

Semantically driven part-based object detection

Coert van Gemeren

Cognitive Artificial Intelligence, department of Philosophy

Utrecht University

A thesis submitted for the degree of

Master of Science (MSc), 60 ECTS

Utrecht, August 2012



"Now! ... *That* should clear up
a few things around here!"

A thesis submitted by:

Coert van Gemeren (0323217)

coert.vangemeren@phil.uu.nl

1. First supervisor: dr. Robby T. Tan
2. Second supervisor: dr. Zhouyu Fu
3. Reviewer: dr. Stefan van der Stigchel
4. Reviewer: prof. dr. Remco Veltkamp

Abstract

Up to now deformable part based object models have initialized their parts by dividing the shape of their components into equally sized regions. The regions then become the parts that are anchored to a slightly moveable position on top of the components. What a part represents fully depends on what is represented in the region of the component that it was created from. However, the size and the location of the part has always been determined without any regard for the conceptual structure of the component.

We use the framework proposed by Felzenszwalb, et al [1] to learn the shape of objects in ImageNet that are the meronyms of a more complex object from a category of interest provided by the Pascal VOC data set [2]. The meronyms are acquired by analyzing the semantic structure of the given category in WordNet. We then use the shape of the meronyms to estimate their size and location in the learning data of the complex object. After calculating the principal components of the found part configurations we use a Fisher Linear Classifier to cluster the learning data.

Instead of only separating the data based on the shape of its bounding boxes, we create additional model components based on semantic structures. After learning the shape of the components of the model, we place the parts from ImageNet at their estimated positions on top of the semantically enhanced components.

“We can’t define anything precisely. If we attempt to, we get into that paralysis of thought that comes to philosophers. . . .”

Richard Feynman, 1918 - 1988

Acknowledgements

When I first knocked on the office door of Robby Tan there was not much more than a rather vague idea from my part on semantics and object recognition. Robby has helped me tremendously with turning this vague idea into a feasible project. Though he warned me of the complexity of such an endeavour, I am much too stubborn to back away from a challenge like that. Naturally it took a lot more effort than I could have anticipated at that time. I can safely say that if it wasn't for Robby's immense patience and rigor, this would not have come together. Thank you Robby!

Not long after starting the project Robby introduced me to Zhouyu Fu. Robby felt it necessary for me to talk to someone with a lot more mathematical knowledge on my thesis subject than I possessed at the time. Thanks to Skype this was possible, because Zhouyu lives in Australia. Zhouyu has always made the effort to be a part of this project, regardless of the inconvenient time difference between the Netherlands and Australia. He has truly been a great help in upgrading my mathematical skills, as well as for giving me much needed feedback on my thesis. Thank you Zhouyu!

During this project there have been some moments where I have felt that there would be no way in which I could possibly finish this. At that time I have had some great talks with Stefan van der Stigchel. He has really helped me to get back on track. I would like to thank him for supporting me during that time, as well as for being my third advisor.

Furthermore I would like to thank prof. Remco Veltkamp for being a reviewer of this thesis, and I would like to thank René Bloemink for giving me some pointers in the philosophical issues of the introduction.

Op persoonlijk vlak wil ik mijn ouders bedanken voor hun werkelijk oneindig grote steun. Het is niet gemakkelijk die dankbaarheid en waardering op een goede manier te verwoorden, want cliché of niet, ze zouden tekort schieten. Jullie zijn altijd mijn vangnet geweest. Ik had dit niet kunnen bereiken zonder die basis.

Dank.

Ook wil ik mijn broer en zijn familie bedanken voor hun steun. Het was goed om er af en toe even uit te zijn en dan bijvoorbeeld een dierentuin te bezoeken met twee geweldige nichtjes en een neefje!

And finally, I would like to thank Wietske. You have inspired me. At times you have given me a much needed push in the right direction. You suggested the Computer Vision course to me, that ultimately led to this thesis. I know it has been rough at times, but I've made it thanks to your patience, support and above all thanks to your love!

Contents

List of Figures	ix
Glossary	xi
1 Introduction	1
1.1 Object recognition	1
1.1.1 Feature description	3
1.1.2 A multi-view multi-object detection model	4
1.2 Conceptual modeling	5
1.3 Research questions	7
2 Background knowledge	9
2.1 Histogram of Oriented Gradients	9
2.2 Support Vector Machine	11
2.2.1 Optimizing β using a latent SVM and SGD	13
2.2.2 The relation between SGD and HOG	17
3 Learning a deformable part model	19
3.1 Initializing the model	19
3.1.1 Acquiring positive learning data	19
3.1.2 Acquiring negative learning data	21
3.2 Training the model	21
3.2.1 Latent positive learning	23
3.2.2 Updating negative learning data	23
3.2.3 Learning stages	24
3.2.3.1 Initialization	25
3.2.3.2 Mirroring	26

CONTENTS

3.2.3.3	Merging	26
3.2.3.4	Parts	26
3.3	Object detection	28
3.3.1	Feature pyramid	29
3.3.2	Model convolution	30
3.3.3	Distance transformation	30
4	Semantic parts	33
4.1	Acquiring parts	33
4.1.1	WordNet	34
4.1.2	ImageNet	38
4.2	Example Disambiguation	41
4.2.1	Principal Component Analysis of part locations	42
4.2.2	Fisher Linear Discriminant Analysis of prototype sets	45
4.3	Component elimination	48
4.4	Semantic part locations and size estimation	49
5	Results	51
5.1	Testing environment	51
5.2	Influences on model performance	53
5.2.1	Number of components	53
5.2.2	Component grid size	54
5.2.3	Semantic parts	54
5.2.4	Component elimination	55
5.3	Model visualizations	56
6	Discussion	63
6.1	Research questions	63
6.1.1	Model creation	63
6.1.2	Model performance	64
6.2	Suggestions for further research	66
	References	69

List of Figures

1.1	A HOG representation	3
2.1	HOG explanation	9
3.1	Bounding box examples	20
3.2	False positives	24
3.3	Training stages	25
3.4	Greedy part placement	27
3.5	The feature pyramid	29
3.6	Detection convolution	31
4.1	ImageNet: Car wheels	39
4.2	ImageNet: Bicycle wheels	39
4.3	ImageNet: Generic wheels	39
4.4	Root components	40
4.5	Parts of the root components	40
4.6	Prototype examples	41
4.7	Parts heat map	43
4.8	PCA explanation	44
4.9	FLC model comparison	47
4.10	Elimination by kurtosis values	49
4.11	Part location & size estimation	50
5.1	Precision / Recall graphs car category	53
5.2	Precision / Recall graphs bicycle category	54
5.3	Car	56
5.4	Car	57

LIST OF FIGURES

5.5	Car	57
5.6	Car	58
5.7	Car	59
5.8	Car	59
5.9	Bicycle	60
5.10	Bicycle	61
5.11	Bicycle	62

Glossary

Acronyms

BoW	Bag of Words
CTM	Computational Theory of Mind
FLC	Fisher Linear Classifier
HOG	Histogram of Oriented Gradients
LDA	Linear Discriminant Analysis
PCA	Principal Component Analysis
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
VOC	(Pascal) Visual Object Classes

Computer vision terms

Aspect ratio	The height of a bounding box, divided by its width.
Aspect ratio group	A set of images with roughly the same aspect ratio.
Bounding box	A rectangle around the outer edges of an object of interest.
Feature	A characteristic of a visual object representation.
Feature detection	A process, such as HOG, that encodes image features.

Feature pyramid A layered structure representing the features of an image at different sizes.

Granularity The pixel width and height of an image patch in one HOG cell.

Object recognition The ability to recognize an object by means of modeling its visual characteristics.

Prototype A visual representation of an object from a view point that is dominant within an aspect ratio group.

View point A visual representation of an object from some specific perspective.

Visual Perception The ability to interpret information by means of processing visible light.

Linguistic terms

Hypernym A noun which is a supertype of another noun. Eg.: *vehicle* is a hypernym of *coach*.

Hyponym A noun which is a subtype of another noun. Eg.: *woman* is a hyponym of *human*.

Meronym A noun which is a part of another noun. Eg.: *finger* is a meronym of *hand*.

Synonym A word with the same meaning as another word. Eg.: *coach* is a synonym for *bus*.

Philosophical terms

Computational Theory of Mind A philosophical theory that states that cognitive processes are in fact computational processes.

Image understanding The process of interpreting regions and objects to figure out what is happening in an image.

GLOSSARY

1

Introduction

One of the goals of Cognitive Artificial Intelligence (CAI) is to use our understanding of intelligence as discovered by cognitive science, linguistics and philosophy to help improve the modeling techniques used in computer science to create intelligent systems. A major challenge in computer science, and specifically in artificial intelligence, is to create systems that are capable of visual perception. Visual perception is best seen as the ability to interpret information by means of processing visible light. When cognitive science emerged as a subfield of functional psychology in the late 1970's, one of the first topics that was studied using a computational approach was visual perception [3]. Ever since, visual perception has proven itself to be very complex, both from the point of view of cognitive science as well as computer science.

1.1 Object recognition

One aspect of visual perception that is studied in Computer Vision¹ deals with the recognition of objects in images or videos. This task has become commonly known as *object recognition*. Part of the difficulty of object recognition lies in providing ways to deal with the enormous amount of variation that naturally occurs in the data that has to be learned to perform the

¹Computer Vision is the sub field of Computer Science that deals with object recognition as well as other methods for acquiring, processing, analyzing and understanding images.

1. INTRODUCTION

task. To achieve good recognition, a computer model has to be capable of ignoring noise, while learning those visual characteristics that best explain what is seen in the learning data.

Before we get into the subject of this thesis it is important to understand the difference between the major approaches to object recognition. While all these tasks show overlap in the way they work, they differ in what their goal is:

- In an *identification task* we seek to identify a specific instance of an example which has been seen in training, under varying circumstances.¹ One of the types of identification tasks that is best understood is face recognition: Given a certain amount of training data in the form of images of a person’s face, the task is to determine whether or not the face on a queried image is an instance of the person for whom the recognition model was created.
- A *classification task* slightly differs. Here, contrary to an identification task, the particular instance of the object in the queried image has not been seen during training. An example of this type of task is bird classification: given an image of a bird, classify its generic name² from a number of trained classes.
- The *detection task* is the most challenging of the three, because it seeks to determine whether *or not* an object from a certain category is present. This entails that a framework that is successful at this task, can also estimate the size and position of the object in the query image, if present.

Given the last definition, we can see that object detection is the type of task that is most analogous to visual perception in humans. Furthermore, an “ideal” object detector would also be capable of doing the first two tasks and it has been argued by Shapiro [4] that this task is AI-complete³. We are still a long way from an “ideal” object detector, but the innovations in Computer Vision discussed here do make it feasible to get closer to what Shapiro describes as *image understanding*, which is a key aspect of visual perception in humans.

¹differences in lighting, varying angles, etc.

²pigeon, sparrow, blackbird, etc.

³The difficulty of AI-complete computational problems is equivalent to solving the central artificial intelligence problem – making computers as intelligent as people.

The learning data used to train an object detection model is usually presented in a very unrestricted way. The Pascal Visual Object Challenge (VOC) provides such a set of learning data [2]. The data set covers some 20 categories of objects¹, each containing several hundreds of example images. In this research the only meta data that is being used to learn the visual characteristics of those categories is the *bounding box* of each image and its category label.

1.1.1 Feature description

In computer vision a category usually refers to a set of images with a common label, such as ‘car’, ‘cat’ or ‘boat’. In order to capture the visual essence of a given category a method has to be devised that encodes the images in such a way that it is possible to find the common visual characteristics. In computer vision this process is referred to as *feature description*. Feature description can focus on different types of visual features such as *colour*, *luminosity*, *contrast* or *shape*, but it is always a method that tries to digitize the learning data in such a way that it becomes feasible to find those shared characteristics that will improve detection.

There are two main approaches to building models for object detection using feature descriptors. Both have their own advantages and disadvantages, and both have been shown to be successful in their own right [5, 6]. The first one is the Bag-of-Words (BoW) approach. This method is usually combined with some type of local feature descriptor (such as Canny edges [7] or SIFT [8]). The BoW takes its origin in the analysis of documents and text. The visual features that are described in a BoW are like the words in a text: each one of them is a unit of information, contributing more or less to the cumulative meaning that emerges from

¹The VOC data set contains images with the labels: *aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train* and *tomonitor*

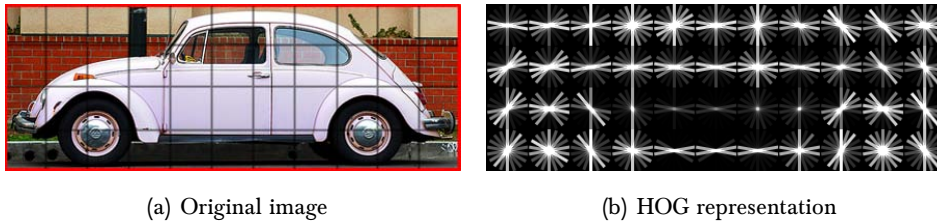


Figure 1.1: The grid on the right shows the two dimensional visual representation of the left image, as seen by the HOG algorithm. Such visual representations are generally not possible in a BoW type of approach.

1. INTRODUCTION

the whole. But, as the name suggests, there is no structure *in between* these units of meaning, other than the amount of meaning that they contribute to the whole.

The BoW approach is contrasted with an approach that does allow for meaningful structural representation. An example of such an approach is a Histogram-of-Oriented-Gradients (HOG). Though HOG is similar in approach to SIFT, in that it represents the gradient information of an image, an essential difference is that contrary to SIFT, a HOG cell is not normalized on its most dominant gradient angle. This means that a local SIFT feature can be used to detect the patch it represents regardless of its rotation. With HOG this is not possible because it cannot compare two patches that may be rotated with respect to each other. Because of this aspect of HOG it is not used in combination with a BoW, but instead with a static grid representation. For an example see figure 1.1. So, for feature description based on HOG a disadvantage is that it cannot compare rotated instances of the patches. The static grid HOG uses also makes it very hard to represent objects with deformable parts¹. Felzenszwalb, et al. counter this by creating a deformable parts model, which makes the grid less static. We will discuss the deformable parts model extensively in chapter 3. The static nature of the HOG grid is also one of its main advantages. Because of it, a trained HOG model has meaningful structural representations that can be visualized, contrary to approaches based on a BoW.

In this work we will make use of this advantage of meaningful representations for HOG grids, as is described in section 4.3. In section 2.1 we will explain in detail how a HOG feature descriptor is created.

1.1.2 A multi-view multi-object detection model

Focusing on recognizing categories of objects poses problems caused by the discrepancy between the actual three dimensional shape of objects representing the same category, and the pictorial two dimensional representations of the learning data in which instances of the category are presented. When detecting the presence of an instance of a given category, the model has to take into account that different representations in two dimensions may actually represent the same three dimensional object from different points of view. A model that

¹Most natural object have deformable parts. Eg.: the limbs of animals and humans.

can take this information into account is said to be capable of *multi-view* detection. The most common approach to multi-view detection, is to encode the shape of a multitude of different view points that are found in the learning data. To do this, the learning data has to be clustered into groups of images representing similar view points, to enable the algorithm to learn multiple models for each one of these different view point clusters.

Object detection is not only about detecting the presence of an object in an image or video, it also deals with determining its size and location in the queried image. When an object recognition model is capable of detecting the locations and sizes of any number of objects from the same category, we speak of *multi-object* detection.

To summarize: the object detection framework described in this work is a multi-view multi-object detection model that was first proposed by Felzenszwalb, et al. [1]. Their work is based on a combination of the Star model proposed by [9] and the HOG feature descriptor proposed by [5]. An innovation in the work by Felzenszwalb, et al. is that they have created a HOG model with two layers. On top of a HOG grid several smaller HOG grids are placed, representing parts of the bottom layer at a higher detail. Also, the parts can drift a certain amount from the position they are anchored to on the bottom layer. The complete framework will be described in detail in chapter 3.

1.2 Conceptual modeling

As described earlier one of the goals of artificial intelligence is *image understanding*. The first step in image understanding is object detection. We have explained that object detection labels regions of images with categorical labels. The label names were decided on creating the group of examples that was the basis for the model that performs the detection. Because of this the labels have become more than meaningless symbolic entities, they have a conceptual quality.

The debate on what exactly constitutes a concept remains unsettled. In philosophy the main debate on concepts is between the *rationalists*, who maintain concepts are innate and can be construed *a priori*, and the *empiricists* who defend the position that concepts are formed through perception and are the primary source of knowledge [10]. Here we will not try to settle this debate. Instead we want to show the relation between the empiricist definition

1. INTRODUCTION

of a concept and the modeling that is done for object detection. We have seen that in the field of computer vision the problem of object detection deals with the appearance of groups of objects, of which there is the assumption that they share one or more visual properties because they share a category label. Following the empiricist paradigm of the definition of a concept, the type of modeling that is being done to create object detection models, is essentially conceptual in nature, when we limit perception to *visual perception*. As is pointed out in [11], the visual aspects capture just one of many aspects of a conceptual description, which seeks to encompass the various kinds of perceptions of objects from the same category. Which can be any kind of combination of tactile, olfactory, auditory or visual aspects.

So, why bother with concepts in a thesis on visual object detection? Well, “Concepts are postulated to explain categorization and comprehension of language.”, Prinz [10]. In that sense it follows that for the empiricist the postulation of a concept is an attempt to bridge the gap between perception and language. The task of visual object detection is in essence also an attempt to bridge a similar kind of gap. Only here it is the gap between *visual appearance* (the learning data) and *formal description* (the model). The way in which visual object detection models are created is by explaining the properties of its learning data in such a way that it can use this explanation as best it can to recognize as many instances of the category as possible, while at the same time avoiding objects that belong to different categories. So, in reference to Prinz, we say that visual object detection models are postulated to explain categorization and comprehension of appearance. They represent the visual aspects¹ of the concept to which they are related through language.

That visual object detection can be seen as a form of concept postulation is helpful for two reasons. First, using theories from conceptual semantics may help to categorize learning data in such a way that it will improve the models created for visual object detection, because in Computer Vision these types of models are always created from specific sets of data. And second, visual object detection models may help to test certain assumptions upheld in theories of conceptual semantics. For instance, the assumption that concepts show prototype effects as posed in [11].

In this work we will focus our attention to the first reason: *Can we use knowledge provided by*

¹or *dimensions* in Gärdenfors’ terminology [11]

conceptual semantics to improve the performance of a visual object detection model?

The conceptual knowledge that we will use in this work is provided by WordNet [12] and ImageNet [13]. How we will use this knowledge is described in detail in chapter 4.

1.3 Research questions

The visual object detection model presented by Felzenszwalb creates several components by clustering the learning data into a given number of groups based on the aspect ratio of each example. We seek to improve this by clustering the examples based on the spatial layout of parts that occur in the examples, which are described in the semantic structure of the object's category on WordNet [12].

The number of parts the model gets and their sizes are chosen beforehand in [1]. All parts have the same size, which is also predefined. The location of the parts is set by a greedy placement algorithm¹ that chooses anchor points based on the most dominant gradients represented on the bottom layer.

Our approach seeks to improve the performance of the models by choosing the components and the sizes and locations of the parts, based on the description of the category the model conceptually represents as given by WordNet. We ask the following questions:

1. Is it possible to structure the learning data of object categories (eg.: cars, bicycles, dogs, cats, etc), based on the visual layout of the objects, using information acquired by analyzing the semantic structure of the category.
2. If so, how does a model with parts placed based on a semantic structure impact the performance, compared to a model where parts were not created based on this structure.

We will provide answers to these questions by first explaining the necessary background information on the techniques used to create a multi-view multi-object detection framework

¹An algorithm that uses a problem solving heuristic of finding the local optimal position for part placement, given the state of a certain configuration of the parts.

1. INTRODUCTION

(chapter 2). After this we will describe the object recognition framework as proposed in [14] in detail (chapter 3). Following this chapter we will explain our approach to disambiguating the learning data to enhance the learning process (chapter 4), followed by our results (chapter 5). Finally, in chapter 6 we will answer the research questions posed here and make suggestions for further research.

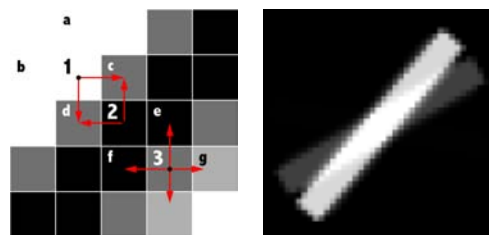
2

Background knowledge

In this chapter we will describe the basic ingredients of the object detection framework. First we will explain the HOG feature descriptor used in this work. After explaining HOG, we will explain how we use a Support Vector Machine (SVM) to create the classifier model used for object detection, based on positive and negative learning examples. After that we explain how Stochastic Gradient Descent (SGD) is used to solve the SVM model.

2.1 Histogram of Oriented Gradients

In order to successfully detect an object we need to have a representation of its visual characteristics. In computational vision this part of the framework is referred to as the *feature descriptor*. The key to the detection of objects with different appearances and poses in the test data, is to generate a model that is able to capture the most dominant characteristics of the object's shape and appearance from all the different view points that may be present in the learning data.



(a) An image patch with a pixel granularity of 5 (b) A visual representation of a HOG cell with two gradients

Figure 2.1: (a) and (b) show how a HOG cell is created. For each of the pixels 1, 2 and 3 four gradient vectors are calculated. The length of the vector is determined by the difference in contrast between the pixels.

2. BACKGROUND KNOWLEDGE

One approach to achieve this goal is to encode the most dominant contrast gradients of the object, given a specific example.

A Histogram of Oriented Gradients (HOG¹) by Triggs and Dalal [5] is such an approach. The basic idea of a HOG-encoding is shown in figure 2.1. The algorithm places the pixels of an image over a grid of cells, creating image patches like figure 2.1(a). The granularity, ie. pixel width and height, of the cells is given beforehand. After that the edge orientation at every pixel under a given cell is estimated by considering the difference in intensity (γ_c) of the pixel's four direct neighbors:

$$\begin{aligned}d_x^c(x, y) &= \gamma_c(x + 1, y) - \gamma_c(x - 1, y) \\d_y^c(x, y) &= \gamma_c(x, y + 1) - \gamma_c(x, y - 1)\end{aligned}\tag{2.1}$$

In a color image the contrast intensity that is used for calculating the angle is the most dominant color channel of $d_x(x, y)$ and $d_y(x, y)$:

$$\begin{aligned}d_x(x, y) &= \arg \max_c d_c(x) \\d_y(x, y) &= \arg \max_c d_c(y)\end{aligned}\tag{2.2}$$

Using $d_x^c(x, y)$ and $d_y^c(x, y)$ we can define $2p$ contrast sensitive gradient orientations, using 360 degree angles, and p contrast insensitive gradient orientations, using 180 degree angles.

If we look at pixel 1 of figure 2.1(a), pixels a , b , c and d are its direct neighbors. Because pixels 1 and a are both white, the length of the vector is 0. The length of a vector in between a black and a white pixel would get the maximum length. This vector represents the difference in intensity between the pixels. As well as being used for the calculation of the orientation bin the pixel is placed in, the vectors are also used as a measure of confidence in that orientation. The vectors in a cell are linearly interpolated among the histograms of the four surrounding pixels. This assures that the edge's orientation is propagated into the body of the object, as there the vectors are all close to zero because there is no edge in between them. We can view this as a kind of smoothing. The reason for doing this is that in the learning data bounding boxes may not be optimally placed. So, when the learning data is used to average each orientation of the grid, interpolation increases the chance that cells of different examples

¹For a glossary of terms see page xi.

actually match each others orientation. This is because a dominant orientation in a cell is spread out to its surrounding cells. If the surrounding cells have weak gradient magnitudes for most their orientations, then the orientation acquired by interpolation becomes dominant, increasing the chance that this orientation is propagated to the model, when other examples have the same dominant orientation in that area of cells.

The magnitude of a gradient vector is calculated by:

$$v = \sqrt{dx^2 + dy^2} \quad (2.3)$$

A further extension to the HOG-algorithm is that the HOG features include histograms for contrast sensitive and contrast insensitive orientation information. This is done by considering the direction of the difference in intensity during its calculation. The orientation information is stored by summing the gradient intensity of every pixel the cell into a histogram of either 9 or 18 orientation bins. For the 9 contrast insensitive bins each bin is the sum of the magnitude at angle it represents in between 0 and 180 degrees and the magnitude of its opposite angle between 180 and 360 degrees. For the contrast sensitive bins each bin has the magnitude of the corresponding angle between 0 and 360 degrees. This orientation histogram constitutes 27 orientation features which together with 4 texture features and 1 truncation feature form 32 features that are encoded for every cell in one component of the model.

In summary, the HOG feature descriptor represents the original image as a grid of cells in which each cell is comprised of an $m \times m$ image patch. The x/y-plane of the HOG cells provide a description of the image at a coarser resolution, while the z-plane provides a description of the magnitudes of gradient angles. In figure 2.1(b) we have shown a HOG cell, in figure 1.1(b) we have shown what the visualization of a full HOG grid looks like.

2.2 Support Vector Machine

A support vector machine (SVM) is described as a tool in computer science to perform classification between two groups of p -dimensional feature vectors [15, 16]. The SVM acts as a linear classifier which creates a hyperplane through the vector space. It attempts to maximize the distance to the hyperplane from those vectors in either class of y_i , that are closest to the

2. BACKGROUND KNOWLEDGE

hyperplane. These vectors are called the support vectors. In that sense a classical support vector machine is a maximum margin classifier.

Formally we have two classes with labels:

$$y_i \in \{+1, -1\} \quad (2.4)$$

And the labeled set of example vectors D :

$$\begin{array}{cccc} \{ & x_1^{(1)}, & \dots, & x_1^{(p)}, & y_1 & \} \\ & \vdots & & \vdots & \vdots & \\ & x_n^{(1)}, & \dots, & x_n^{(p)}, & y_n & \} \end{array} \quad (2.5)$$

We define x_i as:

$$x_i \in \{x_i^{(1)}, \dots, x_i^{(p)}\} \quad (2.6)$$

In the vector space created by D we define hyperplane $H_0 : \beta \cdot x_i + b = 0$, which is the hyperplane that best separates the two classes of vectors. Now we try to find the hyperplanes in the vector space that satisfy the following constraints for the linearly separable cases:

$$H = \begin{cases} \beta \cdot x_i + b \geq 0 & \text{when } y_i = +1 \\ \beta \cdot x_i + b \leq 0 & \text{when } y_i = -1 \end{cases} \quad (2.7)$$

The SVM-algorithm finds two hyperplanes for which the following holds:

$$\begin{aligned} H_1 : \beta \cdot x_i + b &= +1 \\ H_2 : \beta \cdot x_i + b &= -1 \end{aligned} \quad (2.8)$$

The vectors that fall on H_1 and H_2 are called the support vectors. These vectors are the critical elements of the training set as they define the maximal distance that the positive and negative examples in the vector space are apart. The distance between H_0 and H_1 and between H_0 and H_2 is $\frac{|\beta \cdot x + b|}{\|\beta\|}$. Here β is normal to the hyperplane, $\|\beta\|$ is the Euclidian norm of β and $\frac{|b|}{\|\beta\|}$ represents a perpendicular distance from the hyperplane to the origin. The support vectors on H_1 have a perpendicular distance (d_+) to the origin H_0 , which is $\frac{|1-b|}{\|\beta\|}$, similarly the support vectors on H_2 also have a perpendicular distance (d_-) to the

origin H_0 . For the best classification we want to maximize the margin between d_+ and d_- , therefore we should minimize $\|\beta\|$, on the condition that there are no vectors in between H_1 and H_2 . This constraint can be defined as one set of inequalities, which is a combination of equation 2.8:

$$y_i(\beta \cdot x_i + b) \geq 1 \quad \forall i \in \{1 \dots n\} \quad (2.9)$$

The minimization of $\|\beta\|$ can be solved using the Lagrangian multiplier method as the objective function:

$$L = \frac{1}{2}\|\beta\|^2 - \sum \alpha_i [y_i(\beta \cdot x_i + b) - 1] \quad (2.10)$$

In many complex vector spaces however the set of data is not linearly separable. A way to overcome this problem is to introduce the standard hinge loss function as the constraint condition:

$$l(y_i, f_\beta(x_i)) = \max(0, 1 - y_i \cdot f_\beta(x_i)) \quad (2.11)$$

, with:

$$f_\beta(x_i) = \beta \cdot x_i + b \quad (2.12)$$

This turns the objective function for a given training set D , from which we want to obtain a vector of model parameters β , into:

$$L_D(\beta) = \frac{1}{2}\|\beta\|^2 + C \sum \max(0, 1 - y_i \cdot f_\beta(x_i)) \quad (2.13)$$

, where the constant C controls the relative weight of the regularization term.

When a SVM is fully trained it can perform classification of a new vector by calculating the side of the hyperplane, described by β , it falls on. However β itself can be viewed as a model describing those features in D that best discriminate between $y_i = +1$ and $y_i = -1$.

2.2.1 Optimizing β using a latent SVM and SGD

A latent SVM is regular SVM for which not all the modeling information is explicitly part of the training data. Quantifiable implicit (unobservable) modeling information is also called

2. BACKGROUND KNOWLEDGE

a *latent variable*. Latent variables have a well studied history in Hidden Markov Models. However, less is known about their use in discriminative models [17], such as a SVM.

Felzenszwalb et al., use the latent SVM formulation from [18]. In latent SVM each example x gets a score from:

$$f_{\beta}(x_i) = \max_{z \in Z(x)} \beta \cdot \Phi(x, z) \quad (2.14)$$

, where β is some vector of model parameters, z is a particular specification of the object configuration from a set $Z(x)$ of all possible latent values for example x , and $\Phi(x, z)$ is a concatenation of configurations derived from x .

Note that in linear SVM we had equation 2.12, which is the same as equation 2.14, if $|Z(x)| = 1$ (the case with a single possible latent value for each example x).

The latent variables in this work are used to find the optimal configuration of the learning examples that will cause β to perform optimally. This is possible because β is a detection model that can be used on every example x , which results in new learning examples that are derived from the original example. How the latent examples are derived depends on the definition of what can be varied for every example. In this work the following variables are varied:

1. x, y -location of bounding box B on image I . See section 3.2.1 for details.
2. The size of the bounding box B on image I . This is defined by variable l which is a layer of the feature pyramid. See section 3.3.1.
3. The deformation penalty $d_i \cdot \phi_d(dx, dy)$ for the parts of the model, that sit on top of the root components. See section 3.3.3.

By using β as a detector on image I (which corresponds to vector x_i), several new configurations of positive examples are generated, each of which is added as a subexample of x_i to D . One of these is the optimal configuration, given the current state of β . This optimal configuration is found by searching over all latent values of the subexamples of x_i and choosing the one with the highest positive score according to the classifier with state β . Over

several learning iterations of the algorithm, β (the state of the classifier) changes, which in turn causes the best configuration, $\max_{z \in Z(x_i)}$, for positive example x_i to change over time.

Felzenszwalb, et al. have shown that though a latent SVM can lead to a non-convex optimization problem, it can be described as a *semi*-convex problem that becomes convex once the latent information is specified for the positive examples [14].

So, in order to acquire the optimal β a convex optimization problem needs to be solved, once latent information is specified:

- Let Z_p specify one of the latent configurations for a positive example x_i in D
- Define an auxiliary objective function $L_D(\beta, Z_p) = L_{D(Z_p)}(\beta)$, where $D(Z_p)$ is derived from D by restricting the latent values for the positive examples by Z_p .
- Note that for every positive example in D we set:

$$Z(x_i) = \{z_i\} \tag{2.15}$$

, where z_i is the latent value specified for x_i by Z_p .

- So, z_i is a latent value for x_i according to Z_p , which defines the set $Z(x_i)$. We note that:

$$L_D(\beta) = \min_{Z_p} L_D(\beta, Z_p) \tag{2.16}$$

This bounds the ISVM (latent Support Vector Machine) objective because it ensures:

$$L_D(\beta) \leq L_D(\beta, Z_p) \tag{2.17}$$

, which justifies training a ISVM by minimizing $L_D(\beta, Z_p)$.

Now $L_D(\beta, Z_p)$ needs to be minimized. Felzenszwalb, et al. do this in two steps:

1. Relabel the positive examples. Keeping β fixed, optimize $L_D(\beta, Z_p)$ over Z_p by selecting the highest scoring latent value for each positive example.

2. BACKGROUND KNOWLEDGE

2. Optimize β , by solving the SVM. This is the same step as solving the classifier in a standard SVM. As with standard SVM, this can be done by a “coordinate descent” approach, such as quadratic programming or stochastic gradient descent. Felzenszwalb, et al. use the latter:

- Let $z_i(\beta) = \arg \max_{z \in Z(x_i)} \beta \cdot \Phi(x_i, z)$ (*step 1*).

Then $f_\beta(x_i) = \beta \cdot \Phi(x_i, Z_p(\beta))$.

- Compute a sub-gradient of the LSVM objective function as:

$$\nabla L_D(\beta) = \beta + C \sum_{i=1}^n h(\beta, x_i, y_i) \quad (2.18)$$

, with:

$$h(\beta, x_i, y_i) = \begin{cases} 0 & \text{if } y_i f_\beta(x_i) \geq 1 \\ -y_i \Phi(x_i, Z_p(\beta)) & \text{otherwise} \end{cases} \quad (2.19)$$

, where h is the subgradient of the loss function in equation 2.11.

- SGD approximates ∇L_D by using a subset of the examples and taking a step in its negative direction. Felzenszwalb, et al., approximate $\sum_{i=1}^n h(\beta, x_i, y_i)$, for a single example $\langle x_i, y_i \rangle$, with $n \cdot h(\beta, x_i, y_i)$.
- The gradient descent algorithm then repeatedly updates β , as described in algorithm 1.

Algorithm 1 The updating procedure of SGD

```

1: for  $t := 1 \rightarrow num\_descents$  do
2:    $\alpha_t = learning\_rate \cdot t$ 
3:    $i = \text{RAND}(\text{size}(D))$ 
4:    $Z_p(\beta) = \arg \max_{Z_p} \beta \cdot \Phi(x_i, z_i(\beta))$ 
5:    $y_i f_\beta(x_i) = y_i(\beta \cdot \Phi(x_i, z))$ 
6:   if  $y_i f_\beta(x_i) \geq 1$  then
7:      $\beta := \beta - \alpha_t \beta$ 
8:   else
9:      $\beta := \beta - \alpha_t(\beta - C n y_i \Phi(x_i, z))$ 
10:  end if
11: end for

```

We can see that β is shrunk by learning rate α_t ¹, if $y_i f_\beta(x_i)$ is beyond the SVM margin (≥ 1). This means that the example was correctly classified and can be considered an easy example. For the difficult examples ($y_i f_\beta(x_i) < 1$), β is shrunk by α_t and a scalar multiple of $\Phi(x_i, z)$ is added to it.

2.2.2 The relation between SGD and HOG

The HOG description given in section 2.1 represents the shape of an object as a vector of gradient magnitudes. Given an amount of positive examples that share certain shape characteristics, we create a class of feature vectors that represent this data. A negative examples group is then created by generating a certain number of equally large vectors, specifically not representing the objects in the group of positive examples. In sections 3.1.1 and 3.1.2 we will give more details on how the positive and negative example groups are created and updated.

Under the assumption that different instances of the same object class, from the same view point, share the same gradient edge properties at the same x, y -locations, we can use SGD to learn the optimal shape of an object class at a certain view point, described by β . We can view this model as one of the shape prototypes of the object class described by D , as it is comprised of those gradients from D that best describe the general shape of a view point of the class being learned. The idea is to create a model of the most invariant parts of the HOG representations, given in the positive examples. In other words, we are looking for those gradients that are most common at a certain x, y -location over all positive examples, as well as the gradients that are least common in that data. The most common gradients will get positive magnitudes, while the least common ones get negative magnitudes in the HOG-matrix.

In this way we can generate a matrix representation that can be used in a convolution operation with the matrix representation containing one or more potential instances of the object class represented by the model matrix. This convolution results in an energy map of potential locations for the object. The higher (or lower) the value, the more likely the object is present at that location. This method is partly inspired by [19].

¹In practice $\frac{1}{t}$ is used as the learning rate

2. BACKGROUND KNOWLEDGE

3

Learning a deformable part model

Using the multi-view multi-object detection framework to learn a model consists of a number of different stages. Our implementation will follow the implementation of Felzenszwalb, et al. In the next chapter we will explain our changes to the framework in detail, but before we do that it is important to first understand the setup of the original framework.

3.1 Initializing the model

Because we are learning a model with different components for different view points of the same object, we need to separate the learning data as best we can into groups that contain similar view points. As the only annotation used in the examples is the object class and the bounding box location, different views of the object have to be separated from each other in the positive learning data.

3.1.1 Acquiring positive learning data

The first step is to identify a number of different groups. There are simply too many examples in the VOC data set to do this by hand. The only annotation used in the data is a bounding box surrounding the object. The actual object in the bounding box may be presented in an unlimited number of different view points. This effect is illustrated in figure 3.1. To learn the shape of an object from all the different view points at once would be unwise. Instead, a

3. LEARNING A DEFORMABLE PART MODEL

limited number of view points are learned to form a single multi-view model with different components representing the different view points. By choosing an arbitrary number (N) of view points to consider, the learning data is split up into groups of objects that roughly resemble the shape of the objects when seen from those view points.

Splitting the learning data is done by calculating the aspect ratio of each examples' bounding box. The assumption here is that the view point of an object in an image influences the aspect ratio of the bounding box with which it was annotated. However, another aspect needs to be considered: a left sided view of an object will have a bounding box with the same aspect ratio as its right sided counterpart. To bootstrap the learning algorithm into learning different view points of the object, the data should be clustered in such a way that it is more likely that the bounding boxes containing a specific view point of the object will be trained in the same group of our multi-view model. This means that both the aspect ratios, and left or right sided views should be separated from each other before learning can start. Because, if we were to group the bounding boxes containing the different view points of the objects by aspect ratio alone, the problem of learning left and right sided views as the same view would remain.

To create N models encoding the different views we start by separating the positive learning data into N groups. This is done by first sorting the examples on the aspect ratio of the bounding box and then separating the examples into N equally sized groups.

The geometric size of the component model representing each group is then calculated by finding the peak aspect ratio of a smoothed histogram of the aspect ratio values found in each group. From this peak aspect ratio we can calculate a desired HOG grid size $w \times h$ for



Figure 3.1: The images above show three bounding boxes with different aspect ratios from the VOC2011 dataset, resembling three different view points.

the component model. This is done by taking into account the granularity of the HOG cells. By dividing the width and height of the aspect ratio’s target size by HOG granularity m we assure that the target size of a warped image is always an exact multiple of the component size: $(w \cdot m) \times (h \cdot m)$. This method fits the warped image data in each aspect ratio group to the size of the HOG grid for each corresponding component. Doing this also assures that the image data is warped as little as possible, while still getting the object in the bounding box to resemble the most dominant bounding box shape of all the objects in that aspect ratio group.

The next step is to mirror all the example images in each group on their Y -axis. After mirroring, all the images’ HOG features are extracted. Creating a HOG filter F with grid size $w \times h$ for each example. The filters in a given group are then separated into two clusters using a K-means algorithm with the added constraint that if the algorithm places an object into one cluster, its mirrored counterpart should go into the other. This way we end up with two clusters which should contain either predominantly left sided or right sided views of the object in a given aspect ratio group. While the result of clustering the object views in this way is far from perfect, it is good enough to bootstrap N different models that represent a single side of each of the N components. Without clustering, each component model would end up being symmetrical in shape, because for a randomly distributed number of examples there would be about an equal number of examples facing left, as three would be facing right.

3.1.2 Acquiring negative learning data

In order to acquire the necessary negative learning data for each of the N components at initialization, the HOG features of patches of image data of the same size as the component size of the group under consideration, are extracted from random locations in example data that do not contain objects from the class of interest. This way a large amount of negative learning data can be acquired. At this stage the negative learning data basically represents noise.

3.2 Training the model

At the start of every stage the model is used to relabel the bounding boxes B on the positive examples, generating new latent positive examples from $P = \{(I_1, B_1), \dots, (I_n, B_n)\}$.

3. LEARNING A DEFORMABLE PART MODEL

For every data mining iteration the number of false positives is minimized by the `gradient_descent` procedure (see algorithm 2), until either convergence (line 17: $\text{neg_loss}_\delta(\beta) < \Theta$, with Θ as the convergence threshold) or `num_datamine` (line 7: the maximum number of data mining rounds) is reached. After each data mining round $F_p \cup F_n$ is rewritten (lines 15 and 16). During rewriting all the positive vectors are retained, as well as the negative support vectors from F_n , plus half its false positives. After rewriting, the store is grown to `memory_limit` with new false positive examples, found by running the detection procedure on $N = \{J_1, \dots, J_m\}$.

In the pseudo-code of algorithm 2 the `gradient_descent` procedure on line 14 is implemented, which was described in section 2.2.1. The `detect_best` procedure (line 5) is described in the next section, the `detect_all` procedure (line 12) is described in the section after that.

Algorithm 2 The training procedure

```
1:  $F_n := \emptyset$ 
2: for relabel := 1  $\rightarrow$  num_relabel do
3:    $F_p := \emptyset$ 
4:   for  $p := 1 \rightarrow n$  do
5:     Add detect_best( $\beta, I_i, B_i$ ) to  $F_p$ 
6:   end for
7:   for datamine := 1  $\rightarrow$  num_datamine do
8:     for  $j := 1 \rightarrow m$  do
9:       if  $|F_n| \geq \text{memory\_limit}$  then
10:        break
11:      end if
12:      Add detect_all( $\beta, J_j, -(1 + \delta)$ ) to  $F_n$ 
13:    end for
14:     $\beta := \text{gradient\_descent}(F_p \cup F_n)$ 
15:    Retain  $(i, v)$  with  $\beta \cdot v \geq -(1 + \delta)$  in  $F_n$ 
16:    Remove  $\frac{\text{size}(F_n)}{2}$  from  $F_n$ 
17:    if  $\text{neg\_loss}_\delta(\beta) < \Theta$  then
18:      break
19:    end if
20:  end for
21: end for
```

Note that on line 16 we remove half of the vectors from F_n that least resemble the learning data, so that we keep those vectors from F_n that most resemble the learning data, but are not instances of it. This is further elaborated in section 3.2.2.

3.2.1 Latent positive learning

After the initial stage the model is further optimized using the latent learning approach described in section 2.2. The model (β) returned by the learning algorithm is used to generate new positive learning examples by using β as a detector on the example images in the aspect ratio group that β was created from. How detection works is described in section 3.3.

At the learning stage, bounding box B defines the location of a positive example on image I . At $(I, B) \in P$ the detector should fire, so the response score should be high. To achieve this we have seen¹ that β is optimized in such a way that those vectors that best describe the most dominant gradients of P at a certain grid location, are gradually maximized. We will see in section 3.3.1 that the grid location is not just a x, y -location, but also has a layer l in a feature pyramid H .

During learning, the positive examples $x \in D$ are defined for the latent SVM training problem. $Z(x)$ is defined in such a way that it overlaps with B by at least 70%. In this way multiple examples of x , which shift their position with respect to B , are added to the training data, treating the object's root location as a latent variable. The examples are therefore called the *latent positive examples*. It turns out that this is quite helpful to correct the location of badly placed bounding boxes in positive data set P , as is also shown in [20].

3.2.2 Updating negative learning data

After the initial stage, the negative example set is updated by applying β to the negative learning data, using the object detection mechanism. Lowering the detection threshold assures that we will find bounding boxes that, according to β , resemble the object class, but are definitely not objects of interest. These false positives are called the *hard negative examples*. Each data mining round half the number of negative examples is retained, while

¹section 2.2.1

3. LEARNING A DEFORMABLE PART MODEL

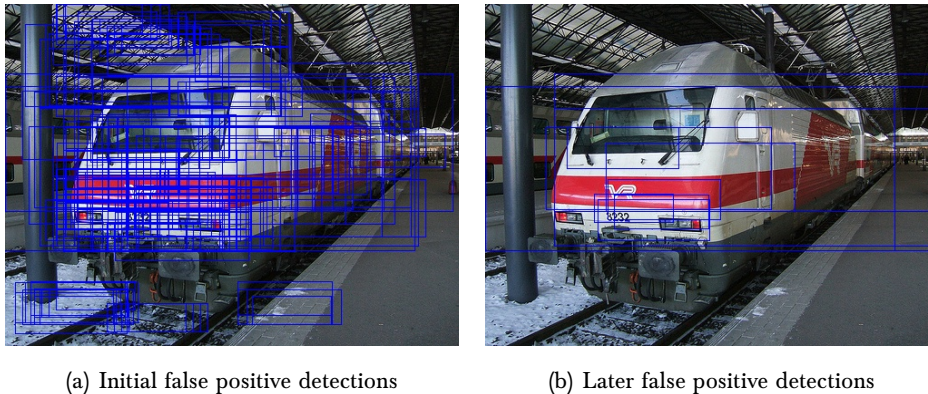


Figure 3.2: The images above show that the detection model learns to avoid false positives better each data mining round.

the negative data set is filled again by finding false positives from a limited selection of all the negative examples available. Half of F_n is retained by sorting the scores $f_\beta(x_i)$ of every $x_i \in F_n$ from high to low. The half with the lower score is removed from F_n .

During learning iterations the number of false positive scores detected in the negative learning data should drop until a convergence threshold is reached. In figure 3.2 we illustrate this effect. On convergence the data mining loop is stopped, and the next round of latent learning can start.

3.2.3 Learning stages

The training algorithm is run several times in four main stages. At every stage the latent learning algorithm generates new latent positive examples and new false positive examples are generated from a generated set of negative learning data. The negative learning data is created from objects from different categories than the one that was queried for. The data mining rounds of the stages are run on a limited sized set of this negative learning data. It is not until the model is fully trained, after the last stage, that the data mining round consists of the full set of negative learning data.

At the first stage, the training algorithm is run once for each of the N component models on a learning set that does not contain mirrored examples. At the second stage, each model is mirrored and the models are trained on a set of learning data containing mirrored examples.

The model should be able to discriminate between the left side and the right side of an object. At the third stage, the models are placed together in one mixture model and it is trained on the original examples set. The model should now be able to decide which of the $2N$ components is represented in each image, by itself. In the last stage the parts are placed on top of the components and the new model is optimized again.

In the following sections each of the subsequent stages of the learning algorithm is explained in more detail, in figure 3.3 a visual overview is given of the state of the model after each stage.

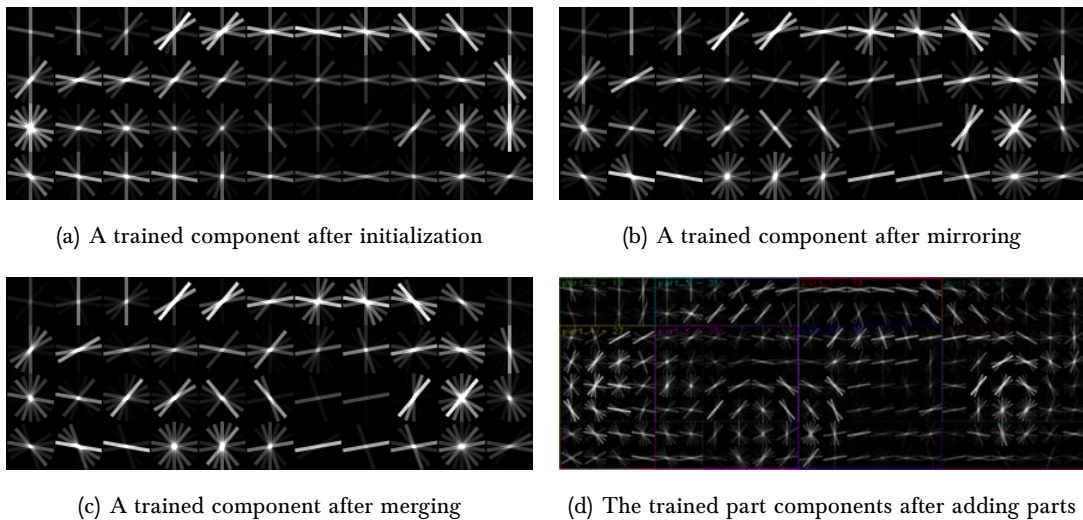


Figure 3.3: The four main stages of training. Each image only shows one component of several different components in one model, with the exception of figure 3.3(d), there all part components on top of a root component are show in their respective positions.

3.2.3.1 Initialization

The main input of the training procedure consists of two data sets. First, there is the feature vector of model weights, which is formed by concatenating the feature values from the current model state β . Initially these are all zero. Second, there is a set of vectors acquired from the learning data, one for every example encountered. Together the vectors form $D = F_p \cup F_n$. One feature vector v_i in $F_p \cup F_n$ contains a header followed by the feature's data blocks ($x_0 \dots x_p$). The header describes the example *label* ($y_i \in \{+1, -1\}$), the example-id (i), layer (l) in the feature pyramid from which the example came, the x -location of the root

3. LEARNING A DEFORMABLE PART MODEL

filter, the y -location of the root filter, the *total* number of data blocks in this example and the *cumulative length* of all the data blocks. Initially the pyramid layer l and the x - and y -locations are all zero, as these do not provide any extra information at the initialization stage. After initialization a component in the model looks like figure 3.3(a).

3.2.3.2 Mirroring

After the initialization stage the oriented root models are copied and mirrored, creating new components representing the opposite view points of the original components. Note that, though each view point has its own model, there are now two components in each of them: the original component and its mirror. After optimization with the positive examples from the mirrored examples groups, the mixture models that are trained in this stage each represent two mirrored perspectives, and are therefore able to detect both an original example and its mirrored equivalent image. The shape of a component after mirroring is shown in figure 3.3(b).

3.2.3.3 Merging

After mirroring every model, we are still stuck with a number of different models. Each one of those models is able to detect a certain view point of an instance of the object class, regardless of its perspective (be it the right side or the left side). These different models can now be merged together into one big mixture model containing all the view points and their mirror perspectives. The mirrored learning data that was separated in different aspect ratio groups can now also be grouped together, as the objective here is to have the new merged mixture model decide which positive example best represents which of the components. This is possible because of the latent nature of the acquisition of positive examples for the training algorithm. The shape of a component after merging is shown in in figure 3.3(c).

3.2.3.4 Parts

After optimizing the merged mixture model, parts are placed on top of each of its components. The number of parts and their size is given beforehand. The default is 8 parts, with size 6×6 HOG cells. As the resolution of the parts on top of the root components is twice as high, each part covers an area of the root component that is 3×3 cells.

The location of the parts is determined by the magnitude of the gradients in each cell on the HOG grid of a component. In a greedy placement algorithm the cumulative gradient energy of each cell in a component is calculated. This creates a heat map in which the locations with the highest cumulative magnitudes, have the highest energy values. Using the heat map, each part is placed, one after another, zeroing out the energy covered by the part on placement. By zeroing out the area covered by the part, a new area on the heat map gets the highest energy level, which determines the location of the next part. This process is illustrated in figure 3.4.

Because the order in which the parts are placed may be disadvantageous with respect to the total area of the root component covered by its parts, the initial placement of the parts is repeated several times in random order to obtain the configuration that leaves the uncovered

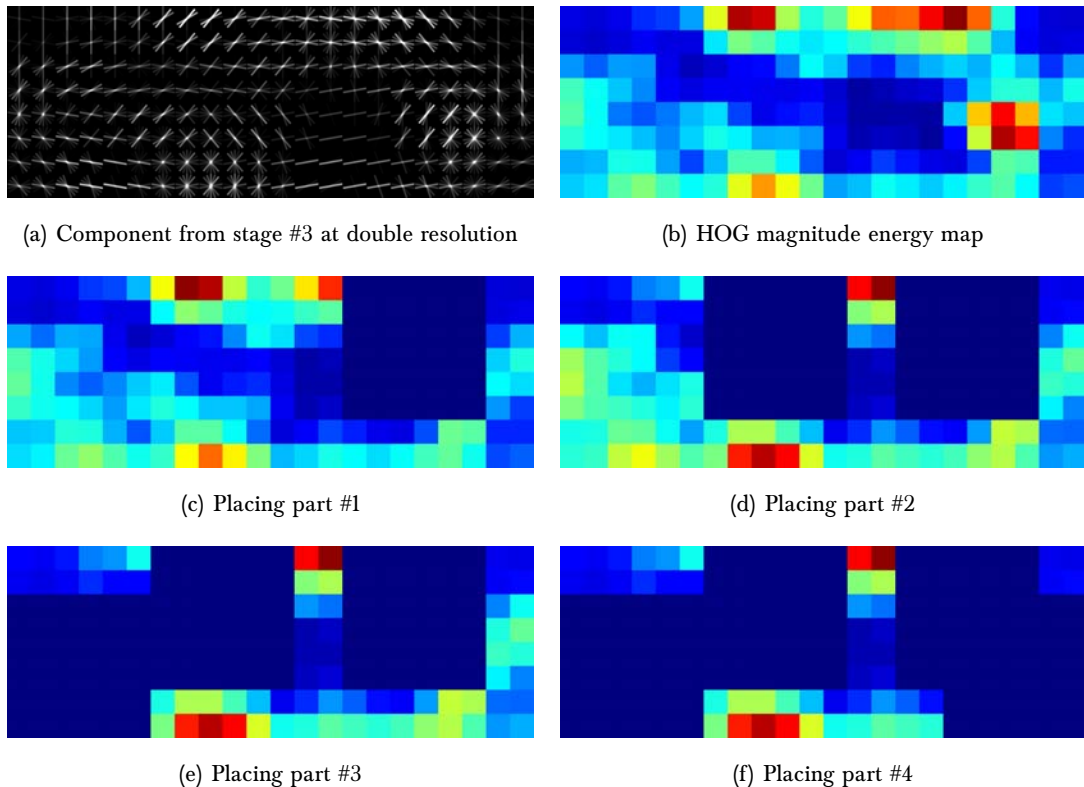


Figure 3.4: Figure 3.4(b) shows a heat map of the sum HOG magnitudes for each cell in figure 3.4(a). Figures 3.4(c), 3.4(d), 3.4(e) and 3.4(f) show the subsequent part placement of the first four parts, based on the energy in the heat map.

3. LEARNING A DEFORMABLE PART MODEL

areas with a cumulative energy level that is as low as possible. In this way the procedure follows a greedy placement heuristic.

The gradient values of the HOG cells in the parts are initialized by the magnitudes of the HOG cells on the bottom layer, after doubling their size. This results in a grid that has twice the resolution of the bottom layer, as can be seen in figure 3.4(a). From this grid we cut out the regions that become the parts.

After training, the part configuration on top of a root component looks like figure 3.3(d).

3.3 Object detection

After training, a model can be used to perform object detection on an image. After the initial stage, a model only contains one specific view point of the object. Therefore that model will not perform very well on just any image, but it will only perform well on an image that resembles the specific view point represented in the aspect ratio group it was trained from. Recall that every mirrored examples group (which has twice the number of examples because it was created by adding a flipped copy of every example) was separated into two oriented groups of data using K -means clustering. Therefore, after the initial learning stage, the model has a specific orientation. This can be explained by the fact that because of the clustering that was done on the examples in the groups of mirrored images, the data in the oriented groups has become skewed to a images of a certain perspective (left or right sided).

The object detection itself is done by calculating the cell activation using the convolution of a feature pyramid H and filters F . The filter collection is the collection of HOG component grids and HOG part grids. The grids of the parts and the components can be placed in the same collection, because aside from their shape they are the same kind of matrix. We convolve each of the model filters with the HOG interpretation of an input image, at each of the layers of the feature pyramid H . This process generates a collection of response matrices R , in which the activation of the HOG cells gives an indication of the location of the part or component of interest (see figure 3.6). The final detection result is the sum of the responses in R . If at a certain location of the sum response a threshold is exceeded, we assume the presence of the object at that location. In this section we will outline the detection process describing the different parts required for object detection in the next subsections.

3.3.1 Feature pyramid

When a detection image is given to the object detector, the size of the object of interest in the image, compared to the size that the model represents the object in, is unknown. This requires a mechanism for the application of the object model to different sizes of the input image. The collection of the feature representations at different sizes of the input image is called a *feature pyramid*. The pyramid is created in such a way that the the bottom level of the pyramid is calculated at twice the image size of the input image, halving the image resolution every λ layers above the bottom level. The number of layers L for a given image of size $w \times h$ is calculated by:

$$L = 1 + \frac{\log\left(\frac{\min(w,h)}{(\lambda/2)^m}\right)}{\log(2^{\frac{1}{\lambda}})} \quad (3.1)$$

, where m is the granularity of a HOG cell and λ is the layer interval size. Figure 3.5 gives a visual representation of the feature pyramid.

We can define a subwindow ϕ of the feature pyramid H as a vector obtained by concatenating

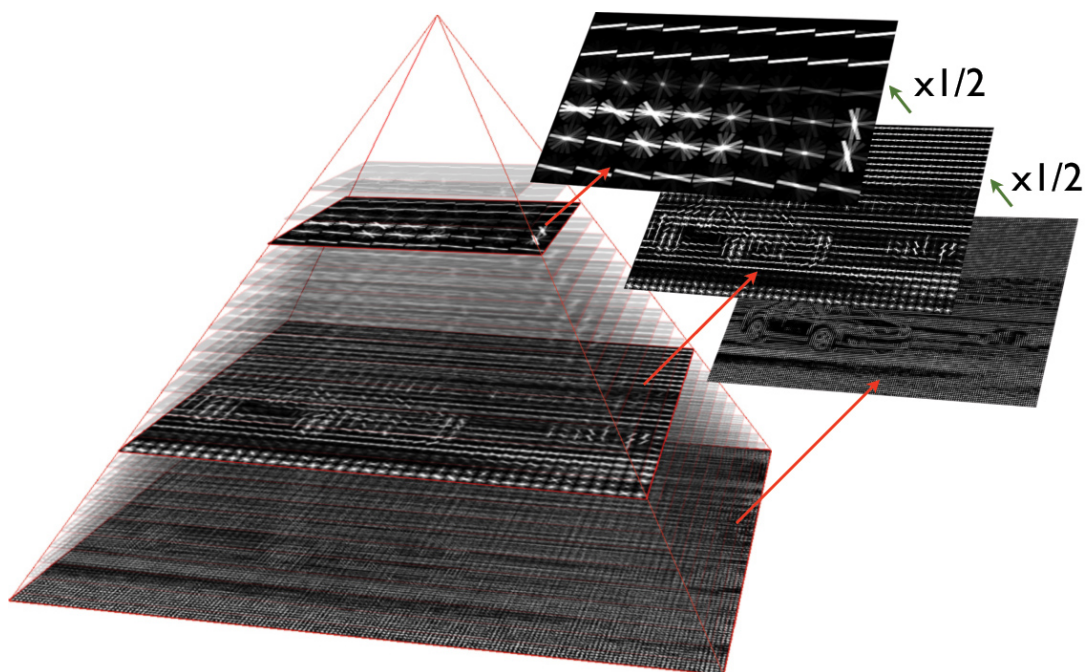


Figure 3.5: A visual representation of a feature pyramid. Note how every 10 layers the resolution of the feature matrix is halved, according to equation 3.1 with $\lambda = 10$.

3. LEARNING A DEFORMABLE PART MODEL

the feature values of level l at position (x, y) with width $w \times h$. The width and height of the subwindow are implicitly defined by the size of F . This is because scoring the locations in $H(l)$ can be viewed as a sliding window approach where we slide filter F over $H(l)$, calculating a score for every possible location of F inside $H(l)$. Therefore we will write the subwindow as $\phi(H, (x, y, l))$.

Recall that in section 2.2.1 we showed that the SGD procedure updates the magnitude for β by a scalar multiple of $\Phi(x_i, z_i)$, for hard examples (examples that had a score within the SVM margin). The latent values $z \in Z(x)$ actually specify an instance $\phi(H, (x, y, l))$ of model M in $H(x)$. Felzenszwalb, et al. then define $\Phi(x_i, z_i) = \psi(H(x), z)$. Now $\beta \cdot \Phi(x, z)$ is exactly the score of the hypothesis z for M on $H(x)$.

3.3.2 Model convolution

To estimate the location of the object, every $\phi(H, (x, y, l))$ should be applied to the different components of the mixture model M acquired after the first stage. This is done by calculating the matrix convolution of layer l and a vector F'_i of the filter features F at part i . As with vector $\phi(H, (x, y, l))$, F'_i is also concatenated in column major order. In this stage, part i is one of the models acquired after bootstrapping, at later stages part i may also refer to parts on the next layer of one of the root components. Response matrix $R_{i,l}$, which is the result of $F'_i \cdot \phi(H, (x, y, l))$, consists of an overall score for every possible filter location in the given pyramid layer. It can be seen as a heat map of the likelihood that a part that should be at v_i is found at a location $(x + dx, y + dy)$ relative to a root location with anchor point (x_0, y_0) in layer l .

3.3.3 Distance transformation

The parts that are anchored to the root layer are also treated as components of the mixture model. During detection, the parts may drift a certain amount from their assigned anchor positions with respect to the root component that they are connected to. This defines the deformable part model. Given the response score for a part we can calculate the spatial uncertainty of its location, taking into account the deformation cost $D_{i,l}$ by calculating:

$$D_{i,l} = \max_{dx, dy} (R_{i,l}(x + dx, y + dy) - d_i \cdot \phi_d(dx, dy)) \quad (3.2)$$

$D_{i,l}$ spreads high filter scores at $R_{i,l}(x + dx, y + dy)$ out over $R_{i,l}$ in such a way that they will contribute more to the overall score, while taking into account the deformation penalty $d_i \cdot \phi_d(dx, dy)$ for displacement of the part with respect to root component v_i :

$$(dx_i, dy_i) = (x_i, y_i) - (2(x_0, y_0) + v_i) \quad (3.3)$$

and,

$$\phi_d(dx, dy) = (dx, dy, dx^2, dy^2) \quad (3.4)$$

At later stages, when the component model is complemented with parts that have twice the resolution of a root filter, the score that a part contributes to the overall score of the component should be taken from a pyramid layer at twice the resolution of the root filter. The number of layers we should go down the pyramid to get to this layer is given by λ . We also need to double the anchor point locations x_0 and y_0 at the part layer to stay at the same location with respect to the anchor at the root layer.

The score for an object location hypothesis in the given image is then given by adding the score of the root filter to the sum of the part filter scores at the given feature pyramid level:

$$score(x, y, l) = R_{0,l}(x, y) + \sum_{i=1}^n D_{i,l-\lambda}(2(x, y) + v_i) + b \quad (3.5)$$

, where b is a value for the bias, which is needed to make the different components that are created from different models comparable when creating the mixture model with all different components.

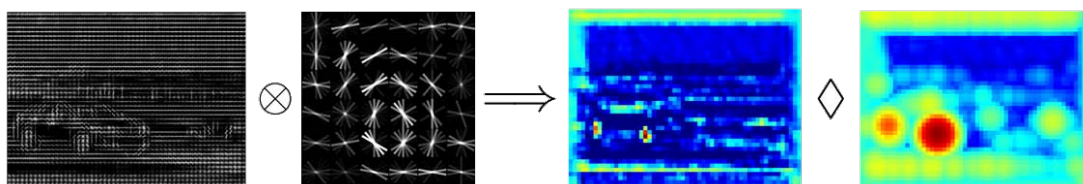


Figure 3.6: From left to right. In detection a layer of the feature pyramid is convolved (denoted by the \otimes -operator) with a filter F , which results in a detection heat map (directly left of the \diamond -operator). The result is then put through distance transformation function 3.2, which results in the final detection result of the given part (directly right of the \diamond -operator).

3. LEARNING A DEFORMABLE PART MODEL

4

Semantic parts

In this chapter we will explain how we get the semantic parts that we will use to create components that represent unique view points of the learning data. We do this in order to be able to split the learning data into groups that represent specific views of the object better than the original implementation does. Next to this, we also use the shape of the parts we learn to find their locations and their sizes on the root components, as they occur in the learning data.

To start we need to acquire and learn the shape of the parts that are found in the semantic structure of the given category. We download the examples of those parts from ImageNet. We then use the object detection framework from chapter 3 to learn the models of the parts. We will use this part model to disambiguate the example data of the given category. At the part creation stage (described in section 3.2.3.4) we will also use the part models to find the regions that they occur in on the components.

4.1 Acquiring parts

In this work we attempt to enhance the object detection framework described in the previous chapter by using semantically inspired parts. We do this because we want to create components based on the shape of the internal structure of the given category, and to be able to place the parts on semantically valid regions of the components. In linguistics a *meronym*

4. SEMANTIC PARTS

denotes a semantic relation which describes that X s are parts of Y (s). In computer vision we usually describe a part as a visually descriptive region inside of the shape of an object. In order to acquire potential part candidates for a detection model, we use WordNet to find suitable meronyms of the given object category. After this step we use ImageNet to find the visual examples of the meronyms we have found using WordNet.

4.1.1 WordNet

WordNet is a lexical database of the English language [12]. Most of the common English nouns are represented in WordNet, as well as a description of the parts associated with the class that the noun represents. For a given class, WordNet returns the corresponding *synset*, from which meronyms of the class can be looked up. Aside from the meronyms, we can also look up the class *synonyms* and *hypernyms* from the synset. A hypernym is a semantic super relation that describes the *is-a* relation known in computer science¹. WordNet also interlinks words based on specific *senses*. A sense is a way to semantically disambiguate classes that have the same word form, though have different meanings, such as: *coach*, which can mean: *manager*, or in a different sense: *passenger car*.

The root of the WordNet database is the *entity* class. From this class all hyponym relations to all nouns known to WordNet branch out. Therefore, using the hypernyms of the given object class we can look up all meronyms that have some relation with it. This includes meronyms that may not be directly related to the object class, but have a relation with it through one of its hypernyms.

For instance, consider the classes of objects that are given in the Pascal Visual Object Challenge [2]. Here we show the lexical tree from WordNet of the classes: *car*², *bicycle*³, *motorcycle*⁴ and *bus*⁵:

car, auto, automobile, machine, motorcar
=> motor vehicle, automotive vehicle

¹In computer science “is-a” describes the relation where one class D is a subclass of another class B

²<http://wordnetweb.princeton.edu/perl/webwn?s=car>

³<http://wordnetweb.princeton.edu/perl/webwn?s=bicycle>

⁴<http://wordnetweb.princeton.edu/perl/webwn?s=motorcycle>

⁵<http://wordnetweb.princeton.edu/perl/webwn?s=bus>

=> self-propelled vehicle
=> wheeled vehicle
=> vehicle
=> conveyance, transport
[...]
=> entity

bicycle, bike, wheel, cycle
=> wheeled vehicle
=> vehicle
=> conveyance, transport
[...]
=> entity

motorcycle, bike
=> motor vehicle, automotive vehicle
=> self-propelled vehicle
=> wheeled vehicle
=> vehicle
=> conveyance, transport
[...]
=> entity

bus, autobus, coach, double-decker, motorbus, motorcoach, passenger vehicle, ...
=> public transport
=> conveyance, transport
[...]
=> entity

We can see that the classes *car*, *bicycle* and *motorcycle* share a relation through *wheeled vehicle*. Though we would expect the class *bus* to also share this relation, the closest relation we can find for this class is the *conveyance* class. However if we look at a different sense of the word *bus*¹, we do find a better suitable relation linking *bus* to *car*:

¹<http://wordnetweb.princeton.edu/perl/webwn?o0=1&o8=1&o1=1&s=bus&zi=4&h=00010000>

4. SEMANTIC PARTS

bus, jalopy, heap

=> car, auto, automobile, machine, motorcar

=> motor vehicle, automotive vehicle

=> self-propelled vehicle

=> wheeled vehicle

=> vehicle

=> conveyance, transport

[...]

=> entity

When we look up the parts for the different classes, WordNet returns the following for *car*¹, *bicycle*² and *motorcycle*³:

car, auto, automobile, machine, motorcar

HAS PART: automobile engine

HAS PART: automobile horn, car horn, motor horn, horn, hooter

HAS PART: bumper

HAS PART: car door

HAS PART: car mirror

HAS PART: car seat

HAS PART: car window

HAS PART: grille, radiator grille

HAS PART: hood, bonnet, cowl, cowling

HAS PART: luggage compartment, automobile trunk, trunk

HAS PART: window

[...]

bicycle, bike, wheel, cycle

HAS PART: bicycle seat, saddle

HAS PART: bicycle wheel

HAS PART: chain

¹<http://wordnetweb.princeton.edu/perl/webwn?o0=1&o8=1&o1=1&s=car&i=2&h=1000000000>

²<http://wordnetweb.princeton.edu/perl/webwn?o0=1&o8=1&o1=1&s=bicycle&i=2&h=100000>

³<http://wordnetweb.princeton.edu/perl/webwn?o0=1&o8=1&o1=1&s=motorcycle&i=2&h=100000>

HAS PART: handlebar
HAS PART: kickstand
HAS PART: pedal, treadle, foot pedal, foot lever
[...]

motorcycle, bike

HAS PART: kickstand
HAS PART: kick starter, kick start
HAS PART: mudguard, splash guard, splash-guard

We can see that a car has a *car wheel* part and the bicycle has a *bicycle wheel*. These classes are both hyponyms of the class *wheel*. The motorcycle itself does not give any kind of *wheel* meronym, however as one of the *hyperyms* of motorcycle is *motor vehicle*, we are still able to find a *car wheel* meronym. If the motorcycle did not have the *motor vehicle* hypernym, we could have reverted to *wheeled vehicle*, in which case the part would have been *wheel* which is a hypernym of *car wheel*.

A selection of hyponyms of the *wheel*¹ class:

wheel
=> bicycle wheel
=> car wheel
=> wagon wheel
[...]

In order to get the best part detection possible we want to learn the meronym which is from a hypernym that is closest to the given object class. We do this because we assume that that particular hypernym will give the most specific visual description, which leads to more accurate part detection. So, for *car* and *bicycle* we will use, respectively, *car wheel* and *bicycle wheel*, both of which are hyponyms of *wheel*.

¹<http://wordnetweb.princeton.edu/perl/webwn?o0=1&o8=1&o1=1&zs=wheel&i=1&h=1000000000000000>

4. SEMANTIC PARTS

Looking at the parts that are returned for a given category, we note that a lot of them do not describe any part of the visual appearance of the outside of the object. For this reason we currently select the class that we want to use by hand.

4.1.2 ImageNet

Closely related to WordNet is the ImageNet initiative [13, 21]. ImageNet is an image database organized according to the WordNet hierarchy. ImageNet currently has an average of 500 visual examples per noun from WordNet. Moreover, some of the image data sets have bounding box information. This is important for our type of learning algorithm because it is shape based. We have seen in chapter 3, that the more accurate the bounding boxes align with the learned objects, the better the performance becomes of the models that emerge from them.

In the previous section we have generated a list of parts from WordNet, given a certain object class. Each part in WordNet is related through a unique ID to a collection of examples in ImageNet, and possibly a collection of bounding boxes that comes with the examples.

We use the object detection framework described in chapter 3 to learn the shape of the parts in the learning data that was acquired from ImageNet. On page 39 we see several random examples from the different synsets downloaded from ImageNet. As it turns out, only the *car wheel* and *bicycle wheel* sets come with associated bounding boxes. Contrary to the images, for the majority of meronyms, bounding box information is not available on ImageNet. For instance, for a car we can find 155 potential part categories, but for only 21 of these there are bounding boxes available, of which only 7 meronyms apply to the visual appearance of the *outside* of a car¹.

For the car category that we learn, we work with the *car wheel* meronym. The final *car wheel* model can be seen in figures 4.4 and 4.5 on page 40. The first figure shows the root components, the second figure shows the corresponding part models. We omit the part deformation overview in this visualization.

¹car wheel, grille, back, windshield wiper, car mirror, automobile horn and car door



Figure 4.1: A selection of the *car wheel* (synset id: [n02974003](#)) from ImageNet

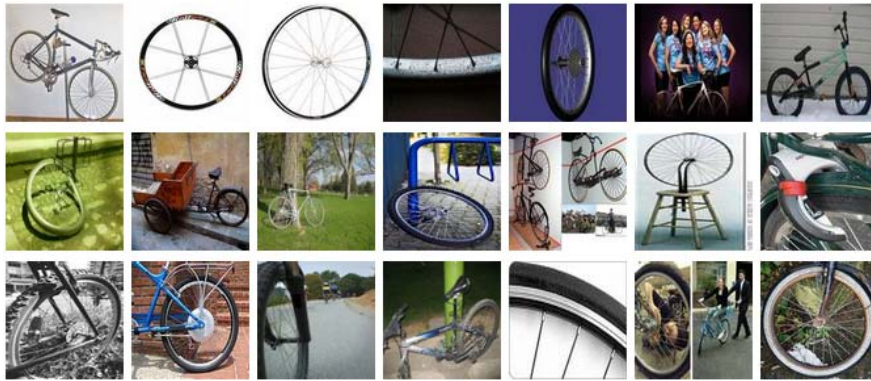


Figure 4.2: A selection of the *bicycle wheel* (synset id: [n02836035](#)) from ImageNet

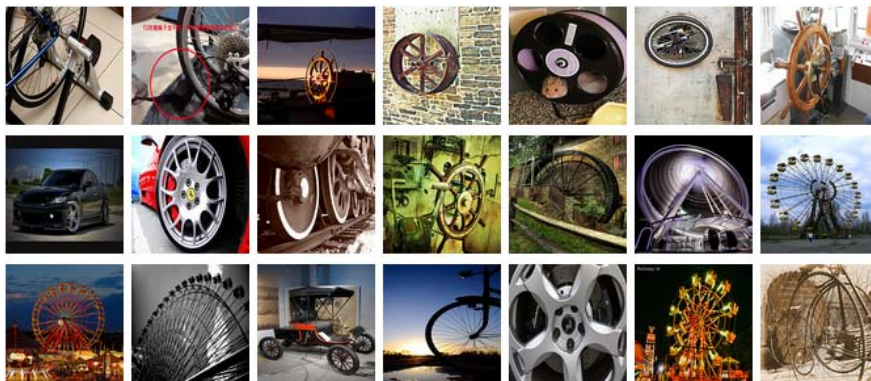


Figure 4.3: A selection of the *wheel* synset (synset id: [n04574999](#)) from ImageNet. Note that this synset encompasses both *car wheel* and *bicycle wheel*, as well as other types of wheels like *farris wheel* and *steering wheel*.

4. SEMANTIC PARTS

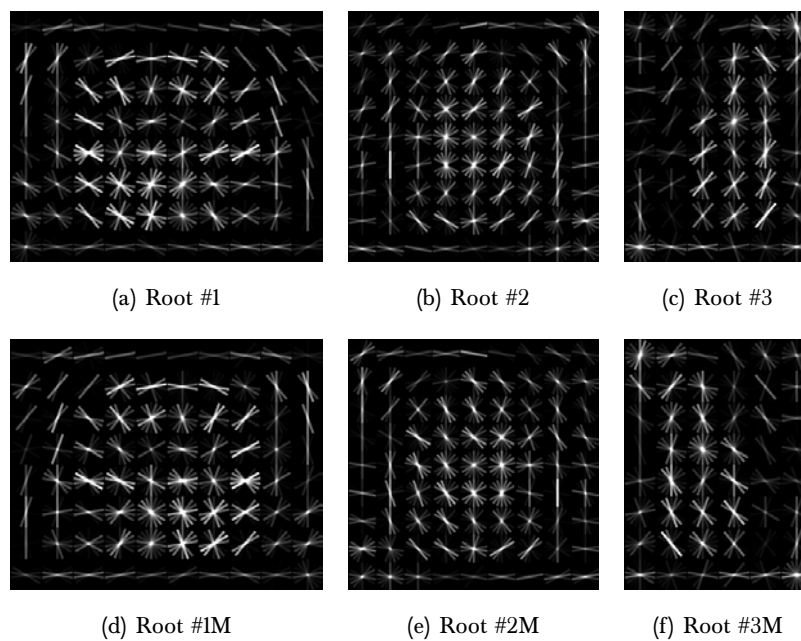


Figure 4.4: The root components created from ImageNet synset n02974003, which together make up a *car wheel* model

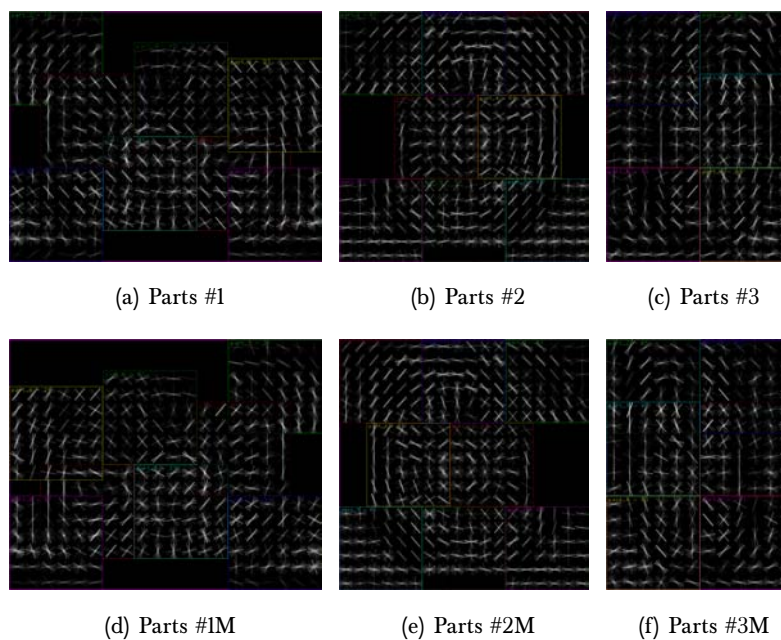


Figure 4.5: The parts that go with the root components in figure 4.4

4.2 Example Disambiguation

In this part we will explain how we disambiguate the learning data in order to create several groups which resemble different view points of the same object class. The internal configuration of the parts in a rigid object can tell us something about the view point the three dimensional object represents on a two dimensional image. For instance, when viewed from the side, most cars show the wheels to be at the lower front and back of the object. However, when the car is rotated towards us, the front of the car becomes visible. As a result the wheels change their 2D-location. This can be seen in figure 4.6. The wheel base becomes smaller and smaller, until the wheels become nearly invisible when the car is viewed from the front.

The aspect ratio of a bounding box drawn around an object may give some information about the view point, given that the object isn't as deep as it is wide. Examples representing the same view point tend to have bounding boxes with similar aspect ratios. Felzenszwalb, et al. use this information to group the learning data into a given number of clustered aspect ratio groups. These N groups are the starting points for learning $2N$ HOG representations of a single object class, as each example is mirrored on its Y-axis to create a components that represent opposite sides. However, using this technique poses problems for objects where rotation on the Y-axis does not alter the aspect ratio of the bounding box much due to the depth of the object. To solve this problem we propose to use PCA on a set of matrices that represent the detection scores of one or more parts that occur in the examples. Because we

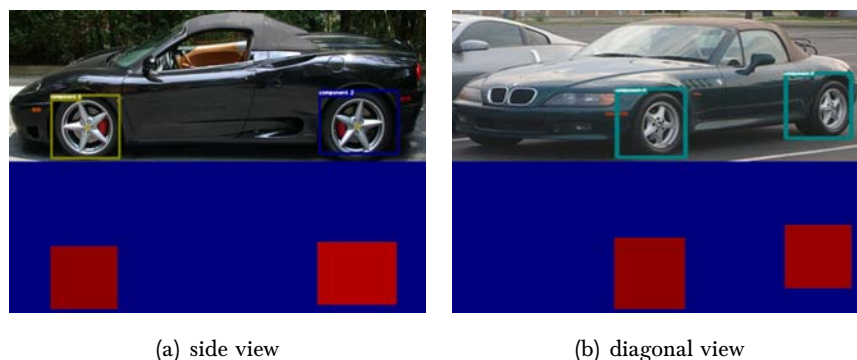


Figure 4.6: (a) and (b) show two examples from different prototype sets, that were in the same aspect ratio group. Note how the placement of the wheel parts differs from one another.

4. SEMANTIC PARTS

have learned the shape of the parts from ImageNet we can look for the location of the parts in the learning data of the given category. For objects with fixed part locations this method may reveal much more detailed information on the view point of the example and the group into which the example should go.

4.2.1 Principal Component Analysis of part locations

Principal Component Analysis (PCA) is a technique used to identify patterns in data. We propose to use PCA because it will allow us to express the learning data in such a way that it will reveal the similarities and differences of part locations within the example images.

Because of the nature of PCA we need vectors which represent the same bit of information on the same location within the vector [22]. Therefore we start the process in the same way as Felzenszwalb, et al: By choosing an arbitrary number (N) of view points to consider, the learning data can be split up into N groups of images with about the same aspect ratio for their bounding boxes, as we explained in section 3.1.1.

We use the object detection framework described in chapter 3 to detect the size, location and confidence score for a given part on an example from one of the aspect ratio groups in the learning data. Using the detection method, we can plot this score on every x,y -location for the part's region on a grid, with the same size as the example image. The result is a detection matrix which shows the most likely x,y -locations for the given part, with respect to the images in the aspect ratio group. If we sum all matrices in the group we get a heat map which shows the most likely part-locations within the given group. In figure 4.7 we show a heat map for a *car wheel*, based on the examples from the aspect ratio group that contains mostly the side views of cars. In figure 4.6 we have seen prototypical examples from this aspect ratio group.

To calculate the principal components for the aspect ratio groups we do the following for each of them:

- Let $X = \{x_1, x_2, \dots, x_n\}$ be a random vector with part observation matrices $x_i \in R_a^c$. Where R_a^c is the group of examples from class c that were warped to the same aspect ratio a .

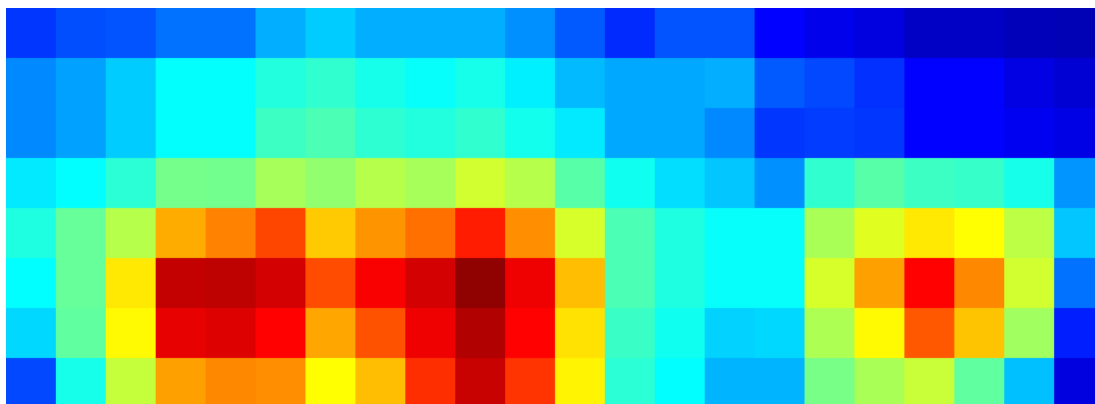


Figure 4.7: Here we see a heat map for the most likely locations for a *car wheel*. We can see that there are three distinguishable locations on which the wheel is most likely to occur.

- Compute the mean μ :

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i \quad (4.1)$$

- Compute the Covariance Matrix S :

$$S = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)(x_i - \mu)^T \quad (4.2)$$

- Compute eigenvalues λ_i and eigenvectors v_i of S :

$$S v_i = \lambda_i v_i \quad i = 1, 2, \dots, n \quad (4.3)$$

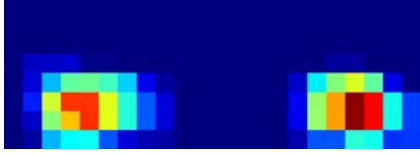
- Compute the sum σ of the eigenvalues λ_i :

$$\sigma = \sum_{i=1}^p \lambda_i \quad (4.4)$$

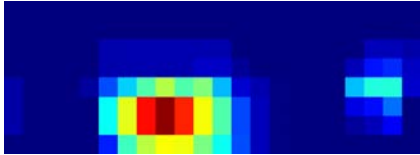
- Order the eigenvectors descending by their eigenvalue. The p principal components are the eigenvectors corresponding to the p largest eigenvalues.
- Using σ we keep k largest eigenvectors while the sum of the variance explained by the eigenvalues is under 95%:

$$\sum_{i=1}^p \frac{\lambda_i}{\sigma} \times 100\% < 95\% \quad (4.5)$$

4. SEMANTIC PARTS



(a) The first principal component of the first aspect ratio group



(b) The second principal component of the first aspect ratio group

Figure 4.8: (a) and (b) show the first two principal components of the first aspect ratio group for the class *car*

The principal components indicate part locations that explain as much of the part placement in the examples as possible. The first principal component explains the largest chunk of the data and each subsequent principal component explains a little less of the data, until we have explained 95% of the part locations in the group. In figure 4.8 we see the first two principal components of the highest scoring wheel part locations in the first group, that contains examples of side views of cars. We immediately notice that these principal components match up nicely with the detections in figure 4.6.

To match the part detection results to the principal components we use template matching with the normalized correlation coefficient:

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'(x', y')^2 \cdot \sum_{x', y'} I'(x + x', y + y')^2}} \quad (4.6)$$

This allows us to assign each example x_i to one of the principal components v_i , which is related to its own component group. We match every example (the first two of which are shown in figure 4.8) or an additional empty component v_{c+1} . The empty component is there to assure that false positives for the given part type will not end up in the wrong component group, because that component happened to be the closest, though wrong, match.

We call the examples that resemble one of the principal components *prototypes*, because the part locations of these examples occur in those locations that explain most of the data. Using this method we generate one set of prototype examples for each principal component G_{v_i} and one spare component set $G_{v_{c+1}}$.

The number of principal components determines the threshold of how many examples need to be in a prototype set for it to be big enough to actually represent a prototype component at

later stages. It makes little sense to keep a component which is only represented by very few examples. We have found that a good rule of thumb is to retain those principal components which represent a fraction equal or larger than the number of examples that would be in every prototype set if every corresponding principal component represented an equal share of examples within the whole aspect ratio group. For instance, if the size of the example group is 100 matrices, which together make up 10 principal components representing 95% of the data, then a principal component would need at least 10 matrices assigned to it by the template matching algorithm, otherwise it will be discarded and its examples are placed in the spare component set $G_{v_{c+1}}$.

During the detection of the parts in the example data set, several examples may not have had any parts detected on them. Therefore, they are not in any of the prototype sets. However, we do need to divide these remaining examples over the newly created prototype groups, as we would like them to contribute to the learning process. Even though these examples may not represent the given part, they may contribute to other aspects of the object class' shape. Because the prototype groups were created from the principal components of the part locations in the data set, we may assume that the data in one prototype set will have some shape characteristics in common in terms of their HOG features. Remember that the HOG features represent the shape of the object and we have been grouping the data by their part locations within the examples of the object class. We can now perform view point specific dimensionality reduction based on the part configurations represented by the HOG features in the prototype groups. We do this using a Fisher Linear Classifier.

4.2.2 Fisher Linear Discriminant Analysis of prototype sets

Given that there are enough examples in a prototype set, we can use the HOG features of the labeled prototypes as learning data for a classifier using Linear Discriminant Analysis, which was first proposed by Fisher [23]. The FLC works by maximizing the ratio of between-class scatter of the sets and the within-class scatter of the features in the examples set. The method we use is analogous to the method used by Bellhumeur, et al.[24] to create glasses recognition and is also described in [25] for gender classification of faces. The FLC can help us decide what to do with the examples that are not in any of the prototype groups because no part was detected on them. As we already have a spare group representing data for which no reliably

4. SEMANTIC PARTS

parts were found, we use this group as a class for the FLC to represent those examples with no parts, next to the prototype classes we have created.

- Let X be a random vector with the HOG feature representations of the prototype examples drawn from c prototype sets and the spare set:

$$\begin{aligned} X &= \{X_1, X_2, \dots, X_c, X_{c+1}\} \\ X_i &= \{x_1, x_2, \dots, x_n\} \end{aligned} \quad (4.7)$$

, where the set X_{c+1} represents the prototype class without any part assignments.

- The scatter matrices S_B and S_W are calculated as:

$$\begin{aligned} S_B &= \sum_{i=1}^{c+1} N_i (\mu_i - \mu)(\mu_i - \mu)^T \\ S_W &= \sum_{i=1}^{c+1} \sum_{x_j \in X_i} (x_j - \mu_i)(x_j - \mu_i)^T \end{aligned} \quad (4.8)$$

, where μ is the total mean:

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i \quad (4.9)$$

and μ_i is the mean of class $i \in \{1, \dots, c, c+1\}$:

$$\mu_i = \frac{1}{|X_i|} \sum_{x_j \in X_i} x_j \quad (4.10)$$

- Next, following Fisher's algorithm, we look for a projection W , that maximizes the class separability criterion:

$$W_{opt} = \arg \max_W \frac{|W^T S_B W|}{|W^T S_W W|} \quad (4.11)$$

As described in [24], a solution for this optimization is given by solving the General Eigenvalue Problem:

$$\begin{aligned} S_B v_i &= \lambda_i S_W v_i \\ S_W^{-1} S_B v_i &= \lambda_i v_i \end{aligned}$$

- Finally, we are left with one other problem. The rank of S_W is at most $N - c + 1$, with N samples and $c + 1$ classes. Even though the number of features in a HOG matrix is lower than the number of pixels in the original image, the number of samples N is still smaller than the length of the HOG vector for one example, so the scatter matrix S_W becomes singular. A detailed description of the problem is given in Raudys et al. [26].

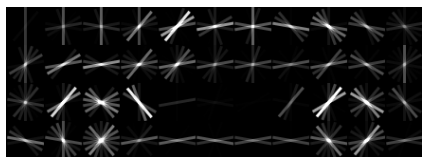
- Belhumeur [24] solved this by performing a PCA on the data projecting the samples into the “ $N - c + 1$ ”-dimensional space. LDA was then performed on the reduced data, because after this step S_W is not singular anymore.

This optimization problem can then be rewritten as:

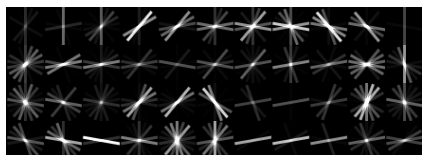
$$\begin{aligned} W_{pca} &= \arg \max_W |W^T S_T W| \\ W_{fld} &= \arg \max_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|} \end{aligned} \quad (4.12)$$

- The transformation matrix that projects a sample into the c -dimensional space is then given by:

$$W = W_{fld}^T W_{pca}^T \quad (4.13)$$



(a) The first root component corresponding to the first principal component of the first aspect ratio group



(b) The second root component corresponding to the second principal component of the first aspect ratio group

Figure 4.9: (a) and (b) show the first two root components of the new model after part disambiguation

By using a FLC we are able to create multiple prototype groups that represent different view points within one aspect ratio group.

After learning the model from the examples provided by the prototype groups created by the FLC procedure we are able to find a new component, which is shown in figure 4.9(a). When the image data was just separated by aspect ratio, only the model generated from the more prominently available image data, resulting in figure 4.9(b), would be found. In the discussion we will see whether the newly created component results in a better performance, if compared to the original implementation.

4.3 Component elimination

Because we create multiple sub components based on the aspect ratio groups as described above, not every one of those groups will represent the shape in its examples as well as the other components. This is due to the fact that for some component groups no coherent HOG configuration can be trained by the learning algorithm, because the shapes in the images in that group are too inconsistent with respect to each other. As we have discussed earlier, this is one of the disadvantages of using a static HOG grid. Luckily, we can turn this disadvantage into an advantage.

We can infer a measure for the quality of a component by looking at the kurtosis¹ of the HOG cells in the grid. Recall that every cell is a histogram on the z -plane of the HOG matrix. Every bin in the histogram of a cell represents an angle between 0 and 360 degrees. A good HOG grid has to meet certain expectations that we want to quantify. One of the expectations is that an outline is visible that represents the edges of the prototypical view point of the component. Because a single HOG cell is actually a histogram of the most dominant angle at a grid location, a cell of a trained model in which every bin has a very low magnitude implies that there is no edge at that location of the model. Alternatively, a cell in which every bin has a high magnitude implies that at that point the gradients in the learning data were not consistent. In other words, every example in the group from which the component was learned, had different information about the gradient at that location. Lastly, a histogram that shows a peak at a certain bin implies that the model has some degree of certainty for the edge orientation that corresponds to the bin in the cell at the location on the component.

If we take the kurtosis of the histogram as a measure of the quality of the gradient at that location, the kurtosis sum of the entire grid is an indication of the representativeness of the component. We can order the components based on this criterion: A component that has a high average kurtosis has a higher quality than a component with a low average kurtosis. Because we have created multiple components per aspect ratio group using the PCA/FLC method, we can eliminate those components that have a significantly lower kurtosis than the best kurtosis found in the group of components. As a rule of thumb we use one standard deviation (σ) from the highest kurtosis sum in the group as the threshold value. Components that have a lower kurtosis than the threshold are eliminated from the model. In figure

¹In statistics the kurtosis is a measure for the “peakedness” of a histogram.

4.10 we show three components created from the same aspect ratio group. The component represented in figure 4.10(c) gets eliminated from the model because its kurtosis sum is too low.

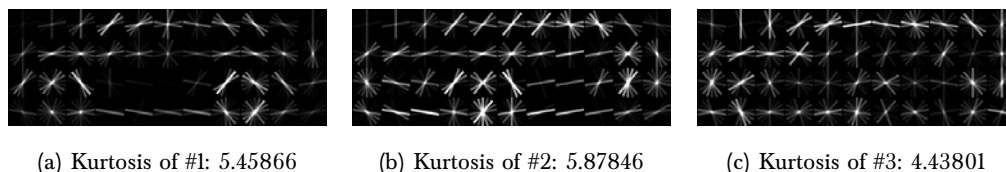


Figure 4.10: The kurtosis sum of the component in figure 4.10(c) is more than σ from the components in figures 4.10(a) and 4.10(b).

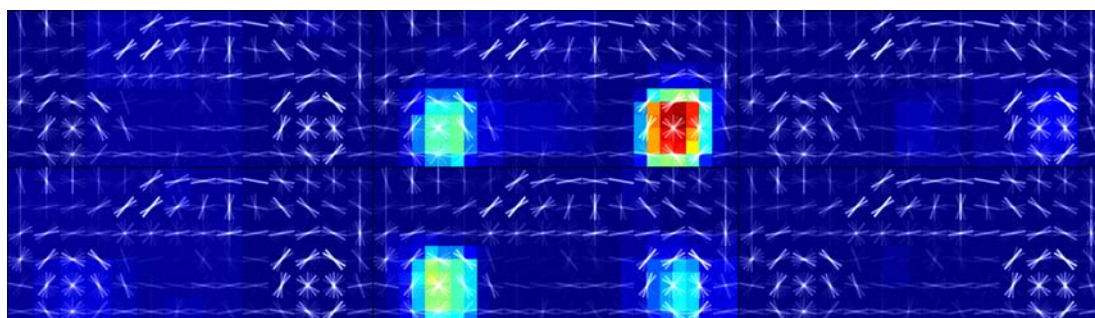
4.4 Semantic part locations and size estimation

After creating and optimizing all components of the model we need to place the parts on top of the root components. As we have models for the semantic parts from ImageNet, we can use this information to estimate the location and size of these parts with respect to the root components.

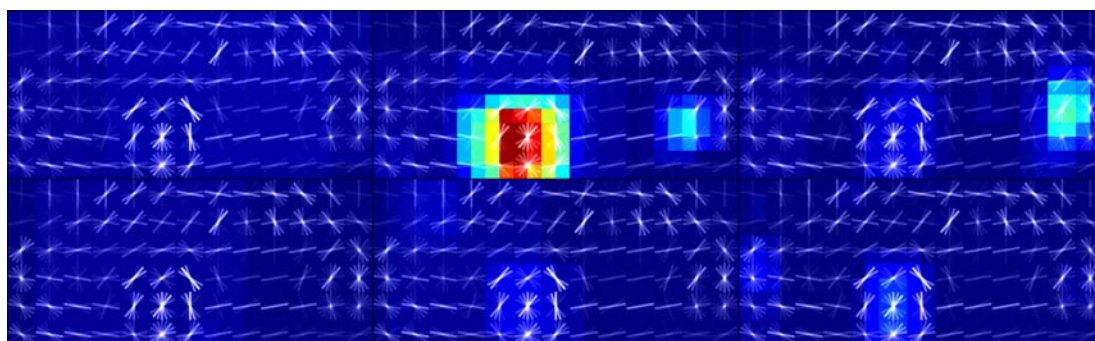
We use the detection mechanism on the examples of each aspect ratio group that remains after component elimination to generate new heat maps with part location estimations. In figure 4.11 we can see that we get very accurate results for the location and size estimation of the wheel on the first three components. The third component is less clear about the location and size of the wheel, this is to be expected as this component tries to model the front/back of the car, from which we know that wheels are hardly visible.

On placing the parts we take a similar approach as described in figure 3.4. Every part that is placed, zeroes out the underlying values of the location estimation heat map, causing the next peak to become the highest location. As long as there are peaks that are over a certain threshold value, parts of the given type will be added to the component.

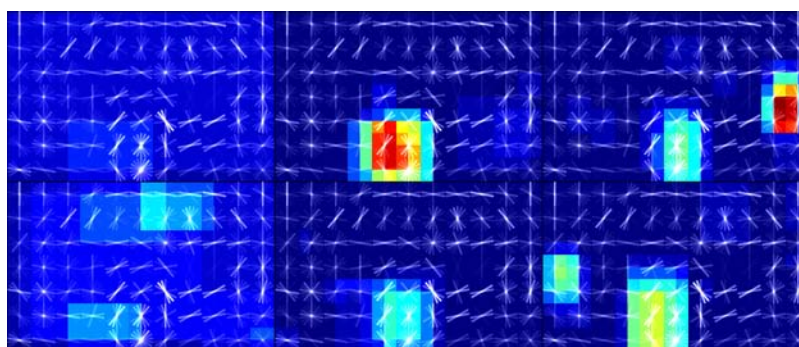
4. SEMANTIC PARTS



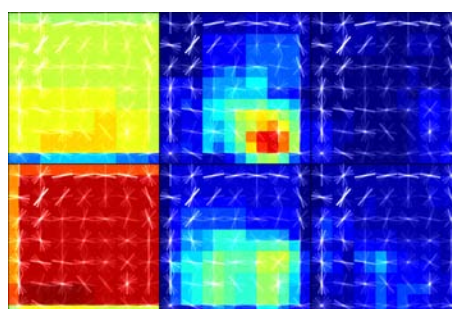
(a) Wheel detections of component #1



(b) Wheel detections of component #2



(c) Wheel detections for component #3



(d) Wheel detections for component #4

Figure 4.11: 4.11(a), 4.11(b), 4.11(c) and 4.11(d) show the detection results of *car wheel* on different root components of the *car* class. The columns show the detection results for each part component from figure 4.4. The rows of each image show the detection results for the original and the mirror components of the *car wheel* from figure 4.4.

5

Results

On the following pages results for different types of setups and object classes are shown and described.

In table 5.1 we show an overview of several different models we have tested using models created with the original implementation and with our changes. Table 5.2 shows a description of the columns used in table 5.1. The model names in the first column of the table that start with Fsz_, refer to the models created using our own framework, but with the specifications of the original implementation of [14], described in chapter 3. We change the number of components and the size of the components in these models to test for differences in performance. The model names starting with Gem_ refer to models that were also created using our own framework, but with the changes described in chapter 4.

5.1 Testing environment

The results of the implementation were generated on a 6 core Intel Core i7 at 3.2GHz with 16GB of RAM memory. Training the original car model (referred to as Fsz_car3c8p in table 5.1) took about 8 hours. Testing the model on the 5823 examples from the VOC2011 validation set (valset) took about 4 hours, which is an average of 2.4 examples per second. Creating CvG_car4c8p took about 16 hours, testing the model took about 8 hours, which is an average testing rate of 1.2 examples per second.

5. RESULTS

Table 5.1: Model Performance Overview

Model	Cat.	Part	Exps.	Cells	Cmps.	Parts	AP-score
Fsz_car3c8p	car	n/a	2026	1860	6	44	0.416
Fsz_car4c8p	car	n/a	2026	2388	8	56	0.401
Fsz_car3c8pxl	car	n/a	2026	2212	6	48	0.364
Gem_car7c8p	car	n02974003 ¹	2026	4676	14	112	0.378
Gem_car4c8p	car	n02974003	2026	2446	8	60	0.398
Gem_car4cwp	car	n02974003	2026	932	8	10	0.375
Fsz_cyc3c8p	bicycle	n/a	706	2182	6	48	0.494
Gem_cyc4c8p	bicycle	n02836035 ²	706	3614	8	64	0.465
Gem_cyc4cwp	bicycle	n02836035	706	1886	8	16	0.470

¹ n02974003 refers to the *car wheel* synset from WordNet

² n02836035 refers to the *bicycle wheel* synset from WordNet

Table 5.2: Descriptions of the columns in table 5.1

Column name	Description
Model	Name of the model.
Cat.	VOC category of the model
Part	ImageNet part used
Exps.	Number of learning examples used to learn the model
Cells	Number of HOG cells in the model
Cmps.	Number of components (incl. mirrors) in the model
Parts	Number of parts on top of all the components
AP-score	Average precision scored on the test set from VOC2011

5.2 Influences on model performance

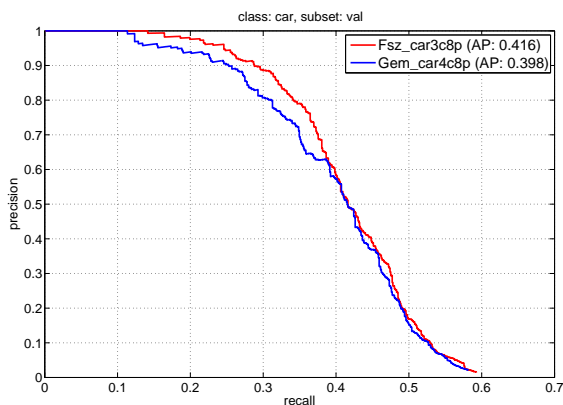
In this section we will present our findings based on comparing the results of the different models from table 5.1 with each other. We will refer the model visualizations presented in section 5.3.

We can tell from figure 5.1(a) that for the car category our best model (Gem_car4c8p) does not outperform Felzenszwalb, et al.'s best performing model (Fsz_car3c8p). We see that the precision of our model falls just below the Fsz_car3c8p model. We also make some other observations though.

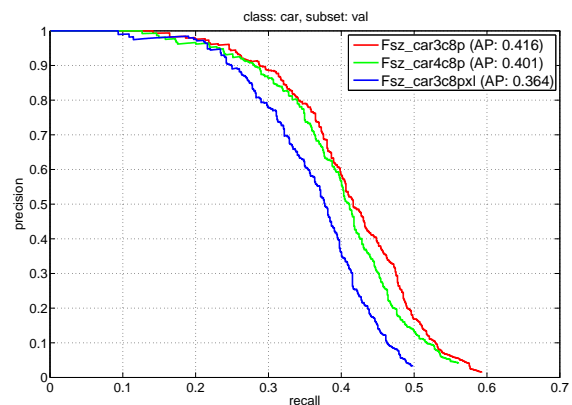
From figure 5.2 we can tell that in the bicycle category we also perform worse than the original implementation by Felzenszwalb, et al. In section 5.2.3 we describe some of the observations with regard to the bicycle models.

5.2.1 Number of components

The number of components influences the performance of the model. Comparing Fsz_car3c8p and Fsz_car4c8p in figure 5.1(b), we see that Fsz_car4c8p scores slightly worse. The only difference between these two models is the number of components that are learned. Fsz_car3c8p has 3 components, compared to the 4 components learned for model Fsz_car4c8p.



(a) PR graph comparing Fsz_car3c8p and Gem_car4c8p



(b) PR graph comparing Fsz models

Figure 5.1: Precision / recall graphs car category

5. RESULTS

The models can be seen in figures 5.3 and 5.4.

5.2.2 Component grid size

We find that another influence on the performance is the size of the components. We can tell from figure 5.1(b) that Fsz_car3c8pxl scores worse on both precision and recall, compared to Fsz_car3c8p. The only difference between the two models is the size of the root components. We can see this in figures 5.3 and 5.5. The size of root component R#1 for model Fsz_car3c8pxl is 11×4 cells, while the size of root component R#1 of model Fsz_car3c8p is 14×6 cells. Besides having to learn more cells because of this increase in size, the given amount of parts does not sufficiently cover the root component of model Fsz_car3c8pxl, which is why we see recall and precision drop in figure 5.1(b).

5.2.3 Semantic parts

From the graph in figure 5.2 we can tell that we do not outperform Fsz_cyc3c8p. However, we do make the observation that Gem_cyc4cwp, which only has bicycle wheels placed as parts, outperforms Gem_cyc4c8p, which has both bicycle wheels and parts that cover the rest of the root components. See the visualizations of the models in figures 5.10 and 5.11. This is noteworthy because in table 5.1 we can see that Gem_cyc4cwp has about half of the number of HOG cells and a quarter of the number of parts, compared to Gem_cyc4c8p.

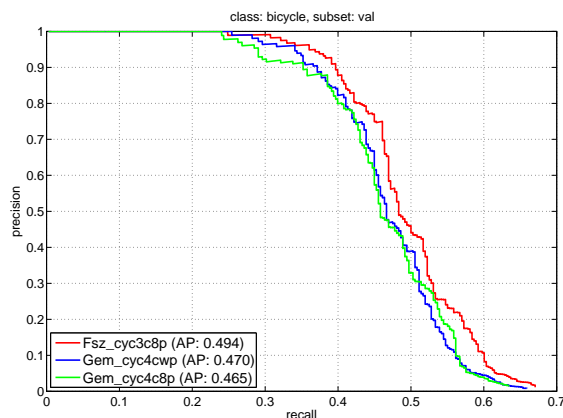


Figure 5.2: Precision / recall graphs bicycle category

5.2.4 Component elimination

By comparing the average precision scores of Gem_car7c8p (0.378) and Gem_car4c8p (0.398), shown in table 5.1, we can tell that eliminating components based on the kurtosis scores of the models makes sense. The only difference between these two models is that before placing the parts on the root components R#2 (figure 5.8(b)), R#4 (figure 5.6(d)) and R#5 (figure 5.6(e)) were removed from Gem_car4c8p, based on the kurtosis sum score of these root components.

5. RESULTS

5.3 Model visualizations

In this section we will show the visualizations of a selection of models from the overview in table 5.1.

The visualizations are structured in three sections. The first figures, referred to with “R# n ”, show the root components of the model. The figures referred to with “P# n ” show the parts layer connected to each root component. The number after the letter connects the parts with the component. The figures referred to with “D# n ” show the deformation penalty for each part. The deformation visualization shows how much a part may drift from the anchor point at which it is represented here. The penalty on the final detection score gets higher as the color in the deformation grid gets brighter (see section 3.3.3).

The parts that were created using the semantic part creation methods described in chapter 4 are visualized in green.

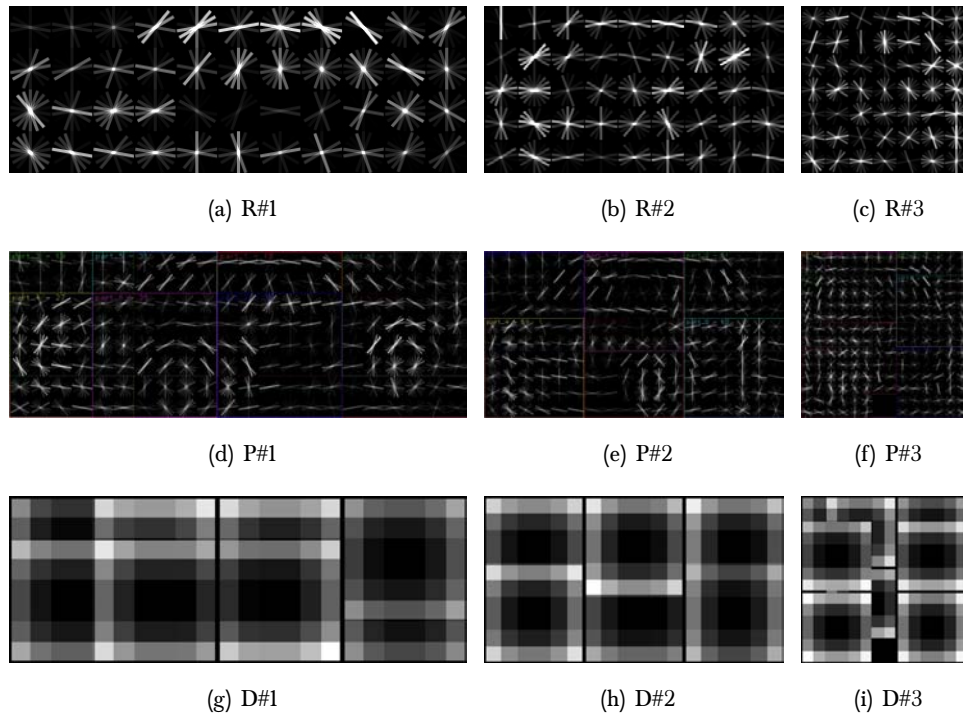


Figure 5.3: Fsz_car3c8p: Original configuration for class “car”

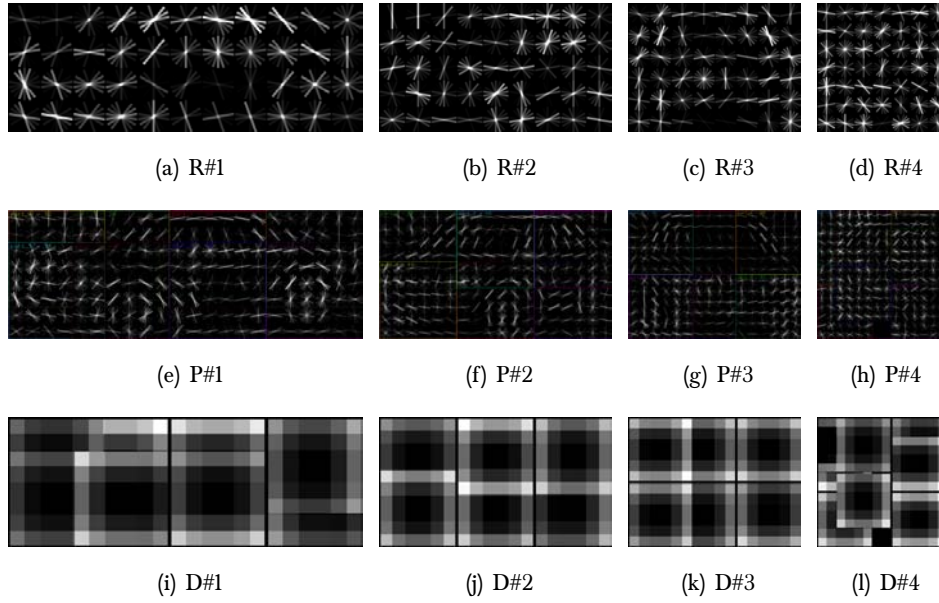


Figure 5.4: Fsz_car4c8p: A 4 component model using the original configuration.

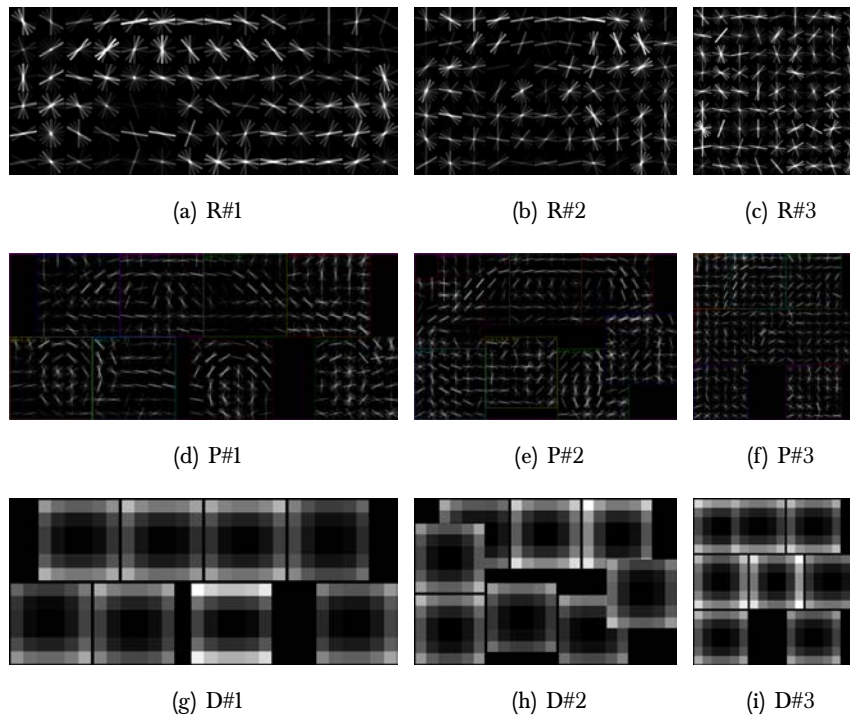


Figure 5.5: Fsz_car3c8pxl: A 3 component model with a 16×4 root component for R#1

5. RESULTS

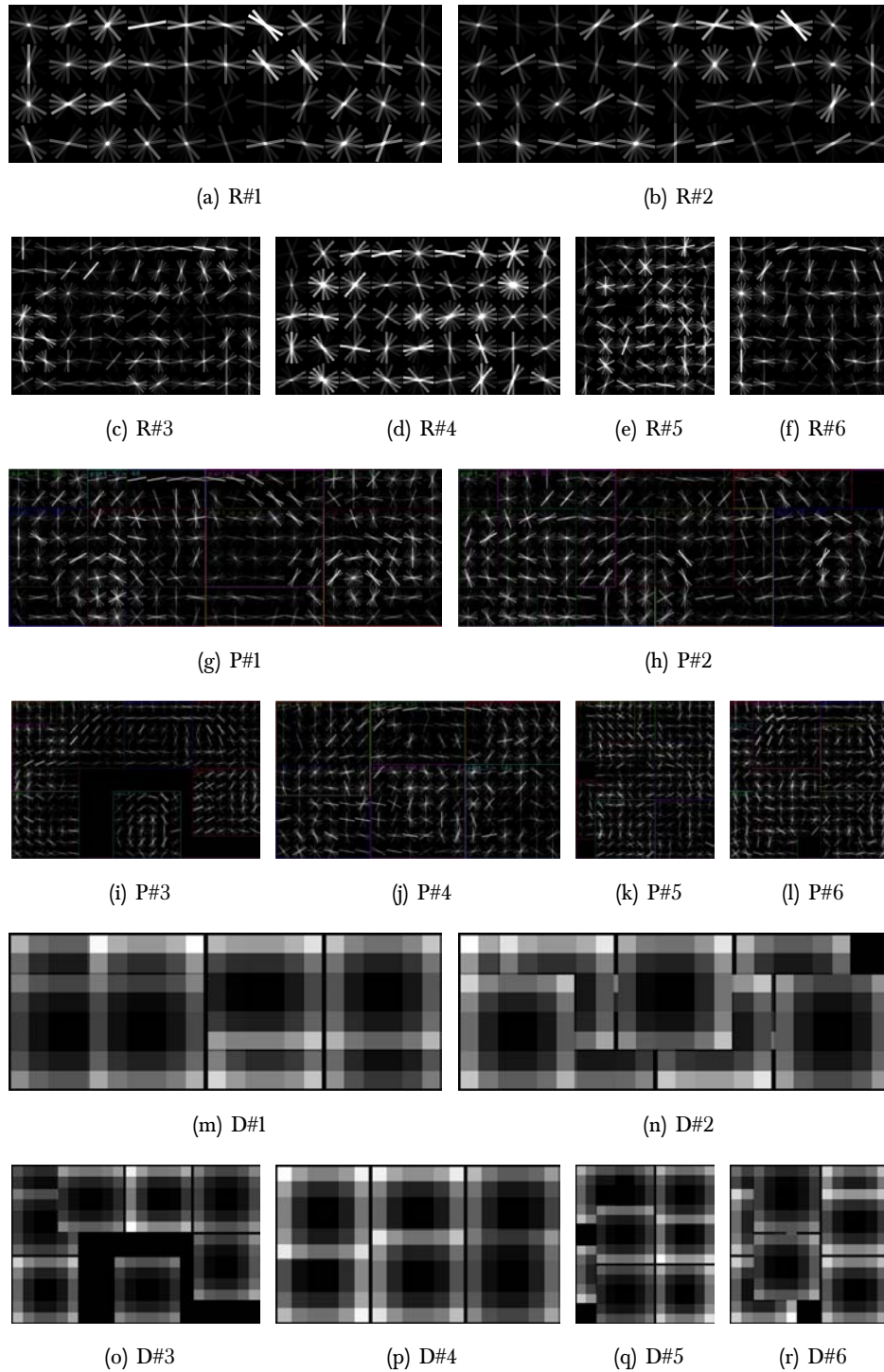


Figure 5.6: Gem_car7c8p: PCA/FLC component disambiguation, no elimination of bad components, no semantic parts

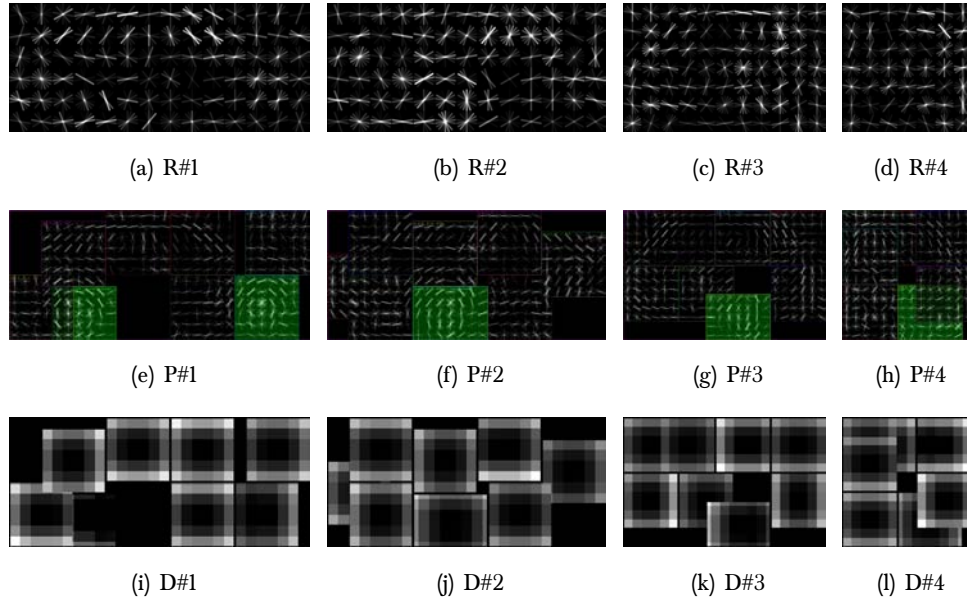


Figure 5.7: Gem_car4c8p: PCA/FLC component disambiguation, elimination bad components. Semantic parts, and fill parts.

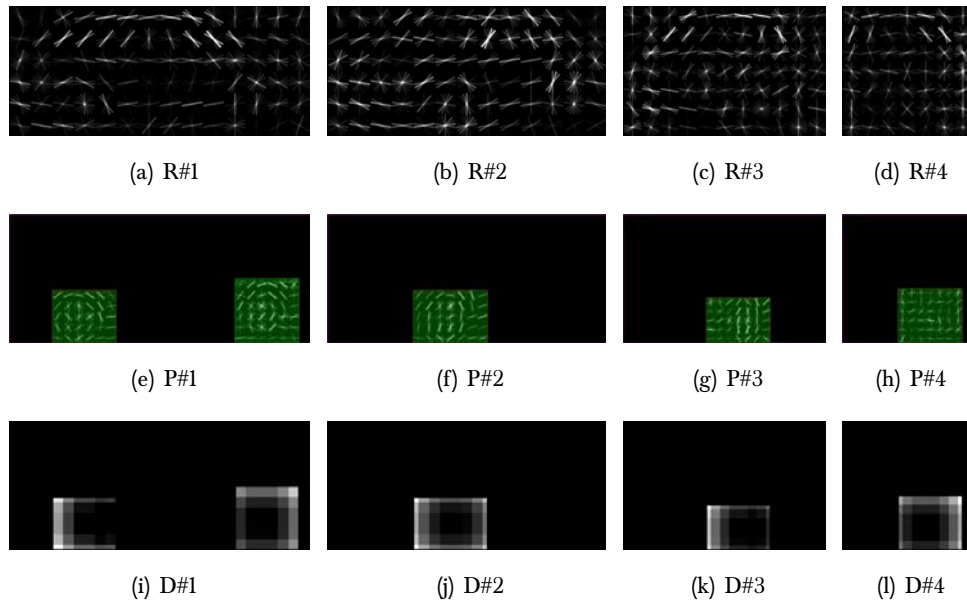


Figure 5.8: Gem_car4cwp: PCA/FLC component disambiguation, elimination bad components. Semantic parts only.

5. RESULTS

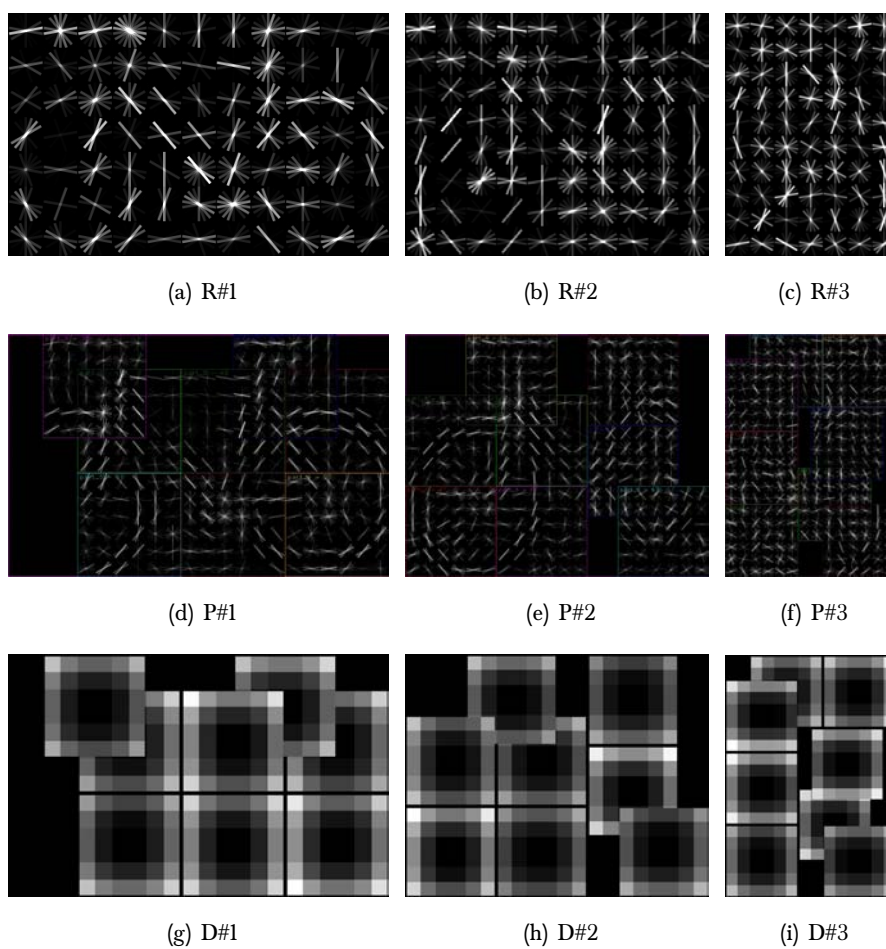


Figure 5.9: Fsz_cyc3c8p: Original setup for class *bicycle*

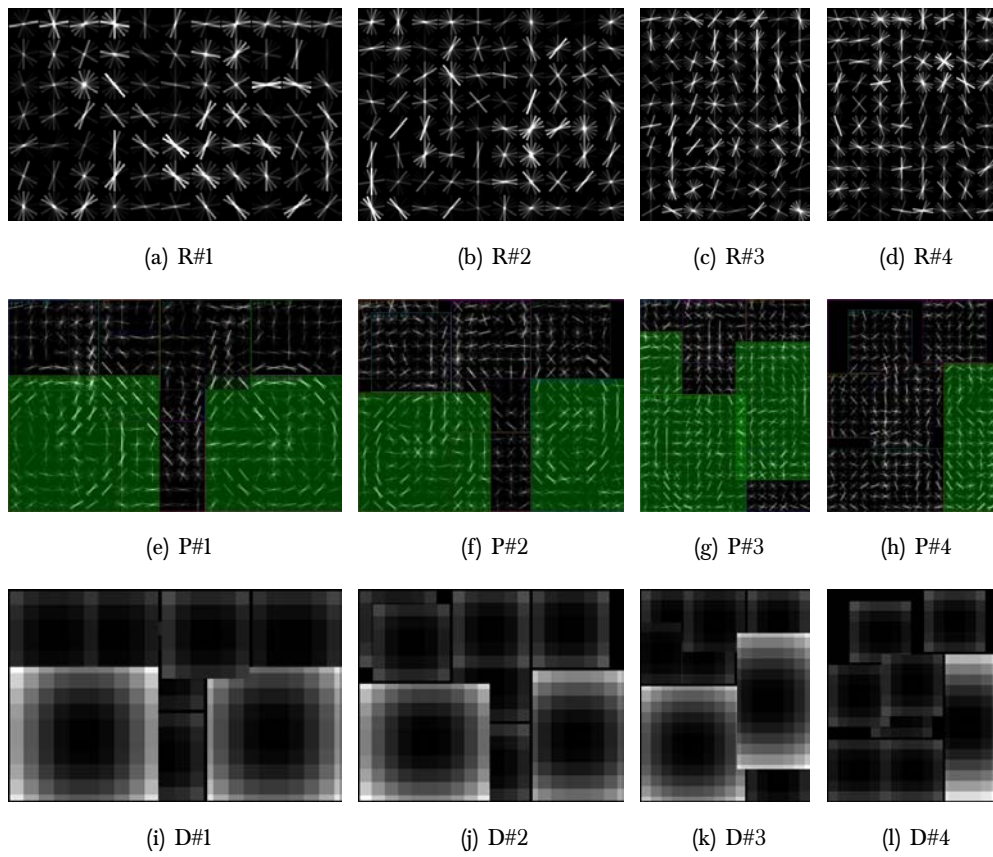


Figure 5.10: Gem_cyc4c8p: PCA/FLC with bad component elimination. Semantic parts and fill parts.

5. RESULTS

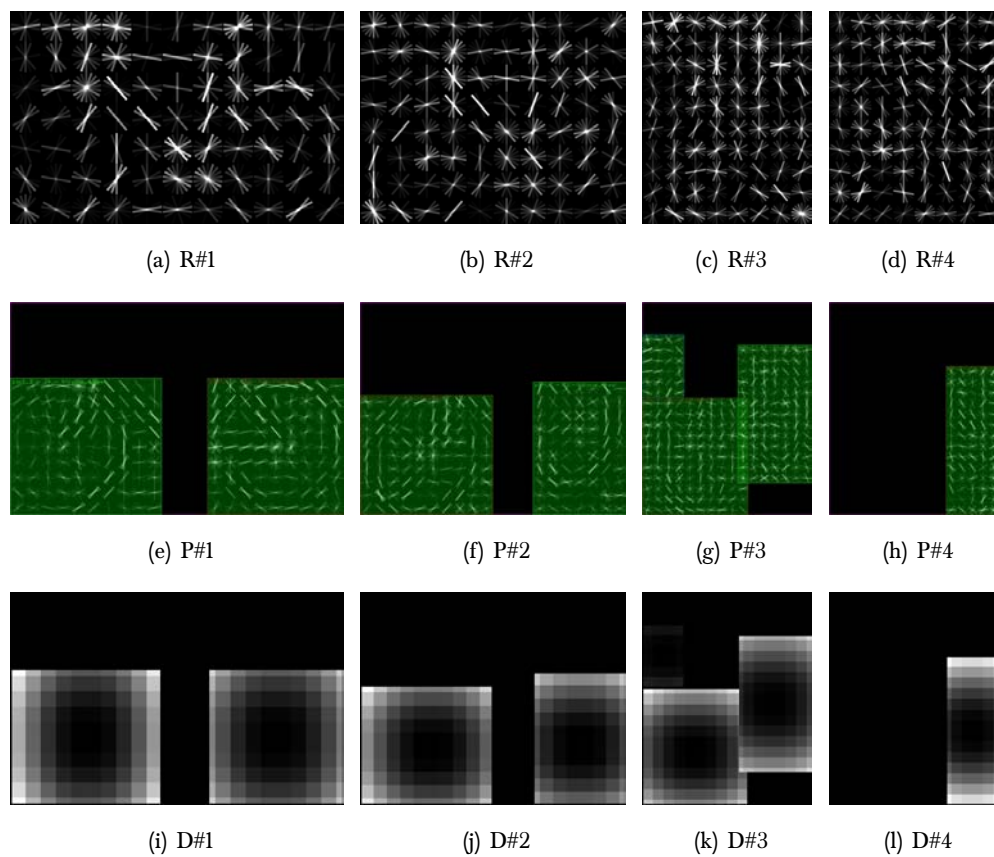


Figure 5.11: Gem_cyc4cwp: PCA/FLC with bad component elimination. Semantic parts only.

6

Discussion

We set out to investigate if we could enhance the part based deformable object model proposed by [14], by looking at the semantic structure of the categories. Though we have shown that it is possible to create components based on semantic structures that naturally occur in the objects, we find that there are a lot of different factors that influence the final performance of these models.

In this chapter we will discuss the research questions we set out to answer in the introduction of this thesis. After that we will use these insights to make some proposals for further research.

6.1 Research questions

6.1.1 Model creation

Is it possible to structure the learning data of object categories, based on the visual layout of the objects, using information acquired by analyzing the semantic structure of the category.

The component shown in figure 5.7(e) can be considered a novel view point of the data that would otherwise go undetected. Figure 5.4 shows that a regular model with four components, using the original implementation proposed by Felzenszwalb, et al., will not find this result. It finds a different view point as the fourth component, which is shown in figure 5.4(g).

6. DISCUSSION

This result shows that it is possible to find new view points of an object in the learning data when it is clustered based on the semantic part configurations in the root components. However, we show that a lot of other factors need to be taken into account for the models to perform equally or better than the original implementation.

Our results show that merely adding more components to a model decreases the precision/recall scores. By adding a new component, the amount of learning data per component drops. This is one of the main draw backs to our approach. We partly counter this by the component elimination based on the kurtosis sum score. Another option would be to try and increase the amount of available learning data.

[27] create a part based model in which both components and parts can be shared among different categories. Their results show that this approach will solve the limitations to the amount of learning data available when increasing the amount of parts or components, because the learning data is shared among components and parts. However, their approach does not add semantics to the parts and like [14] the size of the parts is given beforehand.

Another draw back is that objects that do not have a rigid structure, but instead have movable parts such as limbs, are more difficult to disambiguate based on their location, as there is a much larger variability in their location. [28] address this problem using a component based generative skeleton model. [29] proposes a model that uses a combination of pose description and HOG to address this problem.

6.1.2 Model performance

How does a model with parts placed based on a semantic structure impact the performance, compared to a model where parts were not created based on this structure.

Working with statistical models is notoriously difficult because it is hard to predict the influence of changes to such models on the final performance. We show that the precision and recall scores depend on the number of HOG cells per component. It becomes clear that a root component that is not sufficiently covered by deformable parts scores lower. This finding is consistent with [30], where a model is created on which the HOG based parts are tiled over the root component without any kind of placement heuristic. Covering the

whole root component with parts consistently scores better than covering parts of the root components, regardless of their location.

The number of components is also a factor which influences the results. Comparing the performance of the best scoring original car model to our best scoring model, is therefore somewhat problematic. Based on the difference between the root component sizes, our best model may just score worse because the root component sizes are larger than the root component sizes of the original car model.

Based on these results we conclude that a better comparison could have been made if the root component sizes of our car models would have been fixed to the size which is calculated based on the original aspect ratio group that they are created from. This size is equal to the size of the components, which is determined by the procedure described in section 3.1.1. For the the first car component this size is 11×4 HOG cells.

From the bicycle models it becomes clear that placing semantically inspired parts on a location based on the natural position of the wheels in the learning data can have a positive effect on the performance of the model. Because even though model Gem_cyc4cwp, which only has wheel parts, has fewer than half the number of cells of Gem_cyc4c8p, and a quarter of its number of parts, the performance of the two models is about equal. This is a positive result for increasing detection speeds.

The detection framework using the feature pyramid presented here essentially performs an exhaustive search over all locations and all layers of the feature pyramid, which is slow. The research presented in [6] addresses this issue and shows this may not be necessary, if a filtering stage on the most likely locations of the object is performed first. They do this in a way similar to the approach presented in [31].

6.2 Suggestions for further research

The results presented in the previous sections provide a lot of areas that can be investigated in further research. We will give a short overview on what we think are the most interesting directions:

- The sizes of the root components have a direct influence of the performance of the model. Currently the sizes of the root components of our models seem to be sub optimal. This is due to the fact that we calculate the component size based on the image sizes in the groups created by the PCA/FLC procedure described in chapter 4. An easy first step to improve the performance of the models would be to decrease the size of the root components to sizes comparable to the sizes found in the root components of the original models.
- Currently we only use one semantic part in combination with the model we learn. We would like to use more parts of different types. At this moment this is difficult though, because our framework heavily depends on the availability of bounding box information from ImageNet. [32] address this problem.
- Because we can create models that are grounded in WordNet, we can analyse the shape of the parts that occur in the models and compare them to the shapes of the parts of related concepts. For instance, a bicycle, a car and a motorbike are indirectly related through the hypernym *wheel* via the parts that they have themselves (*car wheel* and *bicycle wheel*). Using this relation we could compare the shape of the wheel parts in each of the models to create something in the line of conceptual spaces as proposed by [11]. This would be helpful to give an overview of how different concepts relate to each other, based on the shapes of their parts.
- The challenges posed by the different categories of the Pascal Visual Object Challenge vary greatly. For instance the category *boat* scores worse than other categories of inanimate objects. However, we believe that conceptually the category *boat* isn't any harder to learn than *car*. It seems to us that the reason for the poor performance of *boat* is due to the much greater variety of types of boats represented in the category, than for instance the variety of cars represented in *car*.

6.2 Suggestions for further research

Though the Pascal VOC distinguishes between the categories *car* and *bus* (both of which are *automobiles* according to WordNet), it does not distinguish between sailing boats and cruise ships. Both occur in the category *boat*. We believe our approach may help solve this problem by learning the part *sail*, by which we can at least distinguish between sailing boats and other boats.

6. DISCUSSION

References

- [1] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. **Object Detection with Discriminatively Trained Part-Based Models**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:1627–1645, 2010. [iii](#), [5](#), [7](#)
- [2] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. **The Pascal Visual Object Classes (VOC) Challenge**. *International Journal of Computer Vision*, 88(2):303–338, June 2010. [iii](#), [3](#), [34](#)
- [3] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt & Company, 1983. [1](#)
- [4] S. C. Shapiro. *Encyclopedia of Artificial Intelligence, Second Edition*. John Wiley Sons, Inc., New York, 1992. [2](#)
- [5] N. Dalal and B. Triggs. **Histograms of Oriented Gradients for Human Detection**. In *Proceedings of IEEE Conference Computer Vision and Pattern Recognition, San Diego, USA, pages 886 - 893*, 2005. [3](#), [5](#), [10](#)
- [6] K. E. A. van de Sande, T. Gevers, and C. G. M. Snoek. **Evaluating Color Descriptors for Object and Scene Recognition**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1582–1596, 2010. [3](#), [65](#)
- [7] J. Canny. **A Computational Approach to Edge Detection**. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986. [3](#)
- [8] D. G. Lowe. **Object Recognition from Local Scale-Invariant Features**. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2, ICCV '99*, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society. [3](#)
- [9] Perona P. Fergus, R. and A. Zisserman. **Object Class Recognition by Unsupervised Scale-Invariant Learning**. *Proc. of the IEEE Conf on Computer Vision and Pattern Recognition 2003*, 2003. [5](#)
- [10] J. Prinz. **The Return of Concept Empiricism**. In H. Cohen and C. Lefebvre, editors, *Handbook of Categorization in Cognitive Science, 1st Edition*. Oxford: Elsevier, 2005. [5](#), [6](#)
- [11] P. Gardenfors. **Conceptual Spaces as a Framework for Knowledge Representation**. *Mind and Matter, Vol.2(2), pp. 9-27*, 2004. [6](#), [66](#)
- [12] C. Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998. [7](#), [34](#)
- [13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. **ImageNet: A Large-Scale Hierarchical Image Database**. In *CVPR09*, 2009. [7](#), [38](#)
- [14] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. **Discriminatively Trained Deformable Part Models, Release 4**. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, No. 9*, 2010. [8](#), [15](#), [51](#), [63](#), [64](#)
- [15] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995. [11](#)
- [16] J. C. Burges. **A Tutorial on Support Vector Machines for Pattern Recognition**. *Data Mining and Knowledge Discovery 2, 121-167*, 1998. [11](#)
- [17] C. J. Yu and T. Joachims. **Learning structural SVMs with latent variables**. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09*, pages 1169–1176, New York, NY, USA, 2009. ACM. [14](#)
- [18] S. Andrews, I. Tsochantaris, and T. Hofmann. **Support vector machines for multiple-instance learning**. In *Advances in Neural Information Processing Systems 15*, pages 561–568. MIT Press, 2003. [14](#)
- [19] Y. LeCun, S. Chopra, R. Hadsell, M. A. Ranzato, and F. Huang. **A Tutorial on Energy-Based Learning**. In G. Bakir, T. Hofman, B. Scholkopf, A. Smola, and B. Taskar, editors, *Predicting Structured Data*. MIT Press, 2006. [17](#)
- [20] P. Viola, J. C. Platt, and C. Zhang. **Multiple instance boosting for object detection**. In *In NIPS 18*, pages 1419–1426. MIT Press, 2006. [23](#)

REFERENCES

- [21] J. Deng, K. Li, M. Do, H. Su, and L. Fei-Fei. **Construction and Analysis of a Large Scale Image Ontology**. Vision Sciences Society, 2009. 38
- [22] M. Turk and A. Pentland. **Eigenfaces for recognition**. *Journal of Cognitive Neuroscience* 3 (1991), 71-86., 1991. 42
- [23] R. A. Fisher. **The use of multiple measurements in taxonomic problems**. *Annals Eugen.* 7 (1936), 179-188., 1936. 45
- [24] P. N. Belhumeur, J. N. Hespanha, and D. J. Kriegman. **Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection**. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, No. 7*, 1997. 45, 46, 47
- [25] P. Wagner. **Face Recognition with OpenCV**. http://bytefish.de/dev/libfacerec/facerec_tutorial.html, 2012. 45
- [26] S. Raudys and A. K. Jain. **Small sample size effects in statistical pattern recognition: Recommendations for practitioners**. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13, 3 (1991), 252-264, 1991. 46
- [27] P. Ott and M. Everingham. **Shared parts for deformable part-based models**. In *CVPR*, pages 1513-1520, 2011. 64
- [28] M. Andriluka, S. Roth, and B. Schiele. **Pictorial structures revisited: People detection and articulated pose estimation**. In *CVPR*, pages 1014-1021, 2009. 64
- [29] M. Eichner, M. Marin-Jimenez, A. Zisserman, and V. Ferrari. **2D Articulated Human Pose Estimation and Retrieval in (Almost) Unconstrained Still Images**. *International Journal of Computer Vision*, 99:190-214, 2012. 64
- [30] Long Zhu, Yuanhao Chen, Alan L. Yuille, and William T. Freeman. **Latent hierarchical structural learning for object detection**. In *CVPR*, pages 1062-1069, 2010. 64
- [31] J. Shotton, J. M. Winn, C. Rother, and A. Criminisi. **TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context**. *International Journal of Computer Vision*, 81(1):2-23, 2009. 65
- [32] D. Kuettel, M. Guillaumin, and V. Ferrari. **Segmentation Propagation in ImageNet**. In *European Conference on Computer Vision (ECCV)*, October 2012. 66