An Attempt to Build Object Detection Models by Reusing Parts

Li Sun
Department of Computer Science
Brown University
Providence, RI 02912
li@cs.brown.edu

May 9, 2013

Abstract

Current state-of-the-art object detection methods, like the Deformable Part Model(DPM)[3] method, have achieved remarkable performance in detecting objects. However, the annotation collecting process is very costly and it won't scale as the number of categories increases. In this work, we make use of the information provided by part models of already trained deformable part models to decrease the amount of training data required for building good models. For easy categories like cars, aeroplanes and bicycles, we achieved decent qualitative results and quantitative results.

1 Introduction

Current state-of-the-art object detection methods, like the Deformable Part Model(DPM)[3] method, have achieved remarkable performance in detecting objects. However, in order to train a good model for a certain category, it usually involves collecting a large number of(typically thousands of) training examples with nontrivial human labeled annotations for these methods, which is quite costly.

The purpose of this research work is to invent a method to reduce the amount of training data required for building a reasonably good object detection model for a new class using some extra information other than training examples, like already trained models of other categories.

The purpose hasn't been achieved yet, but the goal of this technical report is to record all the progress made so far. So far, our approach is able to build fairly good models for easy categories like cars ,aeroplanes and bicycles. More details of what has been achieved and how we achieved it will be discussed below.

This report will be organized in the following way. In section 2, we will discuss the ideas that we used to achieve the purpose of building good object detection methods with less training examples. In section 3, we will give details about our algorithm. Experimental results will be given in section 4, and section 5 will discuss possible future works.

2 Ideas

To realize the purpose of building good object detection methods with less training examples, we adopt the reusable parts idea, the one-exemplar training method idea and the Deformable Part Models method.

The intuition to reuse parts comes from the fact that a lot of object categories share parts with very similar appearances. For example, cars and trucks share wheel parts; cows and sheeps share torso and leg parts. Therefore, if we've already trained good models for some categories, it is possible to reuse parts of those good models to build a model for a new category with similar parts. By using the extra information provided by similar parts, we should be able to build equally good models with less training examples.

The one-exemplar training method idea fits well with our purpose. Since the whole goal is to train good object detection models with less training examples, we want to make the best use of the information provided by each training example. The idea of one exemplar training method is to build a model from each training example so that the information provided by each example won't be averaged out.

The idea of using the deformable part models method comes naturally here as well. Since we want to reuse parts of already trained models, we want to have a clear way of representing parts. The deformable part models offer very clear models for parts called the part models.

3 Algorithm

The general idea of the algorithm is that we want to build a component(the same as defined in [3]), which is basically a submodel of a DPM model, for each training example by picking part models and placing them at positions

in this component according to the similarity of the part models' filters and underlying patches in the training example.

As in [3], each component is formally defined by a $(n+2)$-tupple $(F_0, P_1, \cdots, P_K, b)$, where F_0 is a root filter, P_i is a model for the i-th part and b is a real valued bias term. Each part model is defined by a 3-tuple (F_i, v_i, d_i) where F_i is a filter for the i-th part, v_i is a two-dimensional vector specifying an "anchor" position for part i relative to the root position, and d_i is a four dimensional vector specifying coefficients of a quadratic function defining a deformation cost for each possible placement of the part relative to the anchor position.

The main algorithm is given in Algorithm 1.

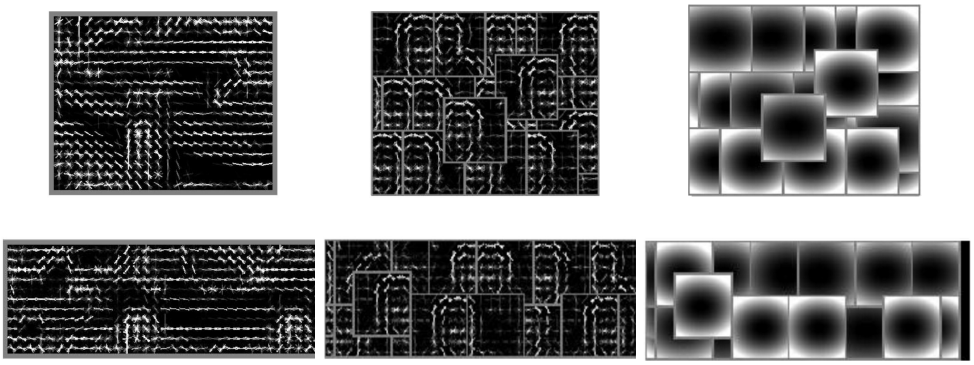
input : a library of already trained part models $P' = \{P'_1, P'_2, \cdots, P'_n\}$, their corresponding histograms of convolution scores with a large randomly selected number of patches $H = \{H_1, H_2, \cdots, H_n\}$, a set of training examples $X = \{X_1, X_2, \cdots, X_m\}$ and their corresponding segmentation maps $S = \{S_1, S_2, \cdots, S_m\}$, the number of part filters in each component K

output: a set of components $M = \{M_1, M_2, \cdots, M_m\}$ such that M_i is built from training example X_i using the part models in P'

1 **for** $i \leftarrow 1 : m$ **do**
2 Initialize M_i, with root filter $F_0 \leftarrow \mathbf{0}$, and $b \leftarrow \mathbf{0}$;
3 Initialize $S'_i \leftarrow S_i$;
4 **for** $k \leftarrow 1 : K$ **do**
5 **for** $j \leftarrow 1 : n$ **do**
6 Convolve part filter F_0^j of part model P'_j with X_i to get a covolution score map C^{ij};
7 $CT^{ij} \leftarrow$ *segment_mask*(C^{ij},S'_i) ;
  `// Collect the highest score`
8 $s^{ij} \leftarrow \max_{x,y}(CT^{ij}_{xy})$;
  `// Collect the location of the hightest score in the map`
9 $L^{ij} = \arg\max_{xy}(CT^{ij}_{xy})$;
10 $s^{ij}_s \leftarrow$*transform_score*(s^{ij},H_j)
11 **end**
12 $l \leftarrow \arg\max_j s^{ij}_s$;
13 $P^i_k \leftarrow P'_l$;`// Set` k`th part model` P^i_k `in the component` M_i
14 $v^{P^i_k} \leftarrow L_{il}$; `// Modify anchor position` $v^{P^i_k}$ `of` P^i_k
  `// Note we keep the filter` F_l `and deformation cost parameter` d_l `components of selected` P'_l
15 $S'_i \leftarrow$ *wipe_out*(S'_i, L_{il});
16 **end**
17 **end**

Algorithm 1: *Main Algorithm*

The functions *segment_mask*(Line 7) , *transform_score*(Line 10), and *wipe_out*(Line 15) will be explained in more detail later in this section.



(a) HoG features (b) Components (c) Deformation Costs

Figure 1: *Components built using the convolution scores.* (a)Visualization of HoG features of examples. (b) Visualization of built components' filters. (c) Visualization of deformation costs

3.1 Similarity Score

In Line 10 of Algorithm 1, we transform a convolution score s^{ij} to a similarity score s_s^{ij} by the function *transform_score*. If we don't do this step and use convolution scores directly, we got the models shown in Figure 1 from the algorithm.

Interestingly, the algorithm only picks part models from the person category's DPM models, although the underlying training example is from the car category.

The reason is that, in the library of part models, the magnitude of weights of their filters are very different. Therefore, the algorithm keeps picking parts with large magnitudes regardless of how similar the part filter is to the patch features. In the case of Figure 1, the part models for person category have part filters with much higher magnitude. That's why the algorithm keeps picking part models from the person category DPM model even though no patches underneath look very similar to those chosen part models.

Therefore, we want a score measure which reflects more about the similarity between a part filter and a patch than a simple convolution score.

To achieve this goal, we will adopt the following idea: if a patch is similar to the part filter, the convolution score between them should be higher than most of convolution scores of random patches from natural scene with this part filter.

According to this idea, we designed the following procedure to collect a new similarity score s_s given a convolution score s for a part model $\hat{P}$:

1. Convolve part filter component $\hat{F}$ of $\hat{P}$ with N(very large) randomly chosen patches from natural scene images, and collect the N convolution scores

2. Given convolution score s of part filter $\hat{F}$ with a patch p, a new score

$$s_s = (\text{number of collected scores smaller than } s)/N$$

This is basically what's implemented in the function *transform_score*. However, there exist some minor differences for practical reasons. First, we compute and store the histogram $\hat{H}$ of the collection of convolution score of $\hat{P}$ instead of the collection itself. Second, instead of recalculating $\hat{H}$ again and again, we pre-compute it once, and take $\hat{H}$ and $\hat{P}$ as input of the function *transform_score*.

Experimentally, we found the histograms of convolution scores are very similar to Gaussian distributions, as shown in Figure 2. So, by approximating the histogram with a Gaussian distribution, we only need to save the mean and the variance of the convolution scores. This saves us a lot of space.

By applying the function *transform_score* to transform convolutions scores, we got the components as shown in Figure 3.

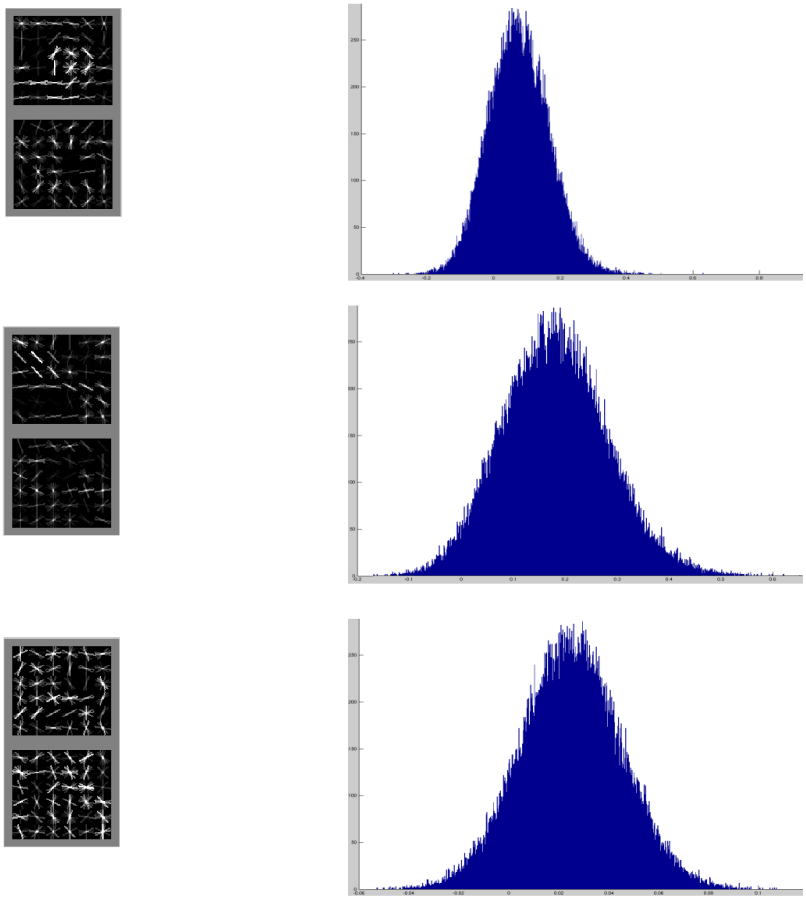
Qualitatively, we can see that we get much better components using the similarity score. Also, quantitively, when we combine all those components of the same category into a model, we got decent performance (which will be shown later in Section 4)

3.2 Using Segmentation Map

In Line 7 of Algorithm 1, we mask the convolution score map C^{ij} according to the segmentation map S'_i with the function *segment_mask*, and output the masked convolution score map CT^{ij}. In *segment_mask*, the new CT^{ij} is obtained by masking entries in C^{ij} to 0 if the fraction of overlapping area of the mask(with the same size as part filters) with true segmentation map S'_i of the example comparing to the mask's size is under a threshold t. In practise, we set $t = \frac{1}{3}$, which works well.

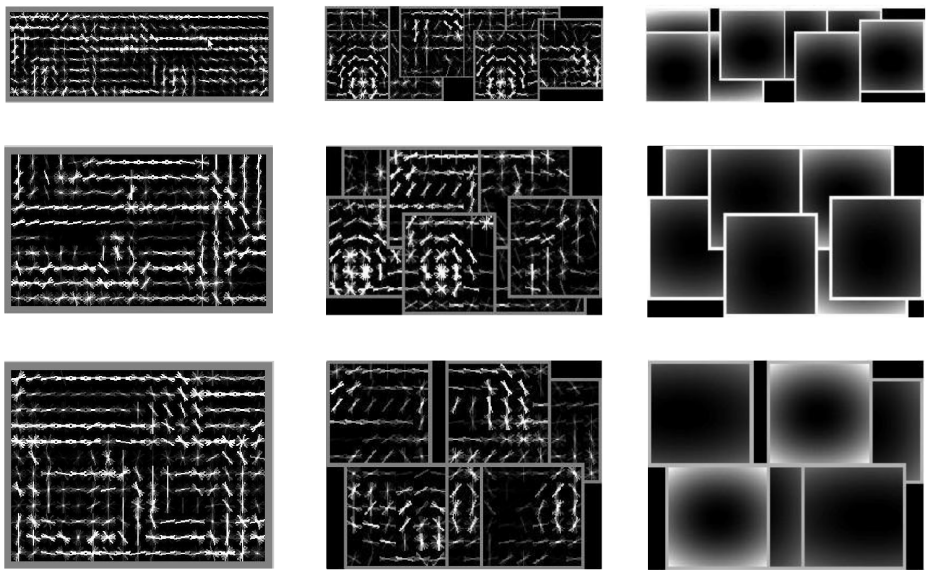
Also, to make sure each time a new area of the segmentation map is covered, we use the function *wipe_out* to update the segmentaion map S'_i, which simply sets the latest covered area of the segmentation map S'_i to 0.

The reason to make use of segmentation maps is that, for ground truth bounding boxes of some categories like the aeroplane category and the horse category, they have a large portion of background clutter. By using information provided by segmentation maps, we are able to focus finding parts which agree best with foreground, instead of background clutter.



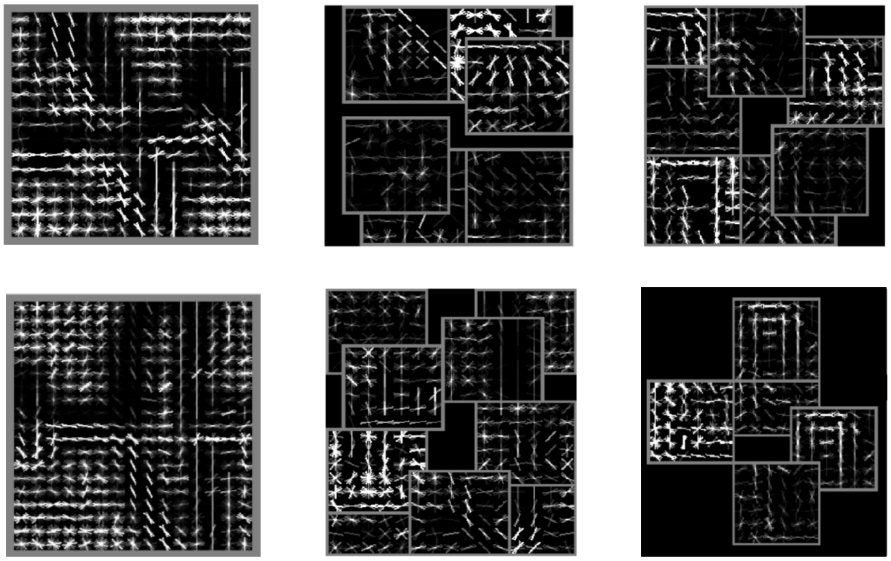
(a) Part filters (b) Histograms

Figure 2: *Histograms of convolution scores.* (a) Visualization of part filters. The upper squares are visualization of the positive weights, and the lower squares are visualization of the negative weights. (b) Corresponding histograms of convolution scores with random natural scene patches of the part filters.



(a) HoG features (b) Components (c) Deformation Costs

Figure 3: *Components built with similarity scores.* (a)Visualization of HoG features of examples. (b) Visualization of built DPM components' filters. (c) Visualization of deformation costs



(a) HoG features (b) Without segmentation maps (c) With segmentation maps

Figure 4: *Comparison of aeroplane components built with and without using segmentation maps.* (a) HoG features. (b) Components built without using segmentation maps. (c) Components built using segmentation maps.

Some qualitative comparisons after using segmentation maps are shown in Figure 4. We can see the components focus more on the foreground after using segmentation maps.

However, not surprisingly, there is almost no difference in performance between using or not using segmentation map for the car category, as shown in Table 1 which we will see later, because usually bounding boxes for cars don't have much background anyway.

More quantitative results will be shown in Section 4.

4 Results

In this section, we will give quantitative results of detection performance of variations of our algorithm in comparison with the the-state-of-the-art DPM [3] method. We will also show some visualizations of the models built with our method to give the readers a sense of how the models look like.

4.1 Performances

Object detector performance is usually measured by a PR curve, which is basically constructed by ploting a set of precision recall pairs.

So, in our case, we'll collect the precision recall pairs by Algorithm 2.

input : a test set with N_t true positives, a precision threshold p_t and a collection of components $M = \{M_1, M_2, \cdots, M_m\}$ built by Algorithm 1
output: recall rate r

1 Initialize $U \leftarrow \{\}$;
2 **for** $i \leftarrow 1 : m$ **do**
3 $\quad D_i \leftarrow build\_model(M_i)$;
4 **end**
5 **for** $i \leftarrow 1 : m$ **do**
6 $\quad PR_i \leftarrow test(D_i)$;
7 $\quad U_i \leftarrow collect\_set(PR_i, p_t)$;
8 **end**
9 $U \leftarrow \cup_{i=1}^{m} U_i$;
10 $r \leftarrow \frac{|U|}{N_t}$;

Algorithm 2: *Recall Rate Collecting Algorithm*

In line 3 of Algorithm 2, the funciton *build_model* takes in a component M, and outputs a Deformable Part Model D. D consists of two components M and M', where M' is the flipped version of M.

In line 6 of Algorithm 2, the function *test* takes in an object detection model D and outputs the PR curve on the test set PR.

In line 7 of Algorithm 2, the function *collect_set* takes in a PR curve PR and a precision threshold p_t, and output the set of true positive detections U.

Note we are cheating a little bit here, because we caliberate the detections from different D_i with the knowledge of the performance(PR curve) on the test set. In real cases, it's impossible to get those information, thus we need to figure out another method to caliberate the detections of different components. However, for the purpose of having a sense of how our algorithm is working, we want to see the performance of our algorithm with the assumption that we already have the best caliberate method.

For the following experiments, we test the object detection models on PASCAL 2007[2] test set. Note, during the training phase, we use the original *train_val* data set in PASCAL 2007 for training DPM model[3], but we only select a subset of 100 random training examples from the *train_val* data set for building models with SS(similarity score) method and SS+Seg(similarity

score and using segmentation map) method.

Precision Threshold	0.55	0.65	0.75	0.85	0.95
Recall(DPM [3])	0.56	0.51	0.45	0.35	0.22
Recall(SS)	0.5121	0.4413	0.3664	0.2831	0.1790
Recall(SS+Seg)	0.5098	0.4442	0.3699	0.2794	0.1790

Table 1: *Performance comparison of DPM method from [3](DPM), SS(similarity score), SS+Seg(similarity score and using segmentation map) methods for the car category*

Precision Threshold	0.55	0.65	0.75	0.85	0.95
Recall(DPM [3])	0.30	0.26	0.18	0.13	0.09
Recall(SS)	0.0245	0.0140	0.0140	0.0105	0.0105
Recall(SS+Seg)	0.1579	0.1509	0.1474	0.1298	0.0842

Table 2: *Performance comparison of DPM method from [3](DPM), SS(similarity score), SS+Seg(similarity score and using segmentation map) methods for the aeroplane category*

Precision Threshold	0.55	0.65	0.75	0.85	0.95
Recall(DPM [3])	0.57	0.53	0.49	0.40	0.32
Recall(SS)	0.0070	0.0070	0.0070	0.0070	0.0070
Recall(SS+Seg)	0.0105	0.0105	0.0070	0.0070	0.0070

Table 3: *Performance comparison of DPM method from [3](DPM), SS(similarity score), SS+Seg(similarity score and using segmentation map) methods for the horse category*

Note that, for the car category(Table 1), using segmentation maps or not doesn't make much difference to the performance due to the small portion of background included in a car true bounding box.

Using segmentation information does have a big impact on the performance of detecting aeroplanes(Table 2), which suggests background clutter is indeed a major reason why SS method didn't work well with the aeroplane category.

For the horse category(Table 3), the performance stays very poor with or without using segmentation information. We'll discuss more about this in Section 5.

Precision Threshold	0.55	0.65	0.75	0.85	0.95
Recall(DPM [3])	0.61	0.58	0.55	0.50	0.30
Recall(SS)	0.4451	0.3857	0.3204	0.2136	0.1751

Table 4: *Performance comparison of DPM method from [3](DPM), SS(similarity score) method for the bicycle category*

4.2 Models

In each model we built, we have about 100 flipped component pairs. Although it's impossible to visualize all component pairs in a model, we'll show the visuzliation of component pairs with the "best" detecting performances(details will be explained below). Experimentally, we found that the top 10 component pairs usually detect more than 90% of all true positive detections.

Now we will explain how we find the best component pairs.

We want a good component pair to detect as many as true positives not detected by other component pairs. That is, a good component pair should be both special and representative. Given true detection sets $U = \{U_1, U_2, \cdots, U_m\}$ of built DPMs $D = \{D_1, D_2, \cdots, D_m\}$, we use Algorithm 3 to collect k best component pairs.

input : A set of true positive sets $U = \{U_1, U_2, \cdots, U_m\}$, a positive integer k
output: A sequence of best components indexes $(j_1, j_2, \cdots, j_k)$ in order

for $i \leftarrow 1 : k$ **do**
    $j_i = \arg\max_l |U_l|$;
    for $h \leftarrow 1 : m$ **do**
        $U_h \leftarrow U_h - U_{j_i}$;
    end
end

Algorithm 3: *Collect top k components*

After finding the best component pairs for cars, aeroplanes, and bikes categories, we visualize the top 9 component pairs in Figure 5, Figure 6, and Figure 7 . Note we only visualize one component in each pair, and the other one is the flipped version of the visualized component.

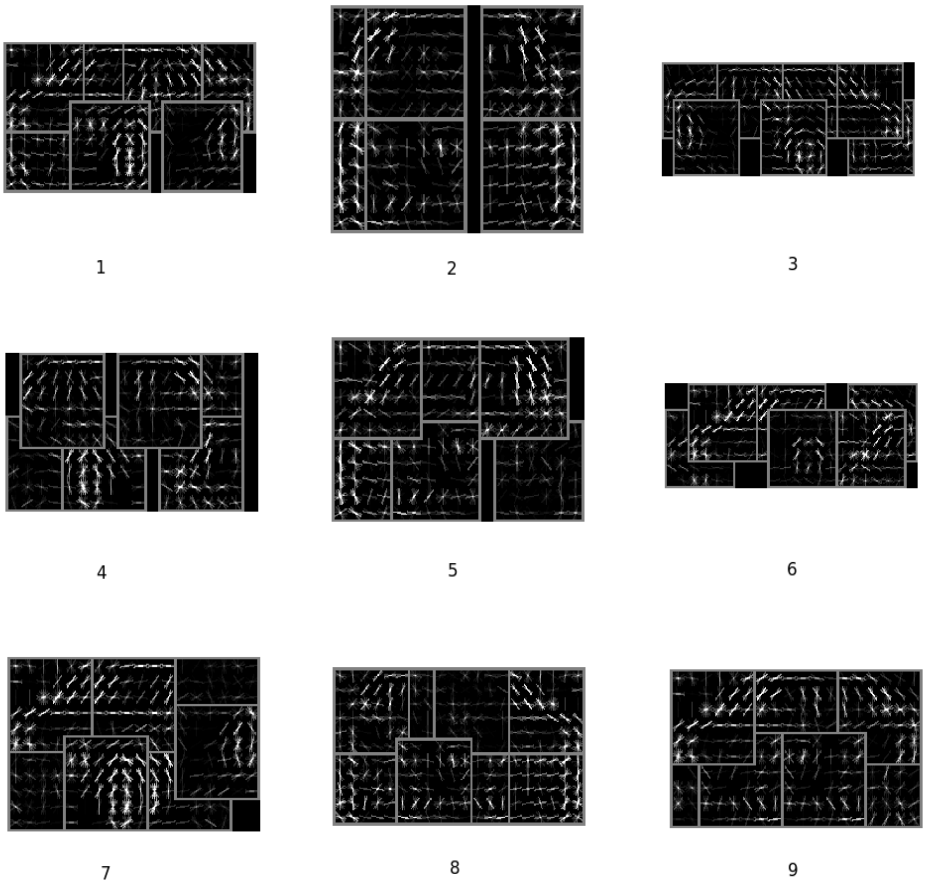


Figure 5: *Visualization of best 9 components for the car category*

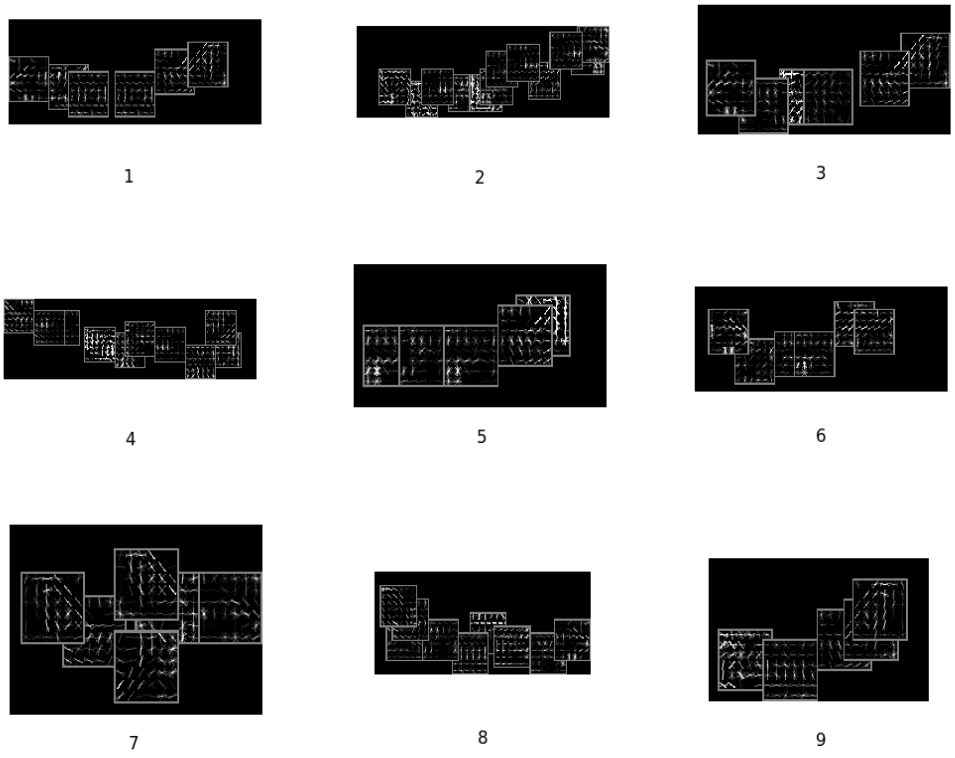


Figure 6: *Visualization of best 9 components for the aero category*

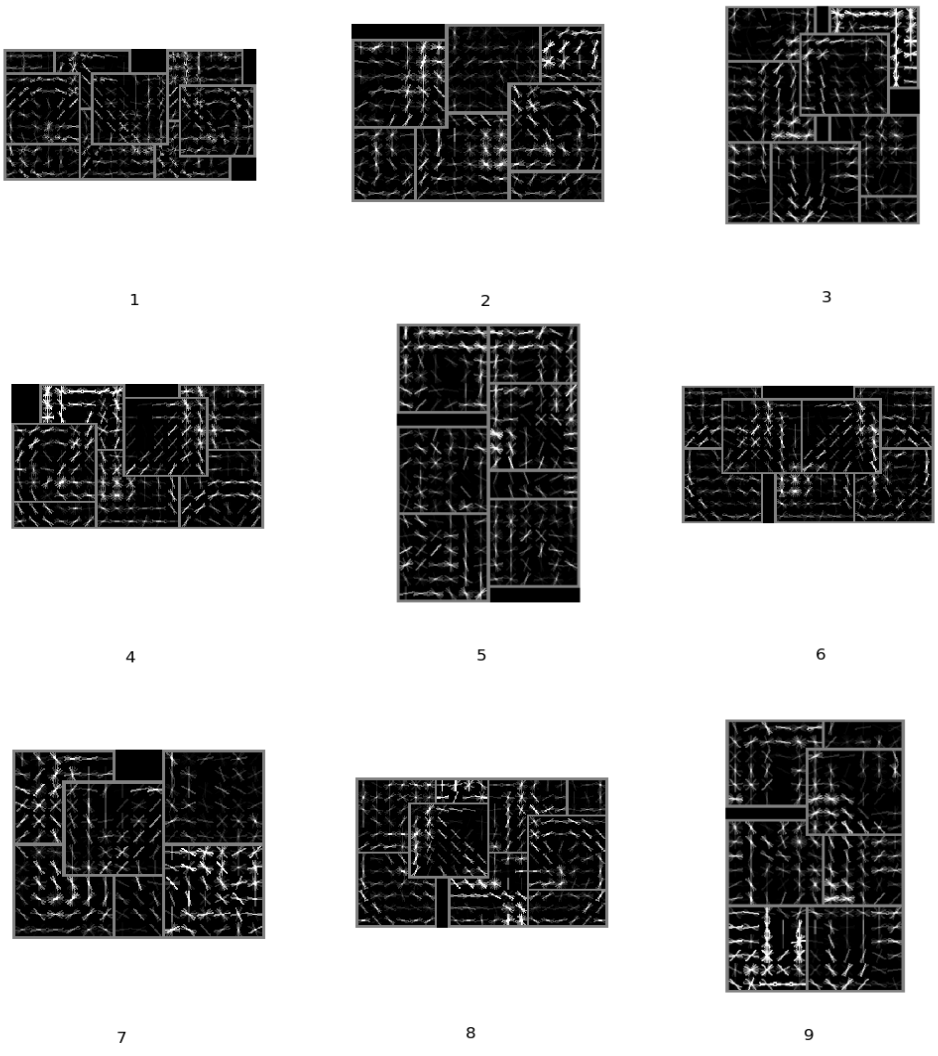


Figure 7: *Visualization of best 9 components for the bicycle category*

5 Future Works

The work so far can produce reasonable models for detecting categories like cars, aeroplanes and bicycles. However, for categories like the horse catgory, the performance is still very poor. Using segmentation maps doesn't help much with the horse category(see Table 3). We think the reason is that for horses, the root filter plays a more important role in detection, and the parts are less reliable since parts are more variant for categories like the horse category. This guess is supported by the experiments in [1] which shows dropping part models from standard DPM method only introduces a minor decrease in performance(56.8 $\rightarrow$ 55.3) for the horse category, as opposed to a large decrease for the car category(57.9 $\rightarrow$ 39.8). To give the readers some sense of how the components of the horse categories are like, we gave some examples in Figure 8. Further research could concentrate on how to incorporate root filters into this method.

Also, we need to figure out a way to caliberate all the component pairs so that a detection decision can be made collectively, combining all the components' detections. Possible directions include applying similar method as in [4] and collecting similar histograms of detecting scores for each component pairs(same idea as the similar scores).

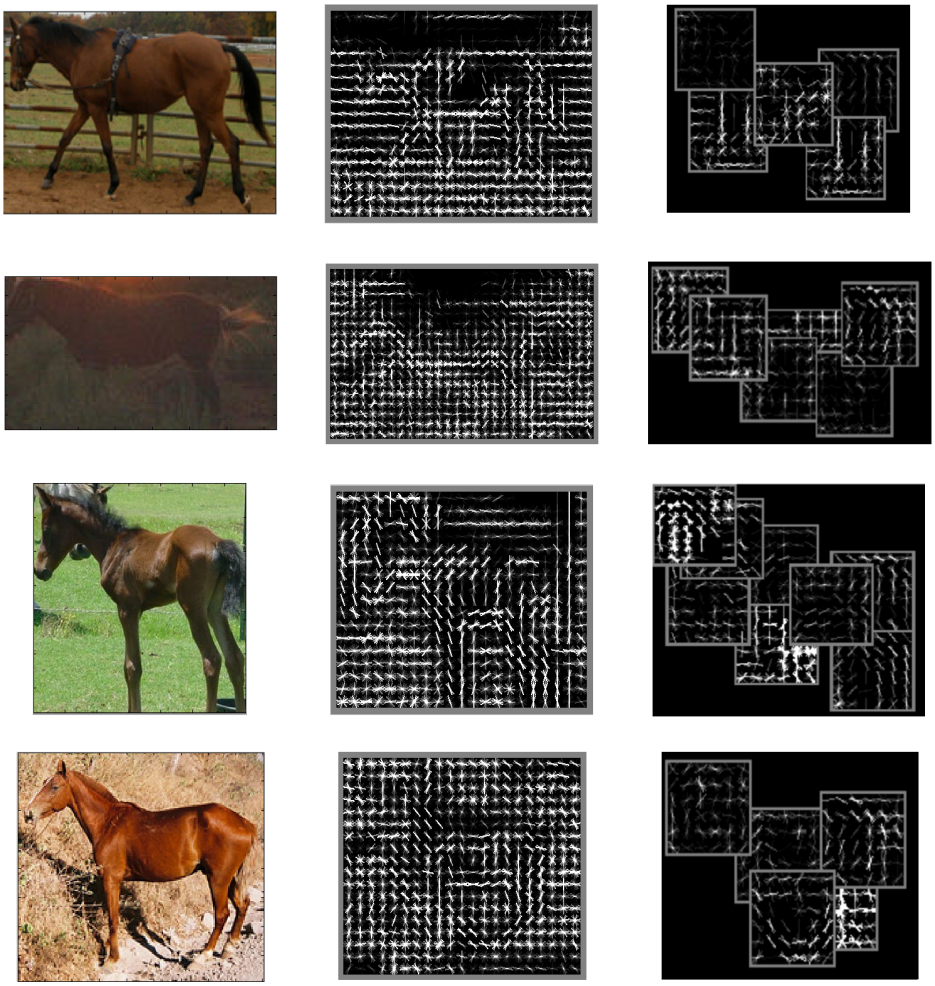
At last, as what we have seen, for our new method, we usually have as many components as number of training examples. Since we usually have a not too small number of training examples (about 100), the component number is much larger than that in a normal DPM(about 10) model. So, it's natural to be concerned with testing time speed. However, we found that among all the components, usually the top 10 components contribute to the most detections. We might be able to use this characteristic to improve testing speed. Also, applying the sparselet method in [5] is another possible way to improve testing speed.

Acknowledgement

The Author would like to thank Prof. Pedro Felzenszwalb for his guidance and valuable advices through the duration of this project.

References

[1] Santosh Kumar Divvala, Alexei A. Efros, and Martial Hebert. How important are "deformable parts" in the deformable parts model? In *ECCV Workshops (3)*, pages 31–40, 2012.



(a) Examples (b) HoG features (c) Components

Figure 8: *Horse components built by method NS+Seg(using similarity scores and segmentation maps)*

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

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

[4] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, 2011.

[5] Hyun Oh Song, Stefan Zickler, Tim Althoff, Ross Girshick, Mario Fritz, Christopher Geyer, Pedro Felzenszwalb, and Trevor Darrell. Sparselet models for efficient multiclass object detection. In *Computer Vision–ECCV 2012*, pages 802–815. Springer, 2012.