EM) algorithm. This iterative algorithm guarantees to monotonically increase the likelihood, but fails to guarantee optimality even at convergence. Since we are only interested in maximizing the expressiveness of the parts model, it is not essential that we find the global optimum.

After training, to avoid overfitting and make sure probabilities are well-defined, we clip all $\theta$ to $[\varepsilon_{\text{edge}}, 1 - \varepsilon_{\text{edge}}]$, where we set $\varepsilon_{\text{edge}} = 0.025$.

### 2.2.1 Visualizing parts

A single part, $\theta_f \in [0, 1]^{S \times S \times E}$, is a probability distribution over oriented edges that can be hard to summarize in a single image. Since the EM algorithm can be seen as a clustering of the training patches, one way to visualize a part is to average all original patches belonging to that cluster. The color scale is stretched out from black to white, to make the shape more pronounced. However, if we use polarity insensitive edges, images of opposite facing gradients will cancel each other out and the average will be close to homogeneous gray.

This could be resolved by further clustering the original parts into two for each part, in which case it will pick up the two different gradients. However, a more principled way of doing this, with benefits potentially beyond visualization, is to do this as part of the EM directly, by coding polarity of the part as a latent variable. An example of a visualized parts model that does this is seen in fig. 2.3.

### 2.2.2 Post processing

After the parts model is trained, a few more steps are performed before they are ready to be used.

We consider each part, $f$, separately to determine whether or not it is worth keeping it
in the final codebook of parts. This is done by considering two simple hypotheses:

\[ H_0 : \theta^*_f = b_f, \quad H_1 : \theta^*_f = \theta_f, \]

where \( \theta_f \) is from the previously trained parts model and \( b_f \) is a stationary background model defined as

\[ b_{f,e} = \frac{1}{L} \sum_l \theta_{f,l,e}. \]

Let \( \ell_f^{(i)}(X) = \log \Pr(X | Z = f, H_i) \) for \( i = 0, 1 \). Then, let

\[ T_f = \mathbb{E}_{H_1}[\ell_f^{(1)}(X) - \ell_f^{(0)}(X)] = D_{\text{KL}}(\text{Bern}(\theta_f) \| \text{Bern}(b_f)), \]

be the expected value under \( H_1 \) of the log-likelihood ratio. We point out that this test statistic is the Kullback-Leibler divergence from the distribution of \( H_0 \) to that of \( H_1 \). We
Figure 2.4: Sorted parts model with 120 parts, most of them abridged. This shows the stark contrast between high-entropy parts and low-entropy parts, which is made apparent by sorting. The parts are visualized in the left-most column and in each column that follows the edge probabilities for all $E = 8$ edges are depicted with blue as probability 0 and red as probability 1. The high-entropy parts tend to default to a lighter blue, making them more forgiving of erratic edge activity.

reject $H_0$ if

$$T_f > \psi_f,$$

where the threshold is set separately for each part to allow them to compare better. We let

$$\psi_f = \psi \sqrt{\mathbb{E}_f [\ell_1 f (X) - \ell_0 f (X)]},$$

where $\psi$ is a parameter we set to 1. If $H_0$ is rejected, we keep the part in our codebook of parts. Otherwise, we discard it.

As a diagnostic tool, the parts are finally sorted in descending order of entropy. Keeping them ordered this way can be useful, since high-entropy parts tend to be correlated with one another. Having them grouped together in the feature space makes it easier to identify how they behave collectively. An example of a sorted parts model can be seen in fig. 2.4 and their entropy in an image in fig. 2.5.
Figure 2.5: In the image to the right, we first code one maximum likelihood part per location (no spreading, no subsampling). Then, the average binary entropy per edge expressed in bits is displayed for that part. Low-entropy parts are shown in red and are common along clean contours. High-entropy parts are shown in green and are common in textured areas, such as in trees and grass. Areas without a coded part are shown in white. If we instead visualize the part ordinal, qualitatively similar results are achieved due to parts being ordered by entropy. (Source: PASCAL VOC 2007 [10])
2.2.3 Optimization

Coding parts is computationally heavy and alternatives like HOG descriptors [8] are much faster. A few simple measures can be done to reduce the overhead if needed.

The edges over the patch can be subsampled, as done by Bernstein [5]. In a similar vein, the parts can also be coded on a coarser grid before pooling. Both of these perform slightly worse than their un-optimized counterpart, which is why we do not employ any of these techniques here.

There are probably more ways in which performance can be improved, for instance by eliminating parts in a coarse-to-fine manner, using some kind of decision tree. This is the subject of future work.

2.3 Latent polarity/orientation

2.3.1 Polarity

Using polarity sensitive parts, e.g. making a distinction between a bright-to-dark edge and a dark-to-bright edge, can bias the model for diverse classes such as cars. Not only will we see bright cars on dark background and dark cars on bright background, it is not uncommon to have mixed combinations with polarity flips (see fig. 2.6). This is why it is difficult to enforce global polarity consistency, without hurting generalization. However, polarity flips are still relatively rare and preferring contours that have local polarity consistency could be valuable, to separate them from weaker features.

This is not achieved if we collapse edges of opposite polarity and build a polarity insensitive parts model on top of that. Instead, we use the full $E = 8$, code polarity sensitive parts and map them to a feature space such that the patches $I$ and $1 - I$ will code to the same feature.

This is done by replacing the latent variable with $Z = (Z_{cmp}, Z_{pol})$, that now incorporates both which part and which polarity. This means that we get $F$ twin pairs of parts. If a part
matches the patch $I$, then we make it so that its twin will match the patch’s polar opposite \((1 - I)\) equally well.

This new parts model is used exactly as the old one, except the codebook now has \(2F\) parts to choose from. When we code the parts, we disregard the polarity and each part in a twin pair excites the same element in a feature descriptor of length \(F\). In short, the idea is that we move the polarity invariance up the hierarchy, so as not to create an information bottleneck at the edge level.

### 2.4 Orientation

Similar to having a latent polarity, it is also possible to add a latent orientation, so that the codebook can be built by a small set of canonical parts, where each part can be rotated in the plane. This can be combined with latent polarity, letting \(Z = (Z_{cmp}, Z_{pol}, Z_{rot})\).

Even though we do not wish to collapse all rotations, the benefits are still several. It gives a more structured model, which makes it possible to analytically rotate an object model. It also makes it possible to do edge spreading to nearby orientations, to promote invariance to small rotations.

#### 2.4.1 Training

Both polarity flips and rotations are cyclic transformations on an image patch. This means that we can generalize the training algorithm to handle any product of cyclic transformations. Normally, each training patch \(I^{(n)}\) would be converted to \(X^{(n)}\) by extracting the binary edge features. Instead, we form \(X^{(n)} = [X_{1}^{(n)}, \ldots, X_{B}^{(n)}]\), where each block \(X_{b}^{(n)}\) corresponds to the edges for a specific transformation of \(I\) (which can be both polarity and orientation, flattened to a single list of length \(B\)). Similarly, the parts model’s parameter space will follow a similar partition \(\mu = [\mu_{1}, \ldots, \mu_{B}]\).

A latent variable then dictates how the blocks in \(X^{(n)}\) should be paired with the blocks
Figure 2.6: Inside the marked box, the car is both darker and brighter than the background, causing a polarity flip along the contour of the car. (Source: UIUC [21])

in $\mu$, where the latent variable in this specific case can be interpreted as both a polarity and an orientation. This new latent variable is easily incorporated into the EM algorithm as described in detail in appendix A.
Figure 2.7: Canonical parts that are trained with latent polarity and orientation. Only one polarity and orientation is shown per part. The representative of each part is selected so that it has most of its high probabilities in the first edge.
Figure 2.8: This shows 18 orientations of a single canonical part.
CHAPTER 3
OBJECT DETECTION

This chapter covers the object detection procedure, including both training and testing. The idea of this work is to train the detector entirely with images of the objects rendered from CAD models. Secondary emphasis is placed on training with a small sample size.

First, we introduce a general training procedure that can be used with real images. Then, in §3.2, we start discussing how to adjust this procedure when training from CAD models.

3.1 Training from real images

3.1.1 Training

Assume we have a set of training images $I = I^{(1)}, \ldots, I^{(N)} \in I$, each associated with a label $Y = Y^{(1)}, \ldots, Y^{(N)} \in Y$. The labels $Y = \{bkg, obj\}$ describe what a training image depicts, choosing between background (bkg) and the object class (obj). All images are assumed to be of the same size and if the object is present it is assumed to be well-posed in the center of the image. Background images are any images from the same image modality, that does not include the object class.

Using the feature extraction function $\phi$ from chapter 2, each image is converted to a feature map to form $X = \{\phi(I) : I \in I\}$. This is not to be confused with earlier notations for $X$ when training the parts model. For each $n$, a sample $X^{(n)} \in \{0, 1\}^{W \times H \times F}$ is a binary feature map that represents an instance of an object or background. In summary, this gives a training set

$$(X^{(1)}, Y^{(1)}), \ldots, (X^{(N)}, Y^{(N)}) \in X \times Y.$$

To deal with the greatest intra-class variation, the object model is described as a Bernoulli mixture model with $M$ components. In the final detection, which component of the mixture model was used is ignored. A component can be characterized as capturing a particular
camera pose of the object, since this offers the greatest source of variation.

We get the following model

\[ Z \sim \text{Cat}(\pi_1, \ldots, \pi_M) \]
\[ X_{l,f}|Z = m \sim \text{Bern}(\mu_{m,l,f}), \quad \text{(conditionally independent)} \]

for all \( m = 1, \ldots, M \) and all binary features indexed by spatial location \( l \) and feature \( f \).

Training can be done in the same way as the parts model using the EM algorithm. Only positive samples \( (Y = \text{obj}) \) are used in the training procedure.

The EM algorithm can be made into a clustering algorithm, by making hard assignments of each sample according to their maximum posterior probability. Under this formulation, the EM algorithm is made to find a partition of \( X_{\text{obj}} \), the subset of \( X \) containing only objects. We get

\[ X_{\text{obj}} = X_1 \cup \cdots \cup X_M. \]

Expressed this way, each cluster is the MLE estimate of the samples in its respective partition, which for a Bernoulli is given by taking the average, yielding the object model

\[ \hat{\mu}_{m,l,f} = \frac{1}{|X_m|} \sum_{X \in X_m} X_{l,f}, \quad (3.1) \]

for \( m, l, f \) as before. This is not quite the same as taking \( \hat{\mu} \) directly from the EM algorithm. However, in practice it will be indistinguishably close to it, since generally most posterior probabilities are degenerate after convergence, especially at this dimensionality. Going the route of taking averages of partition clusters makes it easier to explore alternative ways of arriving at a good partition. It also makes each cluster component more interpretable, since it is simply the average of a collection of samples. The partition can also be used when other methods are considered, such a separate SVM for each component.

A background model is similarly trained from \( X_{\text{bkg}} \), the set of background samples. We
do not partition the background data, even though it could give a richer model that captures
strong false positives.

Since we have a single background model, we assume that it should not pick up on any
spatial structure. We can thus collapse the spatial locations to give

\[ \hat{b}_f = \frac{1}{|X_{bkg}|} \cdot L \sum_{X \in X_{bkg}} \sum_{l} X_{l,f}, \]

for all \( f \).

### 3.1.2 Handling degenerate probabilities

To force the logarithms to be defined and well-behaved, all probabilities are clipped to lie in
\([\varepsilon, 1 - \varepsilon]\), where \( \varepsilon \) is set to a positive value close to 0. Since the coding step has a winner-
takes-all aspect to it, if the number of parts is increased, so is sparsity. As the feature vectors
become sparser, setting \( \varepsilon \) too high becomes a risk.

To accomodate this, before clipping the background model \( b \), we calculate the average
part activity on natural images between all parts and call it \( \bar{b} \in (0,1) \). The minimum
probability parameter \( \varepsilon \) is then set as

\[ \varepsilon = \varepsilon_{bkg} \bar{b}, \]

where \( \varepsilon_{bkg} \) is a parameter set around \( \frac{1}{2} \). It would also be possible to set \( \varepsilon \) to a certain
percentile of the unclipped \( b \), except this does not guarantee that \( \varepsilon > 0 \).

Relating \( \varepsilon \) to the sparsity, means that as the number of parts is increased, the minimum
probability is adjusted appropriately. If instead \( \varepsilon \) would be set to a fixed value, it would
make it harder to examine the influence on parts count, since one could simply be a better
match for the particular value chosen for \( \varepsilon \).
3.2 Training form CAD models

3.2.1 Introduction

The technique that we employ involves pre-rendering each model according to a set of camera parameters. The object is rendered onto an image with an opacity (\( \alpha \)) channel, such that the background is completely transparent. As a consequence, the \( \alpha \) channel describes the support of the object.

In this work, from this point on the CAD objects are not used in any additional way, so the same techniques can be applied to any training set where the support of the object is similarly defined.

The way we will use the \( \alpha \) channel, is by superimposing the objects onto background images (as seen in fig. 3.1) and using the composite images for training. By doing this, the features we detect around the contour of the object should mimic what we might see in testing. It may seem that by creating this composite we are throwing away the information about the support, which is one of the benefits of using CAD models. However, as we shall see, this information is retained in the object and background models by using the same background images for both compositing and for the background model.
If the task is extended from object detection to pose estimation, using CAD images is greatly beneficial, since the training data can be automatically annotated with pose parameters. The same applies if we want to do image segmentation after the object has been detected. Some of these avenues are explored in related work as described in §3.5.

### 3.2.2 Training

In contrast to before, we now have $I = I_{\text{cad}} \cup I_{\text{bkg}}$, where $I_{\text{cad}}$ are CAD-rendered images with an $\alpha$ channel. We assume that background images are easily generated so that we can make sure that $N = |I_{\text{bkg}}| = D |I_{\text{cad}}|$, where $D$ is a parameter that will soon be explained.

Given a background image $I_{\text{bkg}} \in I_{\text{bkg}}$ and a CAD image $I_{\text{cad}} \in I_{\text{cad}}$, we will write

$$I_{\text{sup}} = I_{\text{cad}} \text{ OVER } I_{\text{bkg}}$$

to denote superimposing the CAD-rendered object over the background using alpha compositing. Using this operator, we define a new set

$$I_{\text{obj}} = \{I_{\text{cad}}^{(\lfloor n/D \rfloor)} \text{ OVER } I_{\text{bkg}}^{(n)} \mid n = 1, \ldots, N\}.$$ 

In words, we take each CAD-rendered image and superimpose it onto $D$ background images. Setting $D > 1$ allows us to get a smoother model and a better estimate of the background model, in cases where we train with few samples.

From here, we follow a similar training procedure as for real images with a few things to keep in mind. First, the partition of $X_{\text{obj}}$ using EM can be done before superimposing, by either ignoring the $\alpha$ channel, or include it in the calculation of the gradients.

Since our definition of $I_{\text{obj}}$ associates each CAD image with a background image, a partition of this set defines a partition on both the CAD images and the background images,
yielding

\[ X_{\text{obj}} = X^{(1)}_{\text{obj}} \cup \cdots \cup X^{(M)}_{\text{obj}}, \]
\[ X_{\text{bkg}} = X^{(1)}_{\text{bkg}} \cup \cdots \cup X^{(M)}_{\text{bkg}}. \]

From here, as in (3.1), giving \( \hat{\mu}_{m,l,f} = X_{m,l,f}^{(\text{obj})} \) and \( \hat{b}_{m,l,f} = X_{m,l,f}^{(\text{bkg})} \). This time we keep the background model uncollapsed and separate for each cluster, so that outside the influence of the object, the object model is identical to the background model. This will only yield a performance improvement for small sample sizes, where the background model is not accurately estimated. It is also a useful diagnostic to be able to study the exact influence on the feature space of an object on top of background.

The same background images can be used for each cluster, which reduces the number of background images needed, even though it is never particularly big.

### 3.3 Testing

#### 3.3.1 Classification

Before describing the entire detection pipeline, at its core will be a classifier for each object component. That is, given a sample \( X \in \mathcal{X} \), the classifier should determine the associated label \( y \in \{\text{obj, bkg}\} \). Due to the nature of the testing procedure, we shall require the classifier to give its assessment as a continuous variable (called the decision function). We will take a greater value to mean higher certainty that \( y = \text{obj} \).

At this point, for a given mixture component, we have estimated the parameters for a null hypothesis (\( \hat{b} \)) and an alternate hypothesis (\( \hat{\mu} \)). A starting point for our decision function is a log ratio test (LRT) statistic

\[ R(X) = \log \frac{p(X|Y = \text{obj}; \mu)}{p(X|Y = \text{bkg}; b)} = \sum_{l,f} X_{l,f} \log \left( \frac{\mu_{l,f} 1 - b_f}{b_f 1 - \mu_{l,f}} \right) + \text{const.} = \mathbf{w}^T X + w_0 \]

30
Ideally, $R$ could be used directly as the decision function. However, the naive assumption of conditional independence between the features makes $R$ largely overestimated and misaligned. Since the decision function can be of arbitrary scale and is only important to establish an ordering of classifications, any monotonically increasing function can be used to transform it without changing the results. However, since each object component has its own LRT statistic (let us denote each $R_m$), different amounts of overestimation and misalignment will cause the components to unfairly compare with one another.

As an example, if one component consistently overestimates $R(X)$ more than other components, there will be a large number of false positives that are higher ranked than many true positives for the other components. Such an imbalance is devastating to detection results.

There are two main ways this problem is addressed. First, the dimensionality of $X$ can be reduced in such a way that the conditional independence assumption becomes more true in practice. Secondly, the test statistics $R_m(X)$ can be standardized. These two approaches can be used separately, but work well when used in combination.

### 3.3.2 Standardization

Consider

$$R = w^T X + w_0,$$

where this time $X$ is a random vector (instead of a sample) and thus $R$ is a random variable. In this formulation, the central limit theorem tells us that if the sum of $L \times F$ elements is large enough (which it is), we have that

$$R \sim \mathcal{N}(\mu_R, \sigma^2_R),$$

for some $\mu_R$ and $\sigma^2_R$, which will depend on the distribution of $X$ and the coefficients.
With this in mind, $R$ can be standardized by

$$R^* = \frac{R - \mathbb{E}[R]}{\sqrt{\mathbb{V}[R]}}.$$ 

This can be done both analytically and experimentally. In either case, since the distribution over $R$ depends in turn on $X$, we must choose the distribution from which to draw $X$. There are several possibilities, but we find drawing $X$ from the background distribution ($Y = \text{bkg}$) to give the best results. This is done analytically as

$$R^*(X) = \frac{R(X) - \mathbb{E}_{X|Y=\text{bkg}}[R(X)]}{\sqrt{\mathbb{V}_{X|Y=\text{bkg}}[R]}} = \frac{\mathbf{w}^T(X - b)}{\sqrt{\sum_{l,f} b_f (1 - b_f) w_{l,f}^2}}.$$ 

It can also be done using sampling by drawing natural samples $\mathbf{X}_{\text{neg}} = X_{\text{neg}}^{(1)}, \ldots, X_{\text{neg}}^{(K)}$ from negative data and taking

$$R^*(X) = \frac{R(X) - \hat{\mathbb{E}}[R(X)]}{\sqrt{\hat{\mathbb{V}}[R(X)]}}, \quad (3.2)$$

where $\hat{\mathbb{E}}$ and $\hat{\mathbb{V}}$ are the point estimates of the mean and the variance from the data $\mathbf{X}_{\text{neg}}$.

A potential problem with both of these methods is that we are standardizing around average negatives and not strong negatives. Due to non-maximal suppression, which will be described in §3.3.7, the negatives in our final detections can be seen as drawn from an extreme value distribution of random negatives. This distribution will be sensitive to the shape of the right tail of the negatives, which could be problematic if it does not follow a perfect Gaussian. As an attempt to make this more robust, we can instead make unstandardized detections in background-only images for each component separately. We then take these detections to be $\mathbf{X}_{\text{neg}}$ and standardize as in (3.2).

Without using key points, as will be described shortly, the analytic estimate will not be good enough and it is better to use sample estimates. However, the correlation patterns of $X$ will not distort $R$ evenly for each element of $X$, which means that even though $R^*$
Data: weights \( w \), features indices \( D = \{(l, f) : w_{l,f} \in w\} \), suppression radius \( r_{kp} \), minimum weights \( \varepsilon \)

Result: key points \( D_{kp} \subset D \)

begin

\[ D_{kp} \leftarrow \emptyset \]
\[ w_{pos} \leftarrow \max(0, w) \]
\[ w_{neg} \leftarrow \max(0, -w) \]

for \( w' \in \{w_{pos}, w_{neg}\} \) do

\[ (l, f) \leftarrow \arg \max_{(l,f) \in D} w'_{l,f} \]

if \( w'_{l,f} \leq \varepsilon \) then

break

end

\[ D_{kp} \leftarrow D_{kp} \cup \{(l, f)\} \]

for \( l' \in \text{Ne}_{rkp}(l) \) do

\[ w'_{l',f} \leftarrow 0 \]

suppress locally around \( l \) in a square with radius \( r_{kp} \)

end

end

algorithm 1: Key point algorithm.

may appear as a standard normal, the influence of some elements of \( X \) are overestimated as compared with others. To address this, we use key points.

3.3.3 Key points

The idea of key points is to reduce the dimensionality of \( X \), which not only will make detection faster, but also help make the independence assumption more true. Naturally, features that are spatially close by are heavily correlated. If a line contour is observed, it is likely that the line continues in the closest neighbors. Not only that, but since we do local feature spreading, we are injecting even more correlation structure into our samples.

To cope with this, a subset of the indices of \( X \) is selected using a sequential non-maximal suppression algorithm. The idea of the algorithm is to start by picking one element of \( X \) where the corresponding value of \( w \) has the largest absolute value and thus deemed a discriminative feature. After that element has been picked, the immediate neighborhood of
that point is suppressed from ever becoming picked as a key point. This algorithm continues until all elements are either key points or suppressed. The algorithm is described in detail in algorithm 1.

3.3.4 Minimizing correlation between features

Key points alleviate the problem of spatially correlated features. However, there is also the problem of features that tend to show up together, giving correlation structure across parts. This can for instance happen if two parts are shifts of each other, so that when one is present, the other will most likely be present too at the same relative position as the shift. It is also generally the case that high-entropy parts are positively correlated, since textured areas occur in contiguous patches.

Because of reasons like this, the log-ratio test (LRT) model is sensitive to the choice of descriptor. This is part of the reason why we use our own parts descriptor, since we can make sure it works well in this context.

3.3.5 Detection

To perform object detection, the classifier is applied at every possible location and scale in the image. This is done by first resizing the image $I$ several times to form an image pyramid \{I, g(I), g^2(I), \ldots\}, where $g$ makes $I$ smaller by a factor $2^{1/\lambda}$. This means that there will be $\lambda$ steps per octave (factor 2). The image is resized until deemed too small.

The resizing operation $g$ first applies a Gaussian filter and then subsamples at an even grid. The pre-filtering is important to avoid detecting unwanted edges due to aliasing.

At each scale level $k$, the octave number is $s = k/\lambda$ and the resize factor is $2^s$. Features are extracted by taking $\phi(g^k(I))$. The result is referred to as the feature pyramid. To be able to make detections of objects that are partially sticking out of the image, the feature map may also be enlarged by padding the spatial borders with zeros.

Each rectangle of size $w$ in the feature pyramid can now be evaluated using the decision
Figure 3.2: Example of a response map for a car detector, showing two peaks over the objects.

function. Since the decision function is linear, the image of all decision functions can be formulated as a cross-correlation operation\(^1\) between a feature image \(\phi(g^k(I))\) (at each scale level \(k\)) and \(w\). Due to this signal processing formulation, the weight vector \(w\) is sometimes referred to as a filter. Note that in the case of using key points, the support of the filter \(w\) will not be a spatially contiguous block and it is thus faster to implement using explicit loops over the key points.

We call the resulting signal after standardization a response map, which contains the decision function at each spatial location for that level. An example of a response map is shown in fig. 3.2. At each point in the response map, we create a detection, which consists of a bounding box and a score set to the value of the response map. The bounding box will be centered around its corresponding location in the response map, but the size must be determined somehow.

### 3.3.6 Bounding box

As part of the model training, the size of the bounding box is pre-determined for each object component. When using CAD models, for each positive sample \(X^{(n)} \in X_{obj}\), the exact support is known along with its bounding box, \(B^{(n)} \in B\). A simple way to determine the

---

\(^1\) The community often refers to this step as a convolution, even though this is technically not correct.
appropriate bounding box size is to average all the bounding boxes in $B$, each side and axis separately. Another way, which we use, is to pose a simple optimization problem with the criterion for detection (1.1) in mind. The optimal bounding box becomes

$$\hat{B} = \arg \max_B \sum_{n} 1\{\text{overlap}(B,B^{(n)}) \geq \frac{1}{2}\}.$$  

This uses a zero-one loss, but the optimum can adequately be approximated by exhaustively testing all $B \in B_{\text{obj}}$ along with a few dilations and erosions thereof. This is assumed to be done for each object component separately, using $B^{(1)}, \ldots, B^{(M)}$ to produce $\hat{B}^{(1)}, \ldots, \hat{B}^{(M)}$.

When a detection is made, the bounding box at a certain point in the pyramid is taken to be $\hat{B}$ resized by the inverse of the image resize factor.

### 3.3.7 Non-maximal suppression

Creating a detection at each spatial location creates an abundance of false positives, which is damaging to detection rates due to the way the scoring works. To cope with this, a large portion of bounding boxes are suppressed through a routine called non-maximal suppression. For this, we use a sequential algorithm that is similar to the key point algorithm, in which the best detection, according to the standardized test score $R^*$, is selected and all neighboring bounding boxes are discarded. If $B_0$ has been selected, then a $B$ is deemed to be a neighbor of $B_0$ if $\text{overlap}(B_0, B) \geq f_{\text{bb}}$, where $f_{\text{bb}}$ is left as a parameter (typically around $\frac{1}{2}$). This process is repeated until all bounding boxes are either kept or discarded.

### 3.3.8 Detection of small objects

When extracting the image pyramid, the question arises what the smallest detectable object should be. A natural lower bound is to avoid upscaling the source image, so that if the model is trained using images of size $100 \times 100$ pixels, that becomes the smallest detectable object. This eliminates potentially important detections, so it is not good enough.
A tried-and-true approach is to train separate models specifically for small-object detections. For instance, the deformable parts model (DPM) [11] adds one or two more smaller versions of the model, keeping separate bias terms to ensure the models will compare well. This could be done in our context as well, but instead of the bias terms, we would rely on the standardization alone. Even though this is a perfectly viable approach in our situation, it has not yet been tested and we rely so far on upscaling.

If we look at the bounding box sizes of all positives in the PASCAL VOC training set, excluding truncated objects, we get a histogram as seen in fig. 3.3. However, since the number of detection windows at octave $s$ is proportional to $2^{-2s}$, it means that the opportunities of hitting false positives increase exponentially for each additional level on the wider end of the feature pyramid. This increased risk is not matched with an increased reward. Instead, the number of objects per levels stays roughly the same, even though it does have a distinct bump in the middle.

There are many reasons why we do not see an increasing number of objects at smaller scales. Even in a toy world where voxels are uniformly distributed in 3D and the camera is randomly placed, we would see a decreasing voxel density with a decreasing voxel size, since the risk of occlusion by larger voxels becomes greater and greater. This example flirts
Figure 3.4: Histograms over standardized log-likelihood for detecting a small (left) and big (center) object. The standardization is done all scales together and as a result the mean and standard deviation is not necessarily 0 and 1, respectively. (right) Shows how the mean and standard deviation varies with scale \( s \).

with the idea of a theoretical model for a scale prior. However, the objects of the real world are not uniformly placed and cameras are generally operated by humans with objectives in mind, which makes simple models like this fall short.

Regardless of how it is estimated, the idea is to incorporate this prior distribution over scales into the model, which would act to suppress scores at small object sizes in particular.

### 3.3.9 Scale prior

We could try to incorporate this prior into the model, treating \( S = s \) as an observable random variable of the scale, giving a new log-likelihood ratio

\[
R_{sp}(X) = \log \frac{p(X|Y = \text{obj}, S = s; \mu)p(Y = \text{obj}|S = s)}{p(X|Y = \text{bkg}, S = s; b)p(Y = \text{bkg}|S = s)}
\]

\[
= \log \frac{p(X|Y = \text{obj}, S = s; \mu)}{p(X|Y = \text{bkg}, S = s; b)} + \log p(Y = \text{obj}|S = s)
\]

\[
\equiv R(X) + \log p(Y = \text{obj}|S = s),
\]

where the last step is valid only if we choose to accept that the object and background distributions do not change with the scale. With this premise, we could use fig. 3.3 (right) as a non-parameteric estimate of \( p(Y = \text{obj}|S = s) \), by approximating the logit with a log
so that we only have to know \( p(Y = \text{obj}, S = s) \) up to a multiple. We may approximate the logit as such since the probability of background as opposed to object at any given window is close to 1.

Even though this offers an improvement, experimentally we have seen that this penalizes the smallest and the biggest objects too much. A possible explanation for this is that the step in (3.3) marked with a question mark is not valid. The intuition behind this is that when detecting small objects, many areas of the image are completely devoid of feature activity and thus not real contenders. Similarly, at scales where the biggest objects are detected, the padding-to-image ratio is the highest, introducing areas devoid of feature activity. As a result, areas with most contenders are in the middle range, which would make the background distribution different there. Even if this explanation might be an oversimplification, it is easy to see in fig. 3.4 that \( p(X|Y = \text{bkg}, S = s) \) is as expected peaked in the middle range.

Looking at (3.3), this middle bump should boost the log ratio for small and big objects. However, since the experiment used \( R(X) \) that disregards \( s \), this boost was never expressed. Incorporating this properly would require standardizing \( R(X) \) differently depending on \( s \). It should start to become apparent that to get this just right requires a major complication to the model, which will involve multiple standardizations and a careful estimate of the scale prior.

Instead, in the interest of good results and focus elsewhere, we take a practical approach and keep \( R(X) \) fixed and use a simple scale penalty that use a single parameter and works well in practice. Experimentally we arrived at

\[
R^{**}(X, s) = R^{*}(X) - \alpha_{\text{sp}}2^s,
\]

where \( \alpha_{\text{sp}} \) is the scale penalty parameter and \( 2^s \) is image resize factor. We will use the same to penalize the score of SVMs, even though the parameter will need rescaling.

Since the DPM defines separate biases associated with different values of the scale level
of scale penalty. As a related read, the scale statistics for the pedestrian class is discussed in [9].

3.4 Support Vector Machines

For comparison, the Support Vector Machine (SVM) is used to train a separate detector. A linear SVM is used because it is fast and it makes the parameter space identical to that of the LRT model, which allows for direct comparison. Linear SVMs are commonly used for object detection [8, 11].

To further facilitate comparison, the same method of partitioning $X$ into mixture components is used and training is done on each component separately. The training is done as follows:

- $N_{\text{neg}}$ negative samples are randomly selected and together with $N_{\text{pos}}$ positive samples from a given component are used to train a linear SVM that we will call the base detector.

- This detector is then used to scan for strong false positives across the entire image pyramid of each negative image. This is done by keeping a heap of the top-$N_{\text{fp}}$ scoring false positives while scanning a large set of background-only images.

- These false positives are added to the original training data, giving a training set of size $N_{\text{pos}} + N_{\text{neg}} + N_{\text{fp}}$. A new linear SVM is trained, giving the final model parameters.

This is the same procedure described in the original HOG paper [8]. An alternative is to keep the base detector and train one or many cascade detectors from the positives and the strong false positives alone. Each detector is then used in testing, by starting with the base detector and only passing it along to the first cascade detector if the score is above a certain threshold. This is the general structure of the seminal Viola-Jones detector for facial detection [25].
This type of cascade approach has not proven as performant, or possibly just harder to get right, within the context of this work. This same conclusion was made by Dalal and Triggs [8].

Each time the SVM is trained, 20% of the training data is held out and a grid search is done to determine the optimal value of the regularization parameter, trained on the 80% and tested on the held out 20%. Unsurprisingly, it is common for the optimal amount of regularization to change between the base detector and the final detector. Generally, the second step requires more regularization than the first.

3.5 Related work

This section presents a survey on related work where CAD models are similarly used to train object detectors.

In the work of Liebelt et al. [19], extracted feature descriptors from CAD models are associated with 3D locations, creating a 3D feature map. The descriptors are clustered using \( k \)-means and each cluster has a list of viewpoints, along with weights, where it is present. Initially, a coarse detection is performed by disregarding spatial locations in a bag-of-feature manner. Then, extrinsic camera parameters that best match the 3D feature map to the 2D features are found using RANSAC. The same intrinsic camera parameters are used as when rendering the objects.

Unlike [19], which uses photorealistic rendering, Stark et al. [24] render edges directly. In their approach, CAD models are manually annotated with parts (e.g. left front wheel, windshield, etc.). For each part, a dense version of shape context [3], reminiscent of HOG [8], is used to form feature descriptors. The individual part detectors are trained using AdaBoost and phrased as conditional probabilities using Platt scaling, which is a technique to formulate a probability distribution from a decision function. Finally, they pose a probability distribution over an entire object model using the constellation model. This model is more expressive than a star-shaped parts model (as in [12]), but it makes inference harder. They
solve this by using Metropolis-Hastings sampling.

This method is further developed by Zia et al. in [27], in which each CAD object is now annotated with a coarse wireframe representation of the object, whose edge structure is shared between all CAD models of that object category. Using shape-based analysis [7], a probability distribution is formulated over instantiations of this wireframe model, which is incorporated into the constellation model. The probability distribution over part locations is also made richer, by explicitly considering camera parameters, similar to [19]. Again, inference is expensive, which is why they use the detector from [24] first to find coarse detections. They then refine the detections using the new model with a sampling method similar to the condensation algorithm [16].

Another take on the parts model is offered by Schels et al. [23], who determine part locations greedily based on areas of high mean gradient across the training data. The part appearance models are trained using HOG descriptor and linear SVMs. Part detections are combined into a multi-view object detector using an SVM with an intersection kernel [15]. The intersection kernel gives a measurement of how close two sets of parts match, by comparing histograms at several different levels of coarseness. When the bins of these histograms are over spatial axes the technique is called pyramid matching [17]. The idea of using several different histograms, is so that if two matching parts are not in the same bin at the finest level, they can still match at a coarser level and contribute to the similarity measurement. In the context of our model, this would correspond to using several different part spreading radii to create separate models; the models are then tested separately and combined into a single score.

Pepik et al. in [22] extend the deformable parts model (DPM) by Felzenszwalb et al. [12] in two ways that are facilitated by having synthetic training data: (1) Their first extension is to include a viewpoint category parameter as a label in their model. As in [6], instead of using the hinge loss, they phrase the problem as a structured SVM with a $\Delta$ loss function that favors bounding boxes that overlap well according to the overlap measure used in
the PASCAL VOC challenge [10]. They add a mis-classification of viewpoint as a fixed penalty in the $\Delta$ function. This formulation is particularly suited for synthetic training data since the viewpoint is known. (2) Furthermore, they also replace the parts representation with axis-aligned 3D bounding boxes. This requires part placements to be optimized over object instances (from multiple viewpoints) as opposed to over individual images. This necessitates the training data to be annotated with instance IDs and projection matrices, both of which are easily accessible when synthesizing data from CAD models. The extensions to the DPM focus on improving viewpoint estimates, as well as doing ultra-wide baseline matching. However, for the task of bounding box prediction that we are interested in, the results are not as appealing. With only CAD-based training data, their results are much lower than real data. They see a slight improvement over real data when training with both real data and CAD data together.
CHAPTER 4
EXPERIMENTAL RESULTS

This chapter is divided into four sections: (1) An overview of the datasets we use for testing. (2) The parts descriptors are evaluated in order to make a decision what to use for the rest of the experiments. (3) Results are presented on the UIUC dataset with particular focus on small sample sizes. (4) Finally, results on PASCAL VOC are presented for the classes car and bicycle.

4.1 Datasets

We refer to the following datasets

INRIA Person Dataset with upright people. [20]

UIUC Car Data (single scale). 200 cars in profile view across 170 images. All cars are roughly the same scale and of similar type. [21]

PASCAL VOC 2007 contains just under 10,000 images, divided evenly into training/validation and testing. Each image has a set of objects from 20 classes, among which we focus on the classes car and bicycle. [10]

4.2 Descriptor evaluation

This section will determine the performance of the different parts models. The parts descriptors are also compared with the popular Histogram of Oriented Gradients (HOG) descriptor [8].

To avoid confusion, we report all results for the descriptors in such a way that a lower value is better. For instance, instead of average precision (AP), we consider $1 - AP$. 

44
4.2.1 Tests

The objective is to choose a descriptor that is versatile and thus performs well on several datasets. In a production setting, it would be wiser to optimize all parameters for a specific task. However, in a research setting, versatility is useful lest we constantly second-guess the choice of descriptor when working on various tasks.

The descriptors are compared on three separate tasks (named in lower-case to distinguish from the datasets):

inria The task of the INRIA dataset, phrased as a detection task instead of a classification task. A classification task would be better, since it focuses on the performance of the descriptor, without involving things like bounding box prediction and non-maximal suppression. The original classification task specified by the INRIA dataset has a fixed set of positive samples and extracts negatives by exhaustively scanning the feature pyramid of pure-background images at specific strides. This constitutes a fixed sets of windows (samples), which is then the subject of a classification task. That is, each window must be classified separately, disallowing any change to strides or using non-maximal suppression. A classifier is evaluated by considering the miss rate at different FPPW (false positives per window). This test unfairly hurts blurry response maps, in the sense that a false positive can be counted several times if neighboring windows share similar score. Instead, we treat it as a detection task and use FPPI (false positives per image). The results are summarized as average miss rate, which is the average of the miss rate at 9 specific FPPIs evenly in log-space between \(10^{-2}\) and \(10^0\) (as established by [9]).

uiuc An LRT model is trained using 15 images of CAD cars (30 including mirroring) and tested on the UIUC dataset. Since the scale is known, only one scale is scanned.

vvp The same model as in the uiuc test is used on the validation set of the PASCAL VOC 2007 [10], taking the subset of cars annotated with poses Left or Right. Only images
with at least one positive is included, which means pure negatives are excluded. The results on this dataset is particularly weak, since it was trained for the UIUC (with limited car models) and small levels in the feature pyramid is ignored. However, the relative comparison is still valid.

4.2.2 Parts configurations

The following parts models were considered:

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
<th>$E$</th>
<th>Polarities</th>
<th>Orientations</th>
<th>Training data</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain/natural</td>
<td>240 (230-232)</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>natural</td>
</tr>
<tr>
<td>plain/synthetic</td>
<td>240 (233-239)</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>synthetic</td>
</tr>
<tr>
<td>polarity/natural</td>
<td>240 (240)</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>natural</td>
</tr>
<tr>
<td>polarity/synthetic</td>
<td>240 (212-239)</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>synthetic</td>
</tr>
<tr>
<td>rotation/natural</td>
<td>252 (234-252)</td>
<td>8</td>
<td>2</td>
<td>18</td>
<td>natural</td>
</tr>
<tr>
<td>rotation/synthetic</td>
<td>252 (198-216)</td>
<td>8</td>
<td>2</td>
<td>18</td>
<td>synthetic</td>
</tr>
</tbody>
</table>

Length refers to the length of the descriptor. Due to parts rejections, the length that we request is only an upper bound and the range of the actual descriptor lengths across all trials is shown in parentheses. Number of edges $E$ is described in §2.1.2 and polarities and orientations in §2.3. Each model can be trained from either natural patches taken from the PASCAL VOC (natural) or randomly generated synthetic data as shown in fig. 4.10 (synthetic). Training from synthetic data produces less high-entropy parts as seen in fig. 4.1.

The most important parameters shared between all configurations are summarized below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge spreading radius, $r_{\text{edge}}$</td>
<td>1 (orthogonal)</td>
</tr>
<tr>
<td>$k_{\text{min}}$</td>
<td>5 (out of 6)</td>
</tr>
<tr>
<td>Minimum contrast threshold, $\kappa$</td>
<td>0.05</td>
</tr>
<tr>
<td>Part size, $S \times S$</td>
<td>$6 \times 6$</td>
</tr>
<tr>
<td>Inner frame, $S' \times S'$</td>
<td>$2 \times 2$</td>
</tr>
<tr>
<td>Background threshold, $\rho_{\text{min}}^{\text{(unspread)}}$</td>
<td>0.15 unspread edge density</td>
</tr>
</tbody>
</table>
Figure 4.1: Median of binary entropy/edge (in bits) for the various parts models. The error bars show the interquartile range. Training from synthetic data greatly reduces the amount of high-entropy parts. Introducing latent transformations has a similar, but milder, effect.

These parameters are slightly different from Bernstein’s [5]. Bernstein used $9 \times 9$ parts and subsampled the edges at $3 \times 3$ spatial locations, since it makes coding parts much faster. This is a trade-off that hurts how well the parts perform on tasks, so we accept the performance hit and avoid this. Without this reduction, $9 \times 9$ is a rather big patch size and the edge spreading of 3 only decreases granularity.

### 4.2.3 HOG descriptor

When compared to the HOG descriptor, the version used is the same as that of Felzenszwalb et al. [11], which is described in detail by Girshick in [14]. We use the polarity insensitive version with a descriptor length of 13, where 9 of the features capture gradient activity for 9 polarity insensitive orientations (over 180 degrees) and 4 of the features the overall gradient activity in the four quadrants around the descriptor center. The block size is $4 \times 4$, to make the parameters space comparable to the parts descriptors.
4.2.4 Results

The comparison between the different parts models with descriptor lengths around 240 can be seen in fig. 4.2. Since there is a random element to the training of a parts model, we repeat each experiment six times with different initialization seeds for the EM.

Surprisingly, the plain parts without any latent transformation is generally the best performing. This is particularly true for the \textit{inria} task. However, when using the LRT model on sideview cars (\textit{uiuc} and \textit{vvp}), the difference between \textit{polarity} and \textit{plain} are not statistically significant.

Introducing latent orientation in addition to polarity sees a further increase in miss rate, especially on the \textit{inria} task. The way the \textit{rotation} model is trained gives it less high-entropy parts. These parts tend to show up in trees and other areas with plenty of erratic feature activity. As a result, they also tend to be highly correlated both spatially and between different high-entropy parts. If these parts are more common over trees than the object of interest, it could be a good indicator and thus valuable for discrimination. Trees are a notorious source of false positives, so being able to eliminate them and similar negatives could be what the \textit{rotation} parts are missing.

However, since these parts are highly correlated between parts, the LRT model will not deal with this as well as the SVM. This could give us insight why we see such an improvement on \textit{inria}, but not nearly as much on \textit{vvp} and \textit{uiuc}.

There is no significant difference between using synthetic and natural images for the \textit{rotation} parts. If we instead have only latent polarity, the natural images are starting to stand out as a better choice. This trend continues if we use no latent transformations at all. Something similar happens when training from clean synthetic data, since the edge activity will always be well behaved. This could explain why the \textit{rotation} parts perform equally well regardless of training data, as well as why they are not as competitive as \textit{plain} parts.

In fig. 4.3, the evidence is clear that more parts improves detection rates. However, since it becomes prohibitively more difficult to train parts and object models beyond a descriptor
length of 480, it is unclear how the trend continues beyond this point. The benefits of the plain parts are consistent on inria. However, the opposite is true for vvp and uiuc, particularly for descriptor length 30 and 60. For 240 and above, the differences are not statistically significant.

Going forward, we will use polarity parts for the LRT model and plain parts for the SVM model. Whenever possible, we will use a descriptor length of 480, unless otherwise stated.

4.3 UIUC

Training using CAD models works well on the UIUC dataset and we are able to achieve the same results as Bernstein and Amit [4]. Not only that, but due to the way the training samples are arranged, much fewer positive training samples can be used while still achieving good results.

In the following experiment, we draw $N$ samples (without replacements) from 15 images of CAD-rendered sideview cars. Each sample is flipped and thus effectively $2N$ positive images are used in training. Each positive image is placed on top of $D = 25$ different randomly chosen backgrounds. When $N$ is small, the model will be sensitive to the random selection of positives, so we repeat the experiment 15 times. The medians of all trials for each $N$ is summarized in fig. 4.4.

4.4 PASCAL VOC

4.4.1 Camera poses

As previously discussed, when training from synthetic data, it is not clear what distribution of camera poses from which to generate the data. Two distributions are considered:

uniform Samples are generated evenly along azimuth angles ($0..360^\circ$ in steps of $10^\circ$), altitude angles ($0..40^\circ$ in steps of $20^\circ$) and three different camera angles-of-view (these poses were modelled after [19]). A mixture model is trained to cluster the data into
Figure 4.2: The parts models in §4.2.2 are tested on the three tasks described in §4.2.1. Each test was repeated 6 times and averages are presented with error bars marking one standard deviation in each direction. The parts models use a descriptor length of 240. Lower is better.
Figure 4.3: The *plain* and *polarity* parts models are compared, trained on natural images and for different descriptor lengths. Since the parts model can eliminate parts during training, actual descriptor lengths can be lower than the ones reported. The colors are congruent with fig. 4.2. Lower is better.
Figure 4.4: 1 − AP (average precision) on the UIUC dataset. For each sample size $N$, 20 experiments were run where $N$ images out of 15 possible were randomly selected. The dots with text show the median of all the trials and the error bars the interquartile range. Note that we must be careful when interpreting the interquartile range for larger $N$. For instance, there is only one way to select 15 samples out of 15, so for $N = 15$, there is no variability at all. Lower is better.
$M$ components. When fewer training samples are reportedly used, a subset with this size is randomly selected. Note that if $N_{\text{cad}}$ object models are used, a total of $324N_{\text{cad}}$ images are produced. This is a large sample size, so to make it more manageable as well as comparable to PASCAL VOC, we randomly sample $N$ images from this dataset, where we set $N$ to match the size of the training/validation set for the class in question in PASCAL VOC.

**custom** Six hand-picked mixture components that are deemed particularly common. Each mixture contains a certain camera pose, along with its $180^\circ$ rotation around the azimuth (note that the usage of this symmetry is applicable to cars and bicycles, but not in general). The selected poses are demonstrated in fig. 4.9. The camera distance is not necessarily the same for each pose. With the six camera poses, each with a $180^\circ$ flip, the dataset has $2N_{\text{cad}}$ images per component and thus $12N_{\text{cad}}$ images in total.

The motivation for the hand-picked clusters is two-fold: (1) Particular camera poses are more common than others in natural images taken by a human photographer. For instance, when a close-up is taken of a bicycle, it is commonly taken from a sideview perspective. There can be other reasons as well, such as the backs of cars are commonly seen when the photographers themselves are riding in a car and taking a picture along the direction of traffic. (2) The more camera angles a single component encompasses, the blurrier the model becomes. This can have a detrimental effect, especially to the LRT model. It is also the reason why the models have been scaled and translated to be overlapping over the wheels, since it creates a sharper model in those important regions.

This also brings us to the point of selecting the number of mixtures $M$ for uniform. Too few and the models for each component not only become too blurry, but the pre-calculated bounding boxes may not match all the instances it is supposed to detect. If we increase $M$, the computational load becomes heavier, as well as samples per class go down. Since the standardization process is not perfect, the risk of having one component that gives high scores more generously increases, which can generate a flood of false positives that score
Figure 4.5: Average precision (AP) on the PASCAL VOC validation set for cars using the uniform training set, varying the number of mixture components, $M$. The tests are run 6 times for each number of mixtures, and means are shown as dots, standard deviation as error bars and the extrema as red markers. Since this takes a long time, it uses an LRT model optimized for speed of testing, which is why the results are worse than in fig. 4.6. Higher is better.

higher than true positives for other components. We tune $M$ by running a grid search on the validation set. However, the EM is sensitive to initialization, and as previously mentioned, not all camera poses are equally useful. As a consequence, in each step of the grid search, we test several EM initializations as well as subsets of the uniform dataset. Since this is computationally heavy, we optimized the model for speed by taking the LRT model, using a small descriptor length and ignoring small detections altogether. This makes the results particularly weak, but still valid for relative comparison. Considering the results in fig. 4.5, factoring in the computational costs of setting it too high, we chose to use $M = 12$.

4.4.2 Training data

The classes car and bicycle are both tested. We use $N_{\text{car}} = 23$ models of cars and $N_{\text{bicycle}} = 12$ models of bicycles. Note that this means that each object cluster in the custom dataset is trained from only 46 and 24 images, for car and bicycle respectively.
When training from the *uniform* dataset, there are two sources of variability. First, whether or not the EM that clusters the dataset into six components will hit common poses or not, can be erratic. The EM is restarted 10 times to avoid local optima of too low quality, but this has no notion of good camera poses. Secondly, when compared with *custom*, there is a random choice of what subset of *uniform* to use. This choice will also influence the solution of the EM, so the two sources of variability are closely related. For this reason, the experiments will be repeated several times.

### 4.4.3 Parameters

The most important parameters are set as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>LRT</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD image size</td>
<td>100 x 100</td>
<td></td>
</tr>
<tr>
<td>Subsampling</td>
<td>4 x 4</td>
<td></td>
</tr>
<tr>
<td>Levels per scale space octave, ( \lambda )</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Duplicates, ( D )</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Rotational part spreading radius</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Parts model</td>
<td>polarity/480</td>
<td>plain/480</td>
</tr>
<tr>
<td>Standardization</td>
<td>analytic</td>
<td>none</td>
</tr>
<tr>
<td>Part spreading, ( r_{\text{part}} )</td>
<td>4 (9 x 9)</td>
<td>3 (7 x 7)</td>
</tr>
<tr>
<td>Key point suppress radius, ( r_{\text{kp}} )</td>
<td>3</td>
<td>N/A</td>
</tr>
<tr>
<td>Scale penalty, ( \alpha_{\text{sp}} )</td>
<td>1.0</td>
<td>0.02</td>
</tr>
</tbody>
</table>

These parameters have been decided using a coarse grid search on the PASCAL VOC validation set for cars. The same parameters are used for bicycle without further tuning.

The rotational part spreading is only applicable to the *rotation* model, but has not yielded any improvements during preliminary trials. This may however be specific to the object classes and tasks considered. For instance, the model of a sideview car is largely based on the wheels. The horizontal lines of the car are also important, but not nearly as
Figure 4.6: Results on the PASCAL VOC 2007 test set, trained on custom and uniform. Higher is better.

important as intuition would suggest. Since the wheels themselves are rotation invariant, rotating these parts are largely superfluous, since their rotations will naturally appear in close proximity anyway.

Training the SVM can be trickier when using CAD models, since we run the risk of overfitting. As a result, instead of setting the regularization parameter $C$ using a hold-out set, we fix it to $C = 1.0$. Ideally, the parameter should be tuned carefully by validating it on the PASCAL VOC validation set. However, since we do not want to rely too heavily on the validation set in this treatment, this has not been done automatically; instead, a coarse grid search to determine the fixed $C$ was done on the validation set. Luckily, the parts model has turned out to be fairly robust to the choice of $C$, much more so than when using HOG descriptors.
Figure 4.7: Detections on the PASCAL VOC 2007 testing set. Each negative is marked in red and each positive in green. A hard-decision detector would need to set a threshold value on the $y$ axis, that ideally separates green marks from red. This makes it clear how important it is that a red tails all end at about the same score. The corresponding plots for the SVM experiments are qualitatively similar.

4.4.4 Results

The main results are seen in fig. 4.6. As we can see, the uniform method, despite having more data performs worse than custom. Note that since there is a random element to choosing uniform, these tests were run several times and the result with median AP is displayed. This results in a high experimental cost of running uniform, which is why it was not repeated for the bicycle class.

The problem of uniform seems to stem mainly from standardization issues as fig. 4.7 will show. As a consequence, there is probably room for improvement. In these experiments, the analytic standardization is used. All three standardization methods mentioned in §3.3.2 were tested, but none of them seem to avoid this problem.

The custom model is still subject to the same standardization problems, but is not hurt as badly. Giving a single reason for this is difficult. Our guess is that it is partially because $M$ is lower, but the fact that the model is sharper could also have played a role (custom has on average an entropy of 0.20 bits/feature compared to uniform’s 0.35 bits/feature).

It is unclear how much of the benefits of custom come from cleaner components, how much come from better selected components, and finally how much come from better stan-
standardization. However, most likely all of these contribute to some extent.

A separate test was run with a dataset similar to custom, except the camera pose was for each sample perturbed with Gaussian noise in the azimuth, the altitude, the angle-of-view, as well as the out-of-plane rotation. Despite the intent of promoting invariance, this made the AP worse by 0.5-1.0 points for both LRT and SVM. This suggests that cleaner models are preferred, at least at this sample size.

The SVM with the custom dataset performs well, despite using a fixed $C$ for all components. This is particularly surprising, since samples from different object components are of different size and statistics, so they should require different regularization. Putting each positive sample on top of $D = 10$ background images is for some reason important for the SVM. Without it, the results drop below the LRT model, so there is some stability a greater number of positives is giving.

Even though the SVM is the winner, the LRT model is not falling too short, considering it used a tiny fraction of the negative training data of the SVM. The key points make it possible to standardize the components and compare them. This comparison is still not perfect, neither for LRT nor SVM, but without key points the LRT performs terribly.

The main result as far as CAD-based training is concerned is 26.0% AP and 34.2% AP on cars and bicycles respectively. It is worth noting that the original training/validation sets contain 1583 cars and 408 bicycles, most of which can be flipped to double the sample size. Instead, we used a mere 276 cars and 144 bicycles, which already includes natural flips. This puts our sample size somewhere between a tenth and a fifth of theirs.

On top of that, it uses only CAD-rendered images to train, which in many attempts have performed significantly worse than training from real data [19, 26, 22]. Even Schels et al. [23], who report one of the best results on CAD data, is 15 points short of the DPM [11] trained on real data. Pepik et al. [22] runs the DPM [11] directly on synthetic data, receiving only 24.7% for cars on the PASCAL VOC 2007 test set. This is below our result, despite using almost twice as many CAD models, presumably many more renderings in full RGB
Figure 4.8: Results for the LRT model on the PASCAL VOC 2007 dataset compared to the challenge’s original participants in 2007. Higher is better. (Source: PASCAL VOC 2007 [10])

In the interest of comparing apples to apples, we also choose to compare the results to the original PASCAL VOC 2007 participants, who all pre-date the DPM. This is shown in fig. 4.8.
Figure 4.9: Hand-picked poses demonstrated using one of the car models. The $180^\circ$ rotated pose, which is made to belong to the same cluster, is shown in miniature.

Figure 4.10: Synthetic data used to train the synthetic parts.
CHAPTER 5

CONCLUSION AND FUTURE WORK

The parts model offers a statistically interpretable binary descriptor with competitive expressiveness. It’s main downside is that it suffers from a heavy computational overhead and could use more optimization work. This is not only important in a production setting, but also to speed up the research process by allowing quicker iterations.

Attempts at sorting out high-entropy parts, making the model cleaner, more structured and less correlated between parts in the same spatial location, turned out to show worse results. We accredit this to a discriminative value of high-entropy parts as they imply texture-like properties. A more structured way of doing this would be to try to incorporate this explicitly into the descriptor. If this is done successfully, it is possible that the benefits of having latent polarity and orientation will become more apparent.

The LRT model works surprisingly well for its simplicity, but still falls short of the SVM. One of the main problems with this model is that we must be careful how much the naive independence assumption is false. For this, we need to try to avoid features that have strong correlation patterns; something that is of little concern to the SVM. Solving these problems with the right choice of descriptor, as well as techniques such as using key points, have shown to improve results. Since these problems are far from being solved, the full potential of the LRT is still an open question. However, at some point we might need to incorporate false positives farming, by for instance modeling a richer background model.

As for using CAD models, this work showed promising results considering the sample sizes used. CAD-based training has not taken off in the research community and results are still lagging behind. However, I believe this is not due to irreconcilable differences in the image statistics, but rather the absence of a heterogeneous camera pose prior.

The process of hand-picking camera poses was an experiment to prove this point and not a long-term solution. Eventually, an estimate of the camera pose distribution needs to be made from real data and then integrated into the model, either during training or testing.
There are further problems that can be overcome by training from a mixed dataset of both real and CAD data. However, it is still a valuable endeavor to see how much can be done solely with CAD models, since it exposes the real reasons behind this gap in success rates, which have yet to be fully understood.

## 5.1 Acknowledgement

As a closing note, we would like to acknowledge the Research Computing Center (RCC) at the University of Chicago for lending much appreciated cluster computing time to run many of the experiments in this work.
REFERENCES


APPENDIX A
PERMUTATION EM

A.1 Introduction

Let there be a partition of your data points, so that each block has the same size. Then, define a set of permutations on these blocks and index the permutation with a parameter. This EM algorithm will work for any such set, where the permutation parameter is modeled as a latent variable.

We will denote the permutations as a matrix $A \in \mathbb{N}^{B \times B}$, where $B$ is both the number of permutations and the number of blocks. It is possible to have a non-square permutation matrix, but since we will focus on cyclic groups where this is never the case, we will simplify and always assume $A$ to be square.

A.1.1 Cyclic groups

Even though the algorithm is general to any set of permutations, it is particularly suited for products of cyclic groups. Consider for instance two cyclic groups of image transformations: one group of rotations $(G_{rot}, \circ)$ of size $R$ and one group of polarities $(G_{pol}, \circ)$ of size $P = 2$. The symbol $\circ$ denotes function composition of the transformations. Now form a set from their direct product as $G = G_{rot} \times G_{pol}$ and let all pairs $g = (g_{rot}, g_{pol}) \in G$ also denote the transformation $g = g_{rot} \circ g_{pol}$. Let the transformations be ordered in some fixed way and let $g_b$ denote the $b$:th transformation in $G$. For each image patch $I^{(n)} \in I$ in our raw training data, create a corresponding training set $X$, where each element is

$$X^{(n)} = (\phi(g_1(I^{(n)})), \ldots, \phi(g_B(I^{(n)}))), \quad \forall n$$ (A.1)
where $\phi(I)$ extracts binary edge features from image $I$. Note that $X^{(n)}$ is now partitioned and $X^{(n)}_b$ is how we index block $b$. We define the groups in terms of their generators

$$
(g^{(\text{gen})}_{\text{rot}}(I))(x) = I(\Phi x) \quad \Phi \text{ is a } \frac{360}{R}\text{-degree rotation around the center of } I.
$$

$$
(g^{(\text{gen})}_{\text{pol}}(I) = 1 - I \quad \text{Inverts the intensity.}
$$

For the rotation, since image warping changes the domain of images, we must also consider $I$ a function that takes a pixel location and returns its intensity.

We note that there are only two polarities, while the number of orientations $R$ is a parameter that can be adjusted.

### A.1.2 Permutations

Let $\sigma_N = (0 \ 1 \ \ldots \ N - 1)$ be a cyclic permutation\footnote{This is written in cycle notation and means that $\sigma_N(n) = (n + 1) \mod N$.}. We create the permutation matrix by letting $\sigma_R \times \sigma_P$ generate pairs of indices, where one cycles through the rotations and one through the polarities. Finally, we create a bijection between these pairs into a flattened index space, corresponding to the order in which we did the transformations.

As an example, let $R = 3$ and $P = 2$, and let the flattened index be $i = Rp + r$, so that indices $\{0, 1, 2\}$ are all rotations of one polarity and $\{3, 4, 5\}$ are all rotations of the other.
The matrix becomes

\[
A = \begin{bmatrix}
\sigma_R & \sigma_R + R \\
\sigma_R + R & \sigma_R \\
\end{bmatrix} = \begin{pmatrix}
0 & 1 & 2 & 3 & 4 & 5 \\
1 & 2 & 0 & 4 & 5 & 3 \\
2 & 0 & 1 & 5 & 3 & 4 \\
3 & 4 & 5 & 0 & 1 & 2 \\
4 & 5 & 3 & 1 & 2 & 0 \\
5 & 3 & 4 & 2 & 0 & 1 \\
\end{pmatrix},
\]

(A.2)

where we allowed \( \sigma_R \) to be treated as an \( R \times R \) matrix.

From here on, we will not mention \( R \) and \( P \) anymore since they are abstracted away through \( B \) and the permutation matrix \( A \).

A.1.3 Notation

If \( F \) is the size of a set, \( \mathcal{F} = \{0, \ldots, F - 1\} \) is the set of indices and \( f \in \mathcal{F} \) is an index. Zero-based indexing will be particularly useful for our cyclic groups. I will follow this notation for \( B \) (blocks), \( D \) (edges per block), \( F \) (canonical parts) and \( N \) (samples). The edges \( D \) are both taken over \( L \) spatial locations and \( E \) edge types.

To improve readability, the \((i, j)\) index of the permutation matrix \( A \) will be denoted as \( A(i, j) \), to avoid small fonts.

A.2 Model

Let \( Z_{\text{cmp}} \in \mathcal{F} \) be a latent variable indicating which cluster component or “part” and let \( Z_{\text{per}} \) denote the latent permutation. Then, parts are assumed to be generated from the following
distribution

\[ Z = (Z_{\text{cmp}}, Z_{\text{per}}) \sim \text{Cat}(\pi), \]

\[ X_{b,d} \mid Z \sim \text{Bern}(\mu_{Z_{\text{cmp}}, A(Z_{\text{per}}, b), d}) \quad \forall b, d \]

where \( \mu \in [0, 1]^{F \times B \times D} \) represents our entire parts model and \( \pi \in [0, 1]^{F \times B} \) our joint priors over \((Z_{\text{cmp}}, Z_{\text{per}})\).

As we can see, the permutation parameter \( Z_{\text{per}} \) dictates which block of \( \mu \) a block of \( X \) is generated from. An illustration of this is seen in fig. A.1.

The assumption is that all \( X_{b,d} \) are conditionally independent given \( Z \). When describing the training data, it will become apparent that a certain feature in \( X_b \) will be almost perfectly correlated with its corresponding feature in \( X_{A(b,b')} \), for all \( b' \geq 1 \). As a result, the log likelihoods will be consistently underestimated by a factor \( B \) from this fact alone. However, this will be consistent for all parts, spatial locations and edges, so it should not cause a likelihood imbalance between the parts.

### A.3 Expectation Maximization

Given a training set \( X \) as described in (A.1), the parameters \( \theta = (\pi, \mu) \) are trained using the expectation-maximization (EM) algorithm.

Starting with the E-step:
\[ Q(\theta, \theta^{\text{old}}) = E_{Z|X, \theta^{\text{old}}}[\log \Pr(X, Z; \theta)] \]
\[ = \sum_n E_{Z^{(n)}|X^{(n)}, \theta^{\text{old}}}[\log \Pr(X^{(n)}, Z^{(n)}; \theta)] \]
\[ = \sum_n \sum_Z \Pr(Z|X^{(n)}; \theta^{\text{old}}) \log \Pr(X^{(n)}, Z; \theta) \]
\[ \gamma_{n,Z} \]

Note that the sum over \( Z = (Z_{\text{cmp}}, Z_{\text{per}}) \) is in fact two sums over \( \mathcal{F} \) and \( \mathcal{B} \). The responsibilities, \( \gamma \in [0, 1]^{N \times F \times B} \), are computed as follows

\[ \gamma_{n,Z} = \Pr(Z|X^{(n)}; \theta^{\text{old}}) \]
\[ = \frac{\Pr(X^{(n)}|Z; \mu^{\text{old}}) \Pr(Z; \pi^{\text{old}})}{\sum_{Z'} \Pr(X^{(n)}|Z'; \mu^{\text{old}}) \Pr(Z'; \pi^{\text{old}})} \]

Introducing unnormalized responsibilities \( \gamma' \) and expanding the probabilities gives

\[ \gamma_{n,Z} = \frac{\gamma'_{n,Z}}{\sum_{Z'} \gamma'_{n,Z'}} \]
\[ \gamma'_{n,Z} = \pi^{\text{old}} \prod_{b,d} \Lambda(\mu_{Z_{\text{cmp}},b,d}, X^{(n)}_{A(Z_{\text{per}},b),d}) \]

where, \( \Lambda(\mu, x) = \mu^x (1 - \mu)^{1-x} \). When implementing this, it is better to calculate the log of the unnormalized responsibilities and use the log-sum-exp trick to avoid underflow.

The complete log likelihood is given by

\[ \log \Pr(X, Z; \theta) = \log \Pr(X|Z; \mu) \Pr(Z; \pi) \]
\[ = \log \pi_Z + \sum_{b,d} \log \Lambda(\mu_{Z_{\text{cmp}},b,d}, X_{A(Z_{\text{per}},b),d}) \]
where we note that \( \log \Lambda(\mu, x) = x \log \mu + (1 - x) \log(1 - \mu) \).

The M-step goes as follows

\[
\theta^{(\text{new})} = \arg \max_{\theta} Q(\theta, \theta^{(\text{old})}).
\]

Starting with \( \pi \), this follows the same update as a regular Bernoulli EM. This is given by adding a Lagrange multiplier that ensures that \( \sum_z' \pi_z' = 1 \), deriving and setting the result to zero. We get

\[
\pi_z^{(\text{new})} = \frac{\sum_n \gamma_{n,z}}{N}.
\]

For \( \mu \), we have

\[
\frac{\partial Q(\theta, \theta^{(\text{old})})}{\partial \mu_{f,b,d}} = \sum_{n,b'} \gamma_{n,f,b'} \left[ \frac{X_{A(b,b'),d}}{\mu_{f,b,d}} - \frac{1 - X_{A(b,b'),d}}{1 - \mu_{f,b,d}} \right].
\]

Setting this to zero and rearranging we get

\[
\mu_{f,b,d}^{(\text{new})} = \frac{\sum_n \gamma_{n,f,b'} X_{A(b,b'),d}}{\sum_n \gamma_{n,f,b'}}.
\]

### A.4 Coding parts

Once the parameters \( \mu \) have been determined, we code a certain part over a patch, \( I \), by finding the maximum likelihood part over that patch. If we were true to the model, we would set

\[
Z^{(\text{ML})} = \arg \max_{Z} \Pr(X|Z; \mu),
\]

where \( X \) is arranged as in (A.1). However, this requires the tedious and computationally intensive task of rotating and polarity-flipping the patch \( I \). Instead, we rely on the nearly perfect correlation between the transformed patches and match only the original patch to
Figure A.1: This illustrates how the latent permutation parameter $Z_{\text{per}}$ controls how the blocks of the parameter $\mu$ are paired with the blocks of the data $X$. In this case $R = 3$ and $P = 2$ and the permutations follow the matrix in (A.2).

one of the transformations. This is equivalent of taking the parts model and considering it to be $\mu \in [0, 1]^{FB \times D}$, where $FB$ now is the number of parts. In this formulation, the coding of the parts is identical to the original model without a latent permutation parameter.

Let us remember that the permutation parameter can in our more specific case be broken up further into rotation and polarity as $Z_{\text{per}} = (Z_{\text{rot}}, Z_{\text{pol}})$. To get polarity insensitive features, the $Z_{\text{pol}}^{(\text{ML})}$ part of $Z^{(\text{ML})}$ is simply discarded. In this case, the final descriptor length is $FR$. 