16-720 Computer Vision: Homework 5
Object Detection

Instructors: Srinivasa Narasimhan & Abhinav Gupta
TAs: Chao Liu, Gunnar Sigurdsson, Dawei Wang, Lerrel Pinto, Vivek Krishnan

Due: Tuesday, November 24, 11:59:59.$\bar{9}$ p.m.

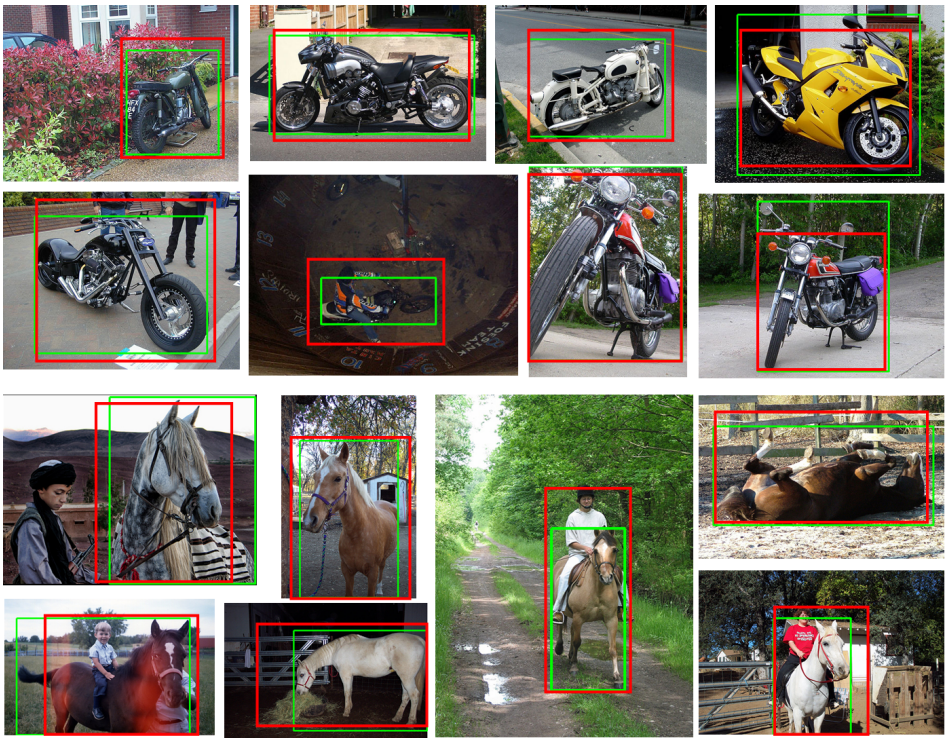


Figure 1: **Object Detection:** Given an image, we answer two questions - 1. Is there an object of interest in it? 2. If yes, "where" is it?

Instructions/Hints

1. **Warning:** It is expected that you write all the code for this homework by yourself. Please do not use external code unless otherwise mentioned.
2. Please pack your system and write-up into a single file named <**AndrewId**>**.zip**, see the complete submission checklist in Section 6.
3. Section 5 contains a list of files provided.
4. **For the implementation part, please stick to the headers, variable names, and file conventions provided. For theory questions, you don't need to write any code.**
5. **Start early!** This will take longer than the last homework and cannot be debugged as easily since there are multiple inter-connected components.
6. **Attempt to verify your implementation as you proceed:** If you don't verify that your implementation is correct on toy examples, you will risk having a huge mess when you put everything together.
7. If you have any questions, please post them on the blackboard. TAs responsible for this HW: **Gunnar Sigurdsson & Lerrel Pinto**

Overview

In this assignment, you will work on two distinct algorithms for object detection. In the first part of the assignment, you will use the Deformable Part Models (DPMs) [1], and implement Non-Maximal Suppression (NMS) for object detection. In the second part, you will use the Exemplar-SVM [3] algorithm and will reduce number of Exemplars used for detection. Finally, you will compute the Average Precision (AP) of your reduced set of exemplars on the test images.

Image Data You will be working with a subset of the PASCAL VOC2007[1]. This dataset contains a very small number of images (50) from the `bus` category as test images.

A complete submission consists of a zip file with the following folders (please keep each system in a separate folder):

1. `baselineESVM/, baselineDPM/`: the baseline folders containing your code for Sections 2,3.
2. `segTransfer/`: (if you do the extra credit): The custom folder with **all your code and data** for segmentation transfer
3. a write-up (`.pdf` format).

Detailed submission checklist in Section 6.

We provide you with a number of functions and scripts in the hopes of alleviating some tedious or error-prone sections of the implementation.

You can find a list of files provided in Section 5. Please read these descriptions.

[1] http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2007/

1 Warming up with some theory: 9 points

Suggested number of lines for each answer are mentioned.

Q1.0 (2 points, 1 line):
You are given a $M \times N$ image and a sliding window detector template of size $h \times w$. Using valid convolution (stride of 1 pixel), how many windows would you classify in this image?

Q1.1 (5 points, 2-3 lines):
Give one concrete example where using precision/recall metric is better than using accuracy metric for judging performance of a detection algorithm.

Q1.2 (2 points, 1 line):
Given 1000 bounding boxes in the training set for a single category, how many Exemplar detectors would one train? How many Dalal-Triggs type template detectors would one train?

2 Object Detection via DPMs and Non-Maximum Suppression: 40 points

All your code for this section should be in the `baselineDPM/` folder.

In this section, you try to detect objects using the Deformable Part Models (DPMs) [1], one of the most popular detection algorithms, along with the Mean-Shift Clustering which is used for Non-Maximum Suppression. First, you implement the Mean-Shift Clustering algorithm using a simple example. Later, you try to use the publicly available DPMs library (provided in this handout with pre-trained models) for detection, and your Mean-Shift algorithm is used in a way to refine the closely located outputs from the DPMs, as Non-Maximum Suppression.

2.1 Mean-Shift Clustering (20 points)

In this part, you will implement the weighted Mean-Shift Clustering algorithm. Different from the simplest Mean-Shift Clustering in which a mean of a cluster is computed by averaging the associated point positions, in weighted Mean-Shift each points also has a positive scalar value as a weight, and you need to compute the weighted mean of them:

$$\mathbf{X}_{mean} = \frac{\sum_{i=1}^{n} w_i \mathbf{X}_i}{\sum_{i=1}^{n} w_i},$$

where $\mathbf{X}_i$ is a feature location and w_i is an associated score value. Intuitively, you may think that we want to increase the importance of the points of higher weight.

Implement a function,

`[CCenters,CMemberships] = MeanShift(data,bandwidth,stopThresh).`

As input, this function takes:

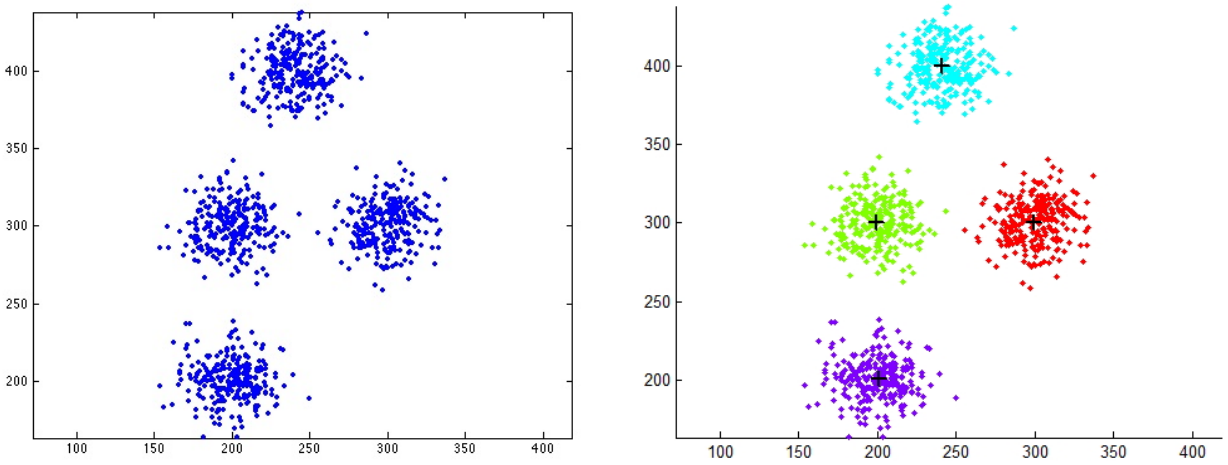
- `data`: $N \times (F+1)$ matrix where N is the number of points and F is feature dimension. The final column means a score value at the feature location (the higher, the better). `Remember` that your function should run with arbitrary dimension of F.
- `bandwidth`: the bandwidth of the window (a scalar value) to be used to update the mean. All the points which are distant from a window center in feature space within this threshold are used to update the mean for the window.
- `stopThresh`: a scalar value used to check convergence of the Mean-Shift algorithm (stop iteration if $||newCenter - oldCenter||_2 <$ `stopThresh`).

And, as output, this function should return:

- `CCenters`: $M \times F$ mode points (finally converged window centers), where M is the number of clusters and determined by your function automatically.
- `CMemberships`: $N \times 1$ membership vector which shows the clustering result. Each value of element represents the corresponding mode index (row index of `CCenters`) for each data point.

`HINT`: Your Mean-shift algorithm should start from every feature points. After iterations, each of them converges to its nearby mode of the density. The feature points that converge to a same mode are in a same cluster. You may use the `stopThresh` again, to group the same modes to handle numerical tolerance.

Q2.1.1: Submit your code, `MeanShift.m`
Q2.1.2: Use the provided sample data `q21_data.mat`, and the `q21_test.m` to test your implementation. This data has 100×3 matrix (feature dimension is 2 with a score value at the last column), and was generated by some functions with different means and a same variance. Remember that your code should automatically determine the number of clusters. You may need to adjust your *bandwidth* to get a good result. Save your $CCenters$ and $CMemberships$ into `q21_result.mat`, and submit with your code. `PLEASE` double check whether you saved the `q21_result.mat` correctly (especially, their dimensions). Also save the visualization result from `q21_test.m`, as `q21_clustering.jpg`, and submit.
Q2.1.2. (at most 3 lines in your write-up): Try with different bandwidth values. Explain briefly how does the bandwidth affect your results, and how did you determine your bandwith.



(a) Input Data

(b) Clustering Result

2.2 Detecting using Deformable Part Models (DPMs): 20 points

In this section, we try to detect objects using the provided DPM library. Particularly in this homework, we are interested in finding buses in the images. Using the provided training data `/data/bus_dpm.mat` and the `imgdetect` function, detect a car in the test image, `q42_test.jpg` by running the sample demo code, `demoDPM.m`. The direct result of `imgdetect` function is a set of on the candidate object positions bounding boxes in [xmin ymin xmax ymax] format, and can be visualized using the provided `showBoxes` function. See the `demoDPM.m`.

As can be seen, the `imgdetect` function returns every location above some internal thresholds, producing multiple bounding boxed around object location.

The main goal of this section is to refine the detection result using your Mean-Shift algorithm using a Non-Maximum Suppression (NMS). The goal of NMS is to find the best one among its neighborhood by suppressing the non-maximum candidates, and it reminds me of Mean-shift algorithm as a mode finder tool. In particular, in this homework, you use your Mean-shift algorithm for the NMS.

Implement a a function,

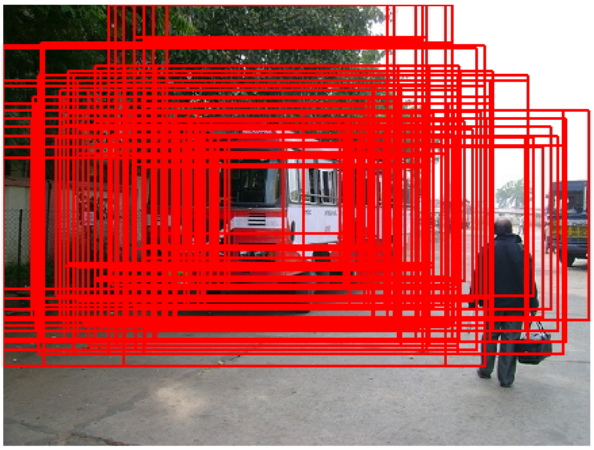
```
[refinedBBoxes] = nms(bboxes, bandwidth,K).
```

As input, this function takes

- `bboxes`: result of imgdetect function, which contains all detected bounding boxes and detection score ($Nx5$ matrix)
- `bandwidth`: same as before
- `K`: expected detection number. Only the top-`K` detections (at most) are selected as final results.

And, as output, it returns $K \times 4$ matrix where `K` is the number given from your input. The each row of `refinedBBoxes` represents a bounding box in `[xmin ymin xmax ymax]` format. Note that you should use your Mean-Shift code in this function with 4 dimensional BBox values as input features. Again, you may need to adjust your bandwidth input to get better results. `BE AWARE` that the score of DPM can have negative numbers. Before applying your Mean-Shift, make sure to make them to positive numbers using any preferred method (ex., adding a constant value or normalize them).

Q2.2.1: Submit your `nms` function
Q2.2.2 (at most 3 lines in your write-up) Explain how does your `nms` function determines the top-K candidates from Mean-Shift results (there could be different ways, so just explain the method you have implemented).
Q2.2.3 Find at least 3 bus images in the provided data folder or any images from the Internet. Apply DPM detector with your `nms` function. Save the result images using the names as `q42_result1.jpg`, `q42_result2.jpg`, `q42_result3.jpg`, and so on. Remember that DPM may failed in some situation even if the image has a bus/buses. In your submission, you may have at most 1 failure case, if you want.



(c) DPM Result

(d) After refinement via Mean-Shift NMS

3 Reducing Exemplar Detectors: 55 points

All your code for this section should be in the `baselineESVM/` folder.

In this section we will use Exemplar-SVMS [3] for detection. You will first see how detection parameters affect performance. Then you will try to cluster the images the Exemplars were trained on, so that you may select a few of them and "'compact them".
For this part of the assignment we will use Exemplar-SVM (ESVM) [3] as our base Exemplar detector. We have provided you with pre-trained ESVM detectors in `bus_esvm.mat`. If you load the file, you should find a cell array named `models`. This cell array is of dimension $1 \times N$ for N detectors. Each element in the `models` variable has the following parameters you need to understand:

- `I`: name of image on which detector was trained. (This image is in the folder `data/voc2007`).
- `gt_box`: ground truth bounding box in [xmin ymin xmax ymax] format
- `model.w`: learned detector template using HOG feature. It will be of size $h \times w \times 31$

Throughout this assignment we will use bounding boxes in the [xmin ymin xmax ymax] format. Figure 1 explains this format.

3.1 Detecting using Exemplar Detectors

As you have learned in class, object detection using ESVM involves convolving the detector template on the image. Specifically, we use the HOG feature space to do this detection task. The trained detector template $h \times w \times 31$ is convolved with the image $m \times n \times 31$ using 2D convolution (i.e. the detector is not moved in the 3rd dimension of the HOG feature).

The function `esvm_detect` performs detection on a given image. Have a look at the function file to understand its input/output. It performs NMS on the output. It accepts detection parameters in the variable `params`. For this assignment you will need to look at the following fields on `params`

- `detect_keep_threshold`: scalar in range [-1,1]. determines score threshold on detections
- `detect_levels_per_octave`: related to number of scales in which we perform detections. Generally set between [3-10]. It is the number of resizings of an image between two octaves. Figure 2 explains this.

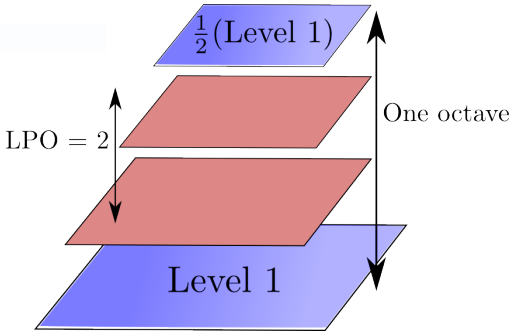


Figure 2: The concept of levels per octave. Octave refers to resizing of an image by half (blue levels). Levels per octave determines how many times we scale the image between these octaves.

Q3.0 (10 points):
You will now implement a simple function `batchDetectImagesESVM` that takes a cell array of images and ESVMs (`models` variable as described in Sec 3), and returns bounding boxes as a cell array. You will also pass detection parameters (Sec 3) `params`. You may use code from HW2 (specifically `matlabpool` in `batchToVisualWords`) to speed up detections. `boundingBoxes` is a $1 \times N$ cell array for N images. Each cell contains detections for a particular image, using `models`.

```
[boundingBoxes] = batchDetectImageESVM(imageNames, models, params)
```

3.2 Evaluating detection performance

In this section you will evaluate (both qualitatively and quantitatively) the performance of your detectors. Please use the images provided with the assignment for all the questions. For **debugging purposes only** you may use a small subset of images. Please **DO NOT** report your answers on subsets. Use the full set of images for the answers. You can use the function `utils/evalAP` for computing AP. Please see Section 5 for details.

Q3.1 Theory (5 points, 2 lines): What is Average Precision (AP)?

You will now change detection parameters and see how that affects Average Precision (AP).
Q3.2 (10 points):
Change the number of scales at which your detector is evaluated (`detectParams.detect_levels_per_octave` (lpo)). Plot a graph of AP vs. lpo on the test set. Set the lpo as `[3,5,10]`. Can you interpret the graph and explain in 2-4 lines? Submit your script as `q3_3.m`

3.3 Compacting the set of exemplar detectors

Getting maximum detection performance with a smaller number of detectors is useful in practical applications like robotics, wearable computing etc. where compute power is limited. Computational issues aside, recent work [2, 4] has shown that sometimes carefully choosing a subset of training set can actually improve performance, i.e. smaller

subset outperforms entire set. In this section, you will attempt to reduce the number of exemplar detectors while trying to maintain (or improve!!) the detection performance (AP).

Recall that for exemplar detectors, each detector corresponds to a bounding box from a training image. Hence the terms detector and bounding box can be used interchangeably for these detectors. The most basic way to reduce the number of detectors is by looking at their corresponding training image bounding boxes and working with them. You can use these bounding boxes to perform clustering and then just select the cluster centers as your final "compacted" set of detectors.[2] For the scope of this assignment, you are expected to follow such approaches and just play with different features/number of clusters. You are welcome to try more advanced techniques if you wish :)

Please do **NOT use the test data** for selecting your subset of Exemplars.

To handle bounding boxes of different sizes, you can resize them to a single size (e.g. 100×100). Figure 3 shows the average image for the clusters.

Figure 3: We cluster the Exemplar-SVMs and then display the average image for each cluster. To select the exemplars, we can just pick the exemplars that are closest to the cluster center. This figure was generated using `imdisp` by resizing every bounding box to 100×100.

Q3.3 (20 points):
For the provided Exemplar-SVM detectors, compute the `filterBank` (yes, this again!) features from HW2 on the corresponding bounding boxes. Now perform k-means clustering on the filter responses (no Bag-of-words needed), and select k Exemplar-SVMs. Compute the AP for these selected k detectors. Vary k and plot a graph of k vs. AP on the test set. Submit your code as `q3_4.m`. Please show the average images for your clusters as shown in Figure 3.

Q3.4 (10 points):
Repeat the steps in Q3.3 using a different feature [3]. Compute the AP for these selected k detectors. Vary k and plot a graph of k vs. AP on the test set. Submit your code as `q3_5.m`. Please show the average images for your clusters as shown in Figure 3.

4 Extra credit: Segmentation transfer using ESVM (20 pts)

Please place all your code for this section in a folder `segTransfer`.

One nice property of exemplar detectors is that they allow you to transfer meta-deta properties. Properites such as viewpoint, segmentation etc. associated with an exemplar can be transfered to its confident detections. Figure 4 shows some examples.

In this section, we want to see if we can transfer segmentations using ESVMs. For this, we want you to pick 2 detectors and create a segmentation mask for them. You should then show 5 examples **each**, of segmentation

[2] To avoid re-training, just pick the exemplar detector "closest" to the cluster centers

[3] you can use external code for this part by placing it in `baselineESVM/external/`

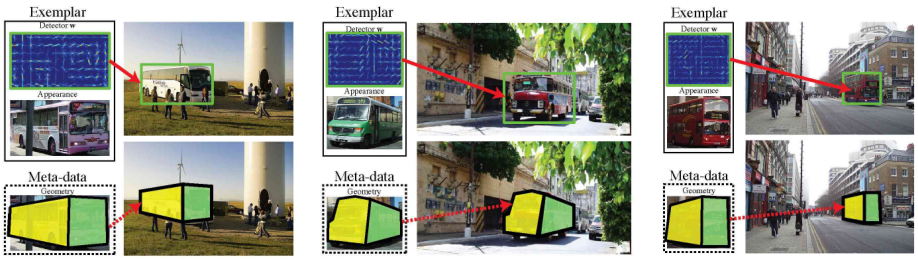


Figure 4: If we know some meta-data (e.g. 3D cuboid) of an ESVM, we can transfer it to the detection. This is possible with an Exemplar detector because the detected bounding box and the detector are very "aligned".

transfer, for both the detectors (total 10 examples). Include all your results (total 10 pairs) as shown in Figure 5. Use showHOG function to visualize your model. Your visualization need not be as appealing as Figure 5, but make sure you show 1. exemplar bounding box, 2. exemplar segmentation mask, 3. detected bounding box 4. transfered segmentation mask to the detection. You can find more pre-trained ESVM models at http://people.csail.mit.edu/tomasz/exemplarsvm/models/

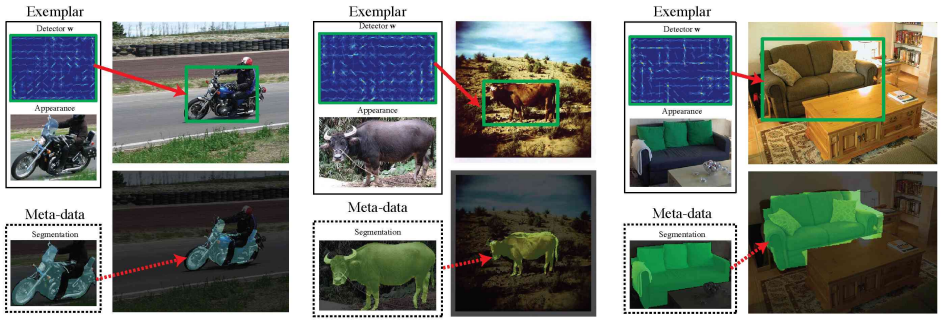


Figure 5: If we know a segmentation mask for an ESVM, we can transfer the mask to the detection.

5 HW5 Code Distribution Checklist

After unpacking `hw5.zip`, you should have a folder hw5 containing the following - code/, data/. The code/ folder contains

- `baselineDPM/`: Folder where you will write code for Section 2.
- `baselineESVM/`: Folder where you will write code for Section 3.
- `lib/esvm, lib/esvm`: Library (MATLAB based) code for DPM/ESVM.
- `utils/`: Utility (MATLAB based) code for DPM/ESVM. This folder contains two subfolders unix and win64 which have C++ code. Include the code for your platform.

In the data/ folder you will find the following files

- bus_dpm.mat, bus_esvm.mat: Pre-trained detectors for "bus" class
- bus_data.mat: Contains images for the test set along with ground truth bounding boxes.
- voc2007: folder containing images

6 HW5 Submissions Checklist

Points will be deducted for not following the guidelines. Before you submit, please read and make sure that

- All your submission is inside a zip file named with your andrew id. e.g. if the andrew id is bovik, then bovik.zip. No rar, tar.gz, tar.bz2, 7Z or other formats.
- Your writeup is named with your andrew id. It is in the pdf format.
- Upload your zip file! No digital dropbox or links please.
- Your homework zip file must contain 2-3 folders - baselineDPM/, baselineESVM, segTransfer/. Please do not include lib/, utils/ folders. Please place the code for relevant sections under the relevant folders - Section 2 in baselineDPM/ etc.
- Any EXTERNAL CODE should be placed in a folder `<andrew id>`/external. This includes export_fig, imdisp etc.
- data/ folder is DELETED. We have the images/detectors we gave you.
- File/Directory structure. Running unzip on your zip file should yield ¡andrew id¿.pdf, baselineDPM/*.m, baselineESVM/*.m, segTransfer/*.m, external/*. Please follow this carefully. If you do not attempt a part (like say extra credit) you will not have that directory in your submission.
- The segTransfer/ folder can have small .mat files.
- You have attempted all questions marked with Q.
- Include any external code you have used for Q3.5 in baselineESVM/external/

REFERENCES

[1] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. TPAMI, 32(9):16271645, 2010.
[2] A. Lapedriza, H. Pirsiavash, Z. Bylinskii, and A. Torralba. Are all training examples equally valuable? arXiv preprint, 2013.
[3] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of exemplar-svms for object detection and beyond. In ICCV, 2011.
[4] I. Misra, A. Shrivastava, and M. Hebert. Data-driven exemplar model selection. In WACV, 2014.