Object Detection with R-CNN

Albert Haque Fahim Dalvi
Computer Science Department, Stanford University
{ahaque,fdalvi}@cs.stanford.edu

Abstract

We evaluate the performance of the R-CNN algorithm with extensive hyperparameter tuning and several feature representations. Using a multivariate bounding box regression, we achieve 60.0%, 22.4%, and 22.6% AP for the cat, car, and person classes, respectively. This resulted in a mAP of 35.0%. To accelerate feature extraction, we use a multi-machine multi-GPU cluster. Additionally, we extend R-CNN to extract foregorund silhouettes from a video using GrabCut image segmentation.

1. Introduction & Related Work

Exhaustive search [8, 16, 28] and segmentation based approaches [5, 9], have both been used for object detection in the past. While exhaustive search attempts to search for all object proposals in an image, segmentation attempts to construct object proposals from the underlying pixels. More recently, the resurgence of convolutional networks has prompted research to fuse object detection with deep learning.

Exhaustive search is computationally expensive and thus requires constraints to prune the search space and reduce the number of considered locations, at the cost of completeness. In [11], Felzenszwalb *et al*. performed an exhaustive search using HOG features and a linear SVM. This resulted in excellent object detection performance. To better guide the exhaustive search, in [20], Lampert *et al*. used an appearance model, combined with a branch and bound technique, to remove scale, aspect ratio, and grid constraints from the previous exhaustive search formulation. However, their method still requires evaluation on over 100,000 proposals [1].

Segmentation-based approaches generate a foreground/background segmentation and predict objects based on their foreground segmentations. In [14], the authors segment objects based on a parts model. Each part is described by appearance and shape features. In [3], Arbelaez *et al*. use a contour detector which has been shown to provide excellent results. Although segmentation methods perform well, they are often computationally expensive.

Ever since Krizhevsky *et al*. demonstrated the success of convolutional networks [19], the computer vision community has flocked to deep learning solutions for various classification, tracking, and detection tasks [23, 21, 17, 10].

In [26], Szegedy *et al*. modify the AlexNet architecture to solve a regression task, namely bounding box regression. Inspired by this, Girshick *et al*. use convolutional network features to train a pipeline consisting of a classifier and regression step, called R-CNN [13]. Szegedy *et al*. revisit the R-CNN and reduce the computational overhead in [25]. Additionally, several other studies have reduced the computational cost for region proposals while maintaining, if not improving, detection performance (e.g. Fast R-CNN [12] and [25]).

In [15], Hariharan *et al*. use a R-CNN for simultaneous detection and segmentation. In [29], the authors fuse a parts based model with convolutional networks for fine-grained category detection.

2. R-CNN Algorithm

In [13], Girshick *et al*. investigate object detection by decomposing the task into three components: (i) region proposal generation, (ii) feature extraction, and (iii) detection.

Region Proposal Generation. Region proposals can be generated by a variety of methods (*e.g*. objectness [2], multiscale combinatorial grouping [4], category-independent object proposals [9], constrained parametric min-cuts [6], and edge boxes [30]). Girshick *et al*. propose to use selective search – a hierarchical grouping method used to generate region proposals [27]. Selective search uses a greedy algorithm to iteratively group similar regions together. On average, this results in 2,300 region proposals per image.

Feature Extraction. A convolutional network (*e.g*. AlexNet [19] or VGG-Net [24]) is used to extract features from each region. Because the CNN requires a fixed size input, Girshick *et al*. warp each region proposal and break aspect ratio. Regions are resized to 227×227 pixels such that there are exactly $p = 16$ pixels (in the new 227×227 warped region) around the original region. After performing a forward pass, the activations at the fully-activated layers (typically fc6 or fc7) are used as features. These fea-

tures are then used during training and test-time detection. A comparison of features and network architectures is in Section 5.

Detection. Given the features for each image, during test time, detection is performed in three steps. First, each region proposal is classified using a class-specific SVM trained by hard-negative mining. Second, a class-specific regression is performed to predict a new bounding box. Third, non-maximal suppression is run to combine bounding boxes with high intersection-over-union (IoU) overlap.

3. Technical Approach

3.1. Features

We compare two convolutional network architectures, namely the JonNet and VGG-16 [24] network. For JonNet, we use the fc6_ft rectified activations – resulting in 512 features per region. For the VGG-16 network, we use both the fc6 and fc7 rectified activations – resulting in 4096 features per region. Our rationale for selecting the VGG-16 network is based on the empirical finding that a network with better CNN classification performance is correlated with better detection performance when used as a feature extractor [12].

3.2. Object Classifiers

We implemented and evaluated the performance of several object classifier designs. We tested various strategies for hard mining (e.g. keeping N most difficult negatives from training, both as a constant and percentage proportional to the number of positive examples). We further experiment by removing a proportion of easy negative examples. This led to many more hard negatives (as we did not have a hard limit on the number of negatives that remain after removing easy examples), but no significant gain in performance was observed.

We experiment with several training strategies such as re-training after a constant number of images have been seen or after we have collected some preset number of hard negatives. Several pre-processing schemes were implemented such as feature standardization and zero-mean unit-variance normalization. We also experimented with ℓ_1 and ℓ_2 SVM regularization.

3.3. Bounding Box Regression

We implement four ridge regressors for each class as outlined in [13]. Since the underlying data distribution for each of the regressors is comparable, we use the same regularization penalty for each regressor. For training examples, we use all bounding boxes that have an overlap of at least 0.6 with any ground truth bounding box in the current image.

4. Code Overview

The code is divided into several files, but all the functionality can be accessed using `main.py`. All of the code can be found in the `python` directory. All parameters such as the directories to use for features and models, the CNN properties etc can be set in `settings.py`. The best hyperparameter values are already set as default in the code. Hence, to get the reported scores, it is sufficient to change the paths in `settings.py` and to run:

```
python main.py --mode extract
python main.py --mode train
python main.py --mode trainbbox
python main.py --mode test
```

Extraction mode takes optional arguments such as the number of GPUs to use. To enable bounding box regression, use the `--bbox_regression normal` option when using the `test` mode. Detailed help is available for each mode upon running `main.py` with the `-h` flag. Note that the code depends on the following libraries: `numpy`, `opencv`, `argparse`, `sklearn`, `cPickle` and `pycaffe`.

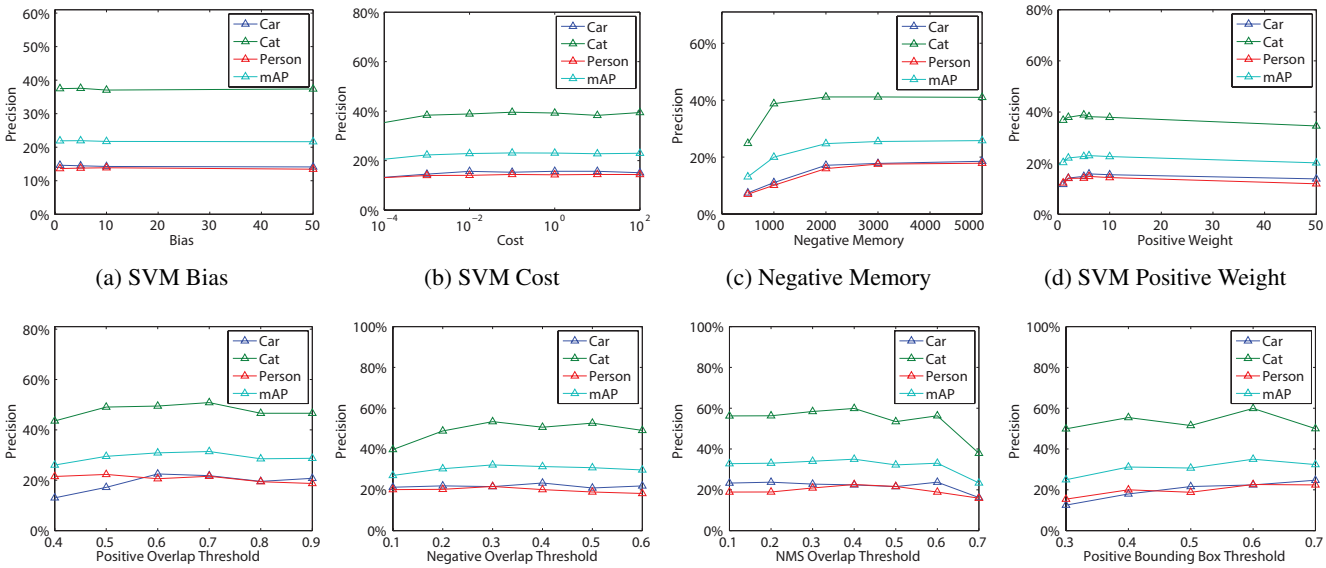
For the various source files, `train_rcnn.py` contains all the code for using the extracted features and training our classifier. We have code to train both an SVM and an SGD classifier. The file also contains code to evaluate our models on a validation set. The `test_rcnn.py` file contains the testing pipeline. It loads the features for the test images, uses the trained classifier and the bounding box regressor, and finally applies non-maximal suppression to output a final set of bounding boxes. The `det_eval.py` file is a direct port of the `det_eval.m` file provided in the starter code. `util.py` contains common functions used by all other modules, `hyperparam_tuning.m` runs the entire pipeline on various sets of hyperparameters, and finally `tester.py` tests our ported Matlab functions against their original Matlab implementations (the cross-checking requires that the Matlab Python engine be installed).

5. Experiments

We use the Caffe [18] and cuDNN [7] libraries for our convolutional network implementation. Experiments were performed on three machines:

- Workstation with 32 GiB of main memory and one NVIDIA GTX Titan X GPU with 12 GiB memory
- Amazon EC2 g2.8xlarge instance with four NVIDIA GRID K520 GPUs with 4 GiB memory each
- Amazon EC2 r3.8xlarge instance with an Intel Xeon E5-2670 CPU with 256 GiB of main memory

Our SVM [22] is implemented in liblinear.



(a) SVM Bias (b) SVM Cost (c) Negative Memory (d) SVM Positive Weight

(e) SVM Positive Overlap (f) SVM Negative Overlap (g) NMS Threshold (h) Positive Bbox Overlap

Figure 1: Hyperparameter Tuning Results

5.1. Features

As shown in Table 1, the VGG network does not perform as well as JonNet. This could be attributed to two factors: (i) insufficient hyperparameter tuning and (ii) too few training examples with too large of a feature space.

Due to the large network architecture of the VGG network, we were not able to tune the network at the same breadth as JonNet. We hypothesize that much better results can be achieved with further hyperparameter tuning and leave this as future work.

We use the non-rectified layer as our feature representation. Typically, the fully-connected layers will output 4096 non-zero real values. Since our feature space is much larger than the number of positive examples, this makes the SVM prone to overfitting. The results in Table 1 indicate this is a possible explanation for VGG's poor performance. By using the rectified features, many of the 4096 values will become zero and thus will provide less information to the SVM.

Network	Features	Car	Cat	Person	mAP
JonNet	fc6_ft	21.3%	52.3%	21.3%	31.7%
JonNet+R	fc6_ft	22.4%	60.0%	22.6%	35.0%
VGG-16	fc6	7.7%	24.7%	10.7%	14.4%
VGG-16	fc7	8.1%	35.6%	12.5%	18.7%

Table 1: Performance of different CNNs and features. 'R' indicates the use of bounding box regression.

As previously mentioned, detection performance depends on the image classification performance. Therefore we hypothesize that the fc7 layer will outperform the fc6 layer due to it's "higher level" image understanding. This is confirmed in Table 1 as the fc7 layer achieves a mAP of 18.7% while fc6 achieves 14.4%.

5.2. Object Classifiers

As a baseline, we trained classifiers with all negative and positive examples, with a class weight equal to the inverse frequency of the number of examples in that class. We saw no improvement over our hard-mined object classifiers in terms of performance.

After implementing the iterative hard-mining method of reducing the number of negatives, we performed extensive hyperparameter tuning to improve performance. We trained and evaluated all object classifiers on 1,120 different sets of hyperparameters. We modified the regularization costs, the weight on the positive class during SVM training, the maximum number of hard negatives, and the bias term. The results are shown in Figure 1.

Once we found the optimal set of hyperparameters, we began tuning our model thresholds (e.g. positive and negative overlap thresholds). We also tuned the threshold for rejecting bounding boxes while performing non-maximal suppression. The results are shown in Figure 1.

5.3. Runtime Performance & Optimizations

On a single Titan X GPU, our JonNet feature extraction took 72 minutes. Before we process each selective search proposal, we need to preprocess it to meet the requirements of the CNN. This process is CPU intensive, and hence parallelizing this portion of the extraction process gave us a significant boost in performance. VGG feature extraction, in parallel, took a total of 3 hours.

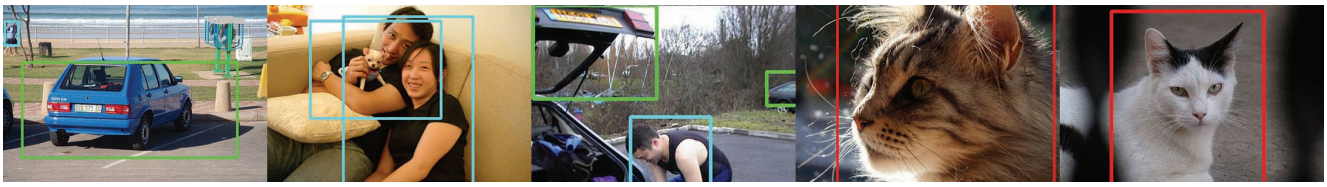


Figure 2: Successful images. Red, green, and cyan bounding boxes correspond to the cat, car, and person classes, respectively.

Figure 3: Difficult images. Red, green, and cyan bounding boxes correspond to the cat, car, and person classes, respectively.

5.4. Error Analysis

Looking at the qualitative results, we do very well on the cat class. Our model has a difficult time with the other two classes. One reason for this might be because our original CNN was trained on images with a lot of cats in them. We can still see some very nice results in Figure 2, for example, in the third successful image, the classifier is able to identify the cars using very small portions - the tail lights for one of the cars, and the side door for the other one. This was expected, as a lot of our training images are cars captured from the back and the side. On the unsuccessful cases, we notice that we have a lot of false positives for the person class. This is expected with the relatively small dataset size we had, as there are a lot of variations in people (different clothing styles and colors, various poses etc). We also sometimes detect false positives for cats, especially where there is a uniformly colored texture - like clothing. An example of this can be seen in the fourth difficult image example in Figure 3. This can be attributed to the fact that because of their weird poses, cats sometimes appear as a uniformly textured blob.

6. Extensions

6.1. Bounding Box Regression

Multivariate Ridge Regression. To improve localization performance, Girshick *et al*. propose four SVMs using the ridge regression formulation:

$$\mathbf{w}_\star = \underset{w_\star}{\text{argmin}} \sum_i^N (t_\star^{(i)} - \mathbf{w}_\star^T \phi(P^{(i)}))^2 + \alpha ||\mathbf{w}_\star||^2 \quad (1)$$

where $\mathbf{w}_\star$ denotes the vector of learned model parameters and $\star$ is one of x, y, h, w, and $\phi(P^{(i)})$ denotes the extracted features for proposal $P^{(i)}$, and N is the number of proposals. We generalize Equation 1 to the multiple-target (*i.e.* multi-variate) case:

$$\mathbf{w} = \underset{w}{\text{argmin}} \sum_i^N ||\mathbf{t}^{(i)} - \mathbf{w}^T \phi(P^{(i)})||^2 + \alpha ||\mathbf{w}||^2 \quad (2)$$

where $\mathbf{t}$ is now a vector composed of four targets. Notice that we now have a single $\mathbf{w}$ per class, as opposed to four. The results are shown in Figure 6a.

Support Vector & Elastic Net Regression. In addition to the ridge regression, we experiment with the support vector regression and elastic net regression. The support vector regression is similar to a SVM classifier but instead outputs real values. To further experiment with the regularization parameter, we use an Elastic Net regression [31] to combine both a ℓ_1 and ℓ_2 regularizer. The objective function is defined in Figure 5. where $0 \leq \rho \leq 1$ controls the strength of the ℓ_1 and ℓ_2 regularization term. We set $\rho = 0.5$.

Alpha Sensitivity. Additionally we evaluate the sensitivity of mAP to α. The results are shown in Figure 6a. For large values of α, mAP increases until 10e4 then begins to decrease. We can attribute this decrease to the fact that large values of α reduce the variance of our predictions in ridge regressions. It might be possible that because of reduced variance, our regressor does not provided optimal offsets to the correct bounding box, thus reducing our overall accuracy.

6.2. Individual Parameter Tuning

While using the same hyperparameters for all of the object classifiers reduces the overall effort, we wanted to see if there is a significant difference if we tuned each of the classifiers individually on their own set of hyperparameters. These tests were run before we tuned our bounding box regression and thresholds. Although there is an improvement for the cat class, surprisingly there is no significant improvement for the other classes as seen in table 2. One reason for this might be better features for the cat class,

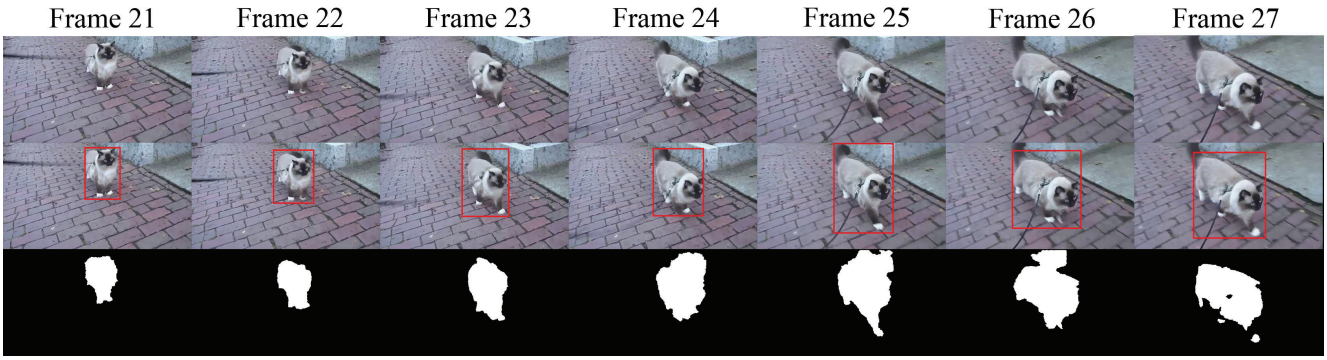
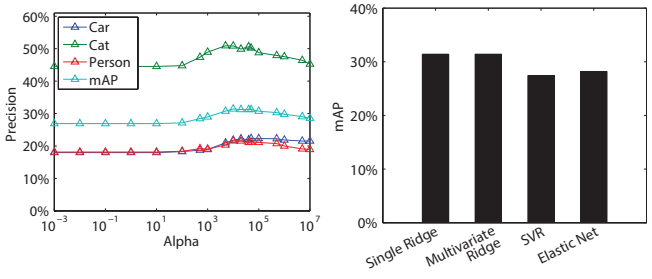


Figure 4: Foreground object detection with GrabCut image segmentation.

$$w = \underset{w}{\text{argmin}} \frac{1}{2N} \sum_{i}^{N} ||\mathbf{t}^{(i)} - \mathbf{w}^T \phi(P^{(i)})||_2^2 + \alpha\rho||w|| + \frac{\alpha(1-\rho)}{2}||w||_2^2$$

Figure 5: Optimization problem for the Elastic Net regression. It uses both ℓ_1 and ℓ_2 regularizers.



(a) Alpha (b) Linear Regressions

Figure 6: Results of bounding box regression tuning

since the CNN was trained on ImageNet, which has a lot more cats than cars and people.

	Shared Parameters	Individual Parameters
Car	22.3%	22.5%
Cat	43.4%	48.1%
Person	21.8%	21.8%

Table 2: Precision (shared vs individually-tuned)

6.3. SGD Classifier

We implemented and tuned an SGD Classifier instead of the standard SVM. The advantage to this was that we could train on the entire dataset with limited memory. We collected positives and negatives from small batches of images, and trained the classifier incrementally. After tuning the class weights and regularization penalties, we achieved a 80% accuracy on our validation set, but do poorly on the object detection task (15 AP for cat class). We also needed to tune the parameters for each class individually in this case, which might suggest that classifier was overfitting on our training data. This would also explain the poor performance on the overall object detection.

6.4. Foreground Extraction from Video

As an additional extension, we use the output bounding box of our R-CNN as input into GrabCut. Since our algorithm successfully detects cats, we apply our algorithm to a video of a cat walking outdoors.[1] We pass each frame of our video through the CNN for feature extraction. Using the original selective search code, we extract between 300 to 1000 region proposals per frame. Due to the high performing cat classifier and regression, our R-CNN returns a single bounding box for all frames. This bounding box is passed as input into GrabCut which is then run for 10 iterations. No user interaction is required at any point in the detection or segmentation process.

The results are viewable online.[2] The video contains the original frames, R-CNN bounding boxes, and foreground silhouettes. A subset of the results are shown in Figure 4.

References

[1] B. Alexe, T. Deselaers, and V. Ferrari. What is an object? In *CVPR*, 2010. 1

[2] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *PAMI*, 2012. 1

[3] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *PAMI*, 2011. 1

[4] P. Arbelaez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping. In *CVPR*, 2014. 1

[1] https://www.youtube.com/watch?v=oYTUfM1vEjs

[2] http://albert.cm/cs231b/p3/

[5] J. Carreira and C. Sminchisescu. Constrained parametric min-cuts for automatic object segmentation. In *CVPR*. IEEE, 2010. 1

[6] J. Carreira and C. Sminchisescu. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *PAMI*, 2012. 1

[7] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint*, 2014. 2

[8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. 1

[9] I. Endres and D. Hoiem. Category independent object proposals. In *ECCV*. 2010. 1

[10] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *CVPR*, 2014. 1

[11] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 2010. 1

[12] R. Girshick. Fast r-cnn. *arXiv preprint*, 2015. 1, 2

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 2

[14] C. Gu, J. J. Lim, P. Arbeláez, and J. Malik. Recognition using regions. In *CVPR*, 2009. 1

[15] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*. 2014. 1

[16] H. Harzallah, F. Jurie, and C. Schmid. Combining efficient object localization and image classification. In *CVPR*. IEEE, 2009. 1

[17] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint*, 2014. 1

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint*, 2014. 2

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 1

[20] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Efficient subwindow search: A branch and bound framework for object localization. *PAMI*, 2009. 1

[21] W. Ouyang, P. Luo, X. Zeng, S. Qiu, Y. Tian, H. Li, S. Yang, Z. Wang, Y. Xiong, C. Qian, et al. Deepid-net: multi-stage and deformable deep convolutional neural networks for object detection. *arXiv preprint*, 2014. 1

[22] F. e. a. Pedregosa. Scikit-learn: Machine learning in Python. *JMLR*, 2011. 2

[23] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint*, 2013. 1

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint*, 2014. 1, 2

[25] C. Szegedy, S. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. *arXiv preprint*, 2014. 1

[26] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *NIPS*, 2013. 1

[27] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *IJCV*, 2013. 1

[28] P. Viola and M. J. Jones. Robust real-time face detection. *IJCV*, 2004. 1

[29] N. Zhang, J. Donahue, R. Girshick, and T. Darrell. Part-based r-cnns for fine-grained category detection. In *ECCV*. 2014. 1

[30] C. L. Zitnick and P. Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*. 2014. 1

[31] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society*, 2005. 4