DETECTING COMMON OBJECTS IN CONEXT

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Tsung-Yi Lin

August 2017

© 2017 Tsung-Yi Lin
ALL RIGHTS RESERVED

DETECTING COMMON OBJECTS IN CONEXT

Tsung-Yi Lin, Ph.D.

Cornell University 2017

Visual scene understanding is a basic function of human perception and one of the primary goals of computer vision. Object detection, which involves recognizing and localizing objects present in an environment, is a fundamental task in scene understanding. In the past years, object detection is one of most rapidly developing research areas in computer vision. Progress has been made through a combined efforts of large scale datasets, high quality annotations, and feature representations learned with novel convolutional neural network architectures.

This thesis discusses both the process of dataset creation and the subsequent challenges in algorithm design for object detection. We create a large scale visual dataset Common Object in COntext (COCO) that contains objects in everyday scenes and detailed instance segmentation masks. The COCO dataset aims to enable research on detecting objects in an unconstrained environment and presents the combined challenges of recognizing objects in context and accurately localizing instances in 2D.

We discuss the algorithm design to address the subsequent challenges in COCO dataset. First, we focus on learning multiscale feature representations to improve object detection performance over a wide range of object scales. We show that by leveraging the pyramidal shape of feature hierarchy in convolutional neural network (ConvNet), we can learn multiscale pyramidal feature representations that are semantic strong at all levels. The proposed Feature Pyramid Networks (FPN) provides generic feature presentations that greatly

improve performance in terms of both accuracy and speed for various object detection applications.

We then identify extreme class imbalance of foreground and background examples is an inherent challenge for designing the training objective of object detection algorithms. We propose a novel Focal Loss that focuses learning from important examples and ignore most easy background examples to solve the issue. We propose RetinaNet, a simple one-stage dense object detector using both the focal loss and FPN, and achieve state-of-the-art performance for both accuracy and speed on COCO dataset.

BIOGRAPHICAL SKETCH

Tsung-Yi Lin received his M.S. in Electrical and Computer Engineering from University of California, San Diego in 2013 and his B.S. in Electrical Engineering from National Taiwan University in 2009. He started the Ph.D. program in 2014 advised by Prof. Serge Belongie at Cornell Tech and Cornell University. His research interests are in crowdsourcing, object detection, and instance segmentation. Tsung-Yi led the effort to create Common Object in Context (COCO) dataset and currently serves as secretary and board member of the Common Visual Data Foundation.

To my parents, Chiou-Feng Wang and Ching-Liang Lin.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor Serge Belongie for the all guidance intellectually and morally. I learned from him to always appreciate the bright side and nurish it to become brilliant.

I would like to thank my co-advior James Hays for the advice and support over years, in particular the time I just started my Ph.D. program. I want to thank my mentor Dr. Piotr Dollár for the long-term support and mentoring. I learned from Piotr to conduct research in a rigorous and organized approach.

I had luck to work with amazing people in COCO consortium. Thanks everyone for providing valuable insight during our weekly meeting and all the hard work to build COCO dataset and run annual COCO challenges, Genevieve Patterson, Matteo R. Ronchi, Yin Cui, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, Larry Zitnick, Piotr Dollár.

I want to thank my collaborators in Facebook AI Research for the valuable discussion on deep learning and object recognition, Ross Girshick, Bharath Hariharan, Kaiming He, Priya Goyal.

To my wonderful labmates, in San Diego and New York, I want to thank them for their friendship and accompany, Andreas Veit, Yin Cui, Michael Wilber, Hani Altwaijry, Mohammad Moghimi, Iljung Kwak, Chen-Yu Lee, Kai Wang, Catherine Wah, Steve Branson.

Finally, I want to thank my wife Yu-Ching Yeh and my parents Chiou-Feng Wang and Ching-Liang Lin for their love and patience when I am away pursuing my research agenda. Without their support, I would not be able to start this fantastic journey.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Visual scene understanding is a basic function of human perception. It involves numerous tasks including estimating 3D scene, localizing objects in 2D and 3D, recognizing semantics such as object categories and attributes and providing a semantic description of the scene that explains the relationships between objects and the environment. In the past few years, scene understanding in computer vision is developing rapidly through the combined efforts of large scale dataset creation and machine learning algorithm design. Recently, computer vision is able to classify an image with more than thousands of categories by training convolutional neural networks (ConvNets) [61, 110, 48] on more than millions of images [18, 129, 59] and achieve human-level performance.

While computer achieves excellent performance on image classification task, it is just an early attempt to enable machines to perceive the world. Beyond classifying an image into one or more concept labels, recognizing and localizing all objects present in an everyday scene is an important next step for solving the ultimate goal of scene understanding. Detecting all objects in a scene is challenging. The objects can be small in the background, partially occluded, amid cluttered. This introduces three core research problems for object detection: detecting objects in non-iconic views, reasoning contextual information of objects, and the precise 2D localization.

Object detection can be formulated as classifying regions with given locations, scales, and shapes on an image. To detect objects in non-iconic views, the feature representations should be robust to occlusion, encode contextual information, and possess enough resolution to predict precise 2D location. De-

signing feature representations has been an active research area in computer vision. Before ConvNet is widely adopted for learning representations, feature design requires sophisticated manual engineering [17][78][74]. Recently, a series of research that uses ConvNets to learn feature representations improves performance significantly over hand-crafted features [61][110][48]. The ConvNets learn feature representations through end-to-end training, which only requires to define input data, model architecture, and training objective for learning. The ConvNet model architecture, which consists of multiple convolution and pooling operations, allows learning semantic strong features that contains global contextual information for a local region [92]. The approach demonstrates strong empirical results for object detection[41, 91, 46] and numerous scene understanding tasks [110, 48]. However, the multiple pooling operations in ConvNet sacrifice feature resolution which is an issue to precisely localize objects, particularly in small scales.

The set of all possible object locations is nearly infinite. However, only limited object instances appear in an image, which means most regions on an image are in the background or partially overlapped with objects. It can be difficult to apply machine learning algorithms with such extreme class imbalance. During training, the optimization process could get stuck with a model that predicts every example as the background class to obtain low training loss. The recent object detectors avoid the problem by resampling training data such that the training loss is computed only on a class balanced subset of examples. The existing subsampling procedures are based on heuristics such as fixing the ratio of background and foreground examples, sampling by training losses, or using two-stage cascade [71, 41, 91].

In this thesis, we introduce a large scale object detection dataset and algorithms to solve the subsequent challenges from the new dataset. In Chapter 2, we introduce Common Object in COntext (COCO) dataset that serves as the training source and testbed for detecting objects in everyday scenes [69]. The COCO dataset currently contains 200k images with annotations, over one million instances, and 80 common object categories. The categories are selected such that they can be easily recognized by 5-year-old kids and cover objects present in a wide range of scenes including indoor and outdoor. All instances are annotated with segmentation masks to enable research on precise 2D localization.

To address challenges present in COCO dataset, first we focus on improving object detection performance by learning multiscale feature representations, in particular developing algorithms that can efficiently compute features that are both semantic strong and higher resolution. The key idea is to leverage feature hierarchy in ConvNet and merge spatially rich information from low-level features with the high-level object knowledge encoded in upper network layers with top-down and lateral connections. In Chapter 3, we introduce SharpMask that predict precise 2D instance segmentation masks by refining low resolution mask encoding through top-down and lateral connections. [85]. In Chapter 4, we continue developing the idea and propose Feature Pyramid Networks (FPN) which learn generic multiscale features representations that are semantic strong at all feature levels. The FPN only adds marginal costs to bottom-up ConvNet architecture and shows great improvement for both accuracy and speed on various object detection applications, including object proposals, box localization, and instance segmentation [67].

In Chapter 5, we propose Focal Loss, a novel training objective that focuses on important examples to address the extreme class imbalance issue [68]. The proposed method simply adjusts the shape of the training loss such that it focuses on hard examples and ignores easy examples mostly coming from the background class. The proposed focal loss provides a new path to address class imbalance problem other than the heuristics used in existing methods. We propose a single-stage dense object detector RetinaNet, which leverages both feature representations from FPN and the focal loss, and achieve state-of-the-art performance for both speed and accuracy.

CHAPTER 2

COMMON OBJECTS IN CONTEXT DATASET

2.1 Introduction

One of the primary goals of computer vision is the understanding of visual scenes. Scene understanding involves numerous tasks including recognizing what objects are present, localizing the objects in 2D and 3D, determining the objects' and scene's attributes, characterizing relationships between objects and providing a semantic description of the scene. The current object classification and detection datasets [18, 27, 122, 20] help us explore the first challenges related to scene understanding. For instance the ImageNet dataset [18], which contains an unprecedented number of images, has recently enabled breakthroughs in both object classification and detection research [61, 39, 102]. The community has also created datasets containing object attributes [29], scene attributes [82], keypoints [8], and 3D scene information [109]. This leads us to the obvious question: what datasets will best continue our advance towards our ultimate goal of scene understanding?

We introduce a new large-scale dataset that addresses three core research problems in scene understanding: detecting non-iconic views (or non-canonical perspectives [81]) of objects, contextual reasoning between objects and the precise 2D localization of objects. For many categories of objects, there exists an iconic view. For example, when performing a web-based image search for the object category "bike," the top-ranked retrieved examples appear in profile, unobstructed near the center of a neatly composed photo. We posit that current recognition systems perform fairly well on iconic views, but struggle to rec-

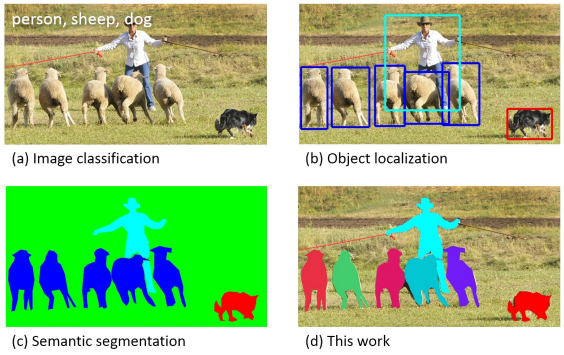


Figure 2.1: While previous object recognition datasets have focused on (a) image classification, (b) object bounding box localization or (c) semantic pixel-level segmentation, we focus on (d) segmenting individual object instances. We introduce a large, richly-annotated dataset comprised of images depicting complex everyday scenes of common objects in their natural context.

ognize objects otherwise – in the background, partially occluded, amid clutter [51] – reflecting the composition of actual everyday scenes. We verify this experimentally; when evaluated on everyday scenes, models trained on our data perform better than those trained with prior datasets. A challenge is finding natural images that contain multiple objects. The identity of many objects can only be resolved using context, due to small size or ambiguous appearance in the image. To push research in contextual reasoning, images depicting scenes [122] rather than objects in isolation are necessary. Finally, we argue that detailed spatial understanding of object layout will be a core component of scene analysis. An object's spatial location can be defined coarsely using a bounding box [27] or with a precise pixel-level segmentation [9, 98, 5]. As we demonstrate, to measure either kind of localization performance it is essential for the dataset to have every instance of every object category labeled and fully segmented. Our dataset is unique in its annotation of instance-level segmentation masks, Fig. 4.1.

To create a large-scale dataset that accomplishes these three goals we em-

ployed a novel pipeline for gathering data with extensive use of Amazon Mechanical Turk. First and most importantly, we harvested a large set of images containing contextual relationships and non-iconic object views. We accomplished this using a surprisingly simple yet effective technique that queries for pairs of objects in conjunction with images retrieved via scene-based queries [80, 122]. Next, each image was labeled as containing particular object categories using a hierarchical labeling approach [19]. For each category found, the individual instances were labeled, verified, and finally segmented. Given the inherent ambiguity of labeling, each of these stages has numerous tradeoffs that we explored in detail.

The Common Objects in COntext (COCO) dataset contains 91 common object categories with 82 of them having more than 5,000 labeled instances, Fig. 2.6. In total the dataset has 2,500,000 labeled instances in 328,000 images. In contrast to the popular ImageNet dataset [18], COCO has fewer categories but more instances per category. This can aid in learning detailed object models capable of precise 2D localization. The dataset is also significantly larger in number of instances per category than the PASCAL VOC [27] and SUN [122] datasets. Additionally, a critical distinction between our dataset and others is the number of labeled instances per image which may aid in learning contextual information, Fig. 2.5. COCO contains considerably more object instances per image (7.7) as compared to ImageNet (3.0) and PASCAL (2.3). In contrast, the SUN dataset, which contains significant contextual information, has over 17 objects and "stuff" per image but considerably fewer object instances overall.

Figure 2.2: Example of (a) iconic object images, (b) iconic scene images, and (c) non-iconic images.

2.2 Related Work

Throughout the history of computer vision research datasets have played a critical role. They not only provide a means to train and evaluate algorithms, they drive research in new and more challenging directions. The creation of ground truth stereo and optical flow datasets [99, 4] helped stimulate a flood of interest in these areas. The early evolution of object recognition datasets [30, 42, 17] facilitated the direct comparison of hundreds of image recognition algorithms while simultaneously pushing the field towards more complex problems. Recently, the ImageNet dataset [18] containing millions of images has enabled breakthroughs in both object classification and detection research using a new class of deep learning algorithms [61, 39, 102].

Datasets related to object recognition can be roughly split into three groups: those that primarily address object classification, object detection and semantic scene labeling. We address each in turn.

Image Classification The task of object classification requires binary labels indicating whether objects are present in an image; see Fig. 4.1(a). Early datasets of this type comprised images containing a single object with blank

backgrounds, such as the MNIST handwritten digits [64] or COIL household objects [75]. Caltech 101 [30] and Caltech 256 [42] marked the transition to more realistic object images retrieved from the internet while also increasing the number of object categories to 101 and 256, respectively. Popular datasets in the machine learning community due to the larger number of training examples, CIFAR-10 and CIFAR-100 [60] offered 10 and 100 categories from a dataset of tiny 32×32 images [117]. While these datasets contained up to 60,000 images and hundreds of categories, they still only captured a small fraction of our visual world.

Recently, ImageNet [18] made a striking departure from the incremental increase in dataset sizes. They proposed the creation of a dataset containing 22k categories with 500-1000 images each. Unlike previous datasets containing entry-level categories [79], such as "dog" or "chair," like [117], ImageNet used the WordNet Hierarchy [31] to obtain both entry-level and fine-grained [121] categories. Currently, the ImageNet dataset contains over 14 million labeled images and has enabled significant advances in image classification [61, 39, 102].

Object detection Detecting an object entails both stating that an object belonging to a specified class is present, and localizing it in the image. The location of an object is typically represented by a bounding box, Fig. 4.1(b). Early algorithms focused on face detection [50] using various ad hoc datasets. Later, more realistic and challenging face detection datasets were created [54]. Another popular challenge is the detection of pedestrians for which several datasets have been created [17, 20]. The Caltech Pedestrian Dataset [20] contains 350,000 labeled instances with bounding boxes.

For the detection of basic object categories, a multi-year effort from 2005 to

2012 was devoted to the creation and maintenance of a series of benchmark datasets that were widely adopted. The PASCAL VOC [27] datasets contained 20 object categories spread over 11,000 images. Over 27,000 object instance bounding boxes were labeled, of which almost 7,000 had detailed segmentations. Recently, a detection challenge has been created from 200 object categories using a subset of 400,000 images from ImageNet [96]. An impressive 350,000 objects have been labeled using bounding boxes.

Since the detection of many objects such as sunglasses, cellphones or chairs is highly dependent on contextual information, it is important that detection datasets contain objects in their natural environments. In our dataset we strive to collect images rich in contextual information. The use of bounding boxes also limits the accuracy for which detection algorithms may be evaluated. We propose the use of fully segmented instances to enable more accurate detector evaluation.

Semantic scene labeling The task of labeling semantic objects in a scene requires that each pixel of an image be labeled as belonging to a category, such as sky, chair, floor, street, etc. In contrast to the detection task, individual instances of objects do not need to be segmented, Fig. 4.1(c). This enables the labeling of objects for which individual instances are hard to define, such as grass, streets, or walls. Datasets exist for both indoor [109] and outdoor [106, 9] scenes. Some datasets also include depth information [109]. Similar to semantic scene labeling, our goal is to measure the pixel-wise accuracy of object labels. However, we also aim to distinguish between individual instances of an object, which requires a solid understanding of each object's extent.

A novel dataset that combines many of the properties of both object detec-

tion and semantic scene labeling datasets is the SUN dataset [122] for scene understanding. SUN contains 908 scene categories from the WordNet dictionary [31] with segmented objects. The 3,819 object categories span those common to object detection datasets (person, chair, car) and to semantic scene labeling (wall, sky, floor). Since the dataset was collected by finding images depicting various scene types, the number of instances per object category exhibits the long tail phenomenon. That is, a few categories have a large number of instances (wall: 20,213, window: 16,080, chair: 7,971) while most have a relatively modest number of instances (boat: 349, airplane: 179, floor lamp: 276). In our dataset, we ensure that each object category has a significant number of instances, Fig. 2.5.

Other vision datasets Datasets have spurred the advancement of numerous fields in computer vision. Some notable datasets include the Middlebury datasets for stereo vision [99], multi-view stereo [101] and optical flow [4]. The Berkeley Segmentation Data Set (BSDS500) [3] has been used extensively to evaluate both segmentation and edge detection algorithms. Datasets have also been created to recognize both scene [82] and object attributes [29, 62]. Indeed, numerous areas of vision have benefited from challenging datasets that helped catalyze progress.

2.3 Image Collection

We next describe how the object categories and candidate images are selected.

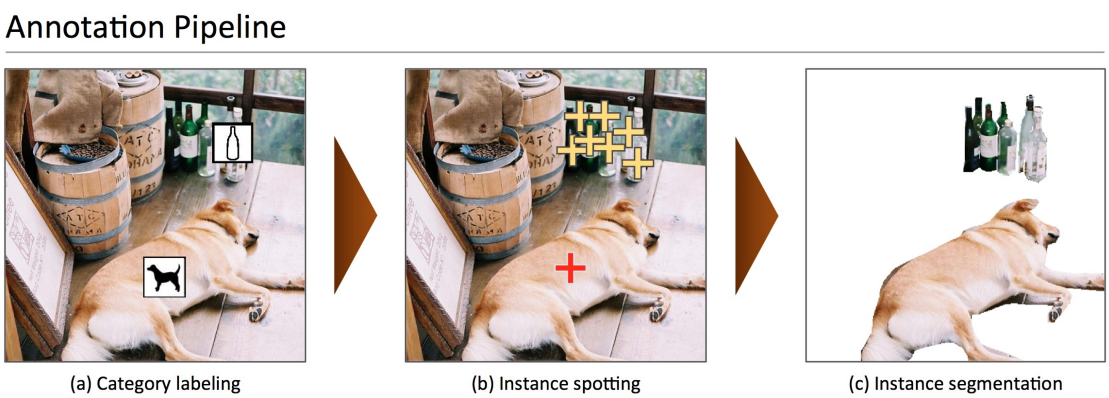


Figure 2.3: Our annotation pipeline is split into 3 primary tasks: (a) labeling the categories present in the image (§2.4.1), (b) locating and marking all instances of the labeled categories (§2.4.2), and (c) segmenting each object instance (§2.4.3).

2.3.1 Common Object Categories

The selection of object categories is a non-trivial exercise. The categories must form a representative set of all categories, be relevant to practical applications and occur with high enough frequency to enable the collection of a large dataset. Other important decisions are whether to include both "thing" and "stuff" categories [49] and whether fine-grained [121, 18] and object-part categories should be included. "Thing" categories include objects for which individual instances may be easily labeled (person, chair, car) where "stuff" categories include materials and objects with no clear boundaries (sky, street, grass). Since we are primarily interested in precise localization of object instances, we decided to only include "thing" categories and not "stuff." However, since "stuff" categories can provide significant contextual information, we believe the future labeling of "stuff" categories would be beneficial.

The specificity of object categories can vary significantly. For instance, a dog could be a member of the "mammal", "dog", or "German shepherd" categories. To enable the practical collection of a significant number of instances per cate-

gory, we chose to limit our dataset to entry-level categories, i.e. category labels that are commonly used by humans when describing objects (dog, chair, person). It is also possible that some object categories may be parts of other object categories. For instance, a face may be part of a person. We anticipate the inclusion of object-part categories (face, hands, wheels) would be beneficial for many real-world applications.

We used several sources to collect entry-level object categories of "things." We first compiled a list of categories by combining categories from PASCAL VOC [27] and a subset of the 1200 most frequently used words that denote visually identifiable objects [111]. To further augment our set of candidate categories, several children ranging in ages from 4 to 8 were asked to name every object they see in indoor and outdoor environments. The final 272 candidates may be found in Appendix B. Finally, the co-authors voted on a 1 to 5 scale for each category taking into account how commonly they occur, their usefulness for practical applications, and their diversity relative to other categories. The final selection of categories attempts to pick categories with high votes, while keeping the number of categories per super-category (animals, vehicles, furniture, etc.) balanced. Categories for which obtaining a large number of instances (greater than 5,000) was difficult were also removed. To ensure backwards compatibility all categories from PASCAL VOC [27] are also included. Our final list of 91 proposed categories is in Fig. 2.5(a).

2.3.2 Non-iconic Image Collection

Given the list of object categories, our next goal was to collect a set of candidate images. We may roughly group images into three types, Fig. 2.2: iconic-object images [7], iconic-scene images [122] and non-iconic images. Typical iconic-object images have a single large object in a canonical perspective centered in the image, Fig. 2.2(a). Iconic-scene images are shot from canonical viewpoints and commonly lack people, Fig. 2.2(b). Iconic images have the benefit that they may be easily found by directly searching for specific categories using Google or Bing image search. While iconic images generally provide high quality object instances, they can lack important contextual information and non-canonical viewpoints.

Our goal was to collect a dataset such that a majority of images are non-iconic, Fig. 2.2(c). It has been shown that datasets containing more non-iconic images are better at generalizing [116]. We collected non-iconic images using two strategies. First as popularized by PASCAL VOC [27], we collected images from Flickr which tends to have fewer iconic images. Flickr contains photos uploaded by amateur photographers with searchable metadata and keywords. Second, we did not search for object categories in isolation. A search for "dog" will tend to return iconic images of large, centered dogs. However, if we searched for pairwise combinations of object categories, such as "dog + car" we found many more non-iconic images. Surprisingly, these images typically do not just contain the two categories specified in the search, but numerous other categories as well. To further supplement our dataset we also searched for scene/object category pairs, see Appendix B. We downloaded at most 5 photos taken by a single photographer within a short time window. In the rare cases

in which enough images could not be found, we searched for single categories and performed an explicit filtering stage to remove iconic images. The result is a collection of 328,000 images with rich contextual relationships between objects as shown in Figs. 2.2(c) and 2.6.

2.4 Image Annotation

We next describe how we annotated our image collection. Due to our desire to label over 2.5 million object instances, the design of a cost efficient yet high quality annotation pipeline was critical. The annotation pipeline is outlined in Fig. 2.3. For all crowdsourcing tasks we used workers on Amazon's Mechanical Turk (AMT). Our user interfaces are described in detail in Appendix A. Note that, since the original version of this work [70], we have taken a number of steps to further improve the quality of the annotations. In particular, we have increased the number of annotators for the category labeling and instance spotting stages to eight. We also added a stage to verify the instance segmentations.

2.4.1 Category Labeling

The first task in annotating our dataset is determining which object categories are present in each image, Fig. 2.3(a). Since we have 91 categories and a large number of images, asking workers to answer 91 binary classification questions per image would be prohibitively expensive. Instead, we used a hierarchical approach [19].

We group the object categories into 11 super-categories (see Appendix B).

For a given image, a worker was presented with each group of categories in turn and asked to indicate whether any instances exist for that super-category. This greatly reduces the time needed to classify the various categories. For example, a worker may easily determine no animals are present in the image without having to specifically look for cats, dogs, etc. If a worker determines instances from the super-category (animal) are present, for each subordinate category (dog, cat, etc.) present, the worker must drag the category's icon onto the image over one instance of the category. The placement of these icons is critical for the following stage. We emphasize that only a single instance of each category needs to be annotated in this stage. To ensure high recall, 8 workers were asked to label each image. A category is considered present if any worker indicated the category; false positives are handled in subsequent stages. A detailed analysis of performance is presented in §2.4.4. This stage took ~20k worker hours to complete.

2.4.2 Instance Spotting

In the next stage all instances of the object categories in an image were labeled, Fig. 2.3(b). In the previous stage each worker labeled one instance of a category, but multiple object instances may exist. Therefore, for each image, a worker was asked to place a cross on top of each instance of a specific category found in the previous stage. To boost recall, the location of the instance found by a worker in the previous stage was shown to the current worker. Such priming helped workers quickly find an initial instance upon first seeing the image. The workers could also use a magnifying glass to find small instances. Each worker was asked to label at most 10 instances of a given category per image. Each

image was labeled by 8 workers for a total of ~10k worker hours.

2.4.3 Instance Segmentation

Our final stage is the laborious task of segmenting each object instance, Fig. 2.3(c). For this stage we modified the excellent user interface developed by Bell et al. [5] for image segmentation. Our interface asks the worker to segment an object instance specified by a worker in the previous stage. If other instances have already been segmented in the image, those segmentations are shown to the worker. A worker may also indicate there are no object instances of the given category in the image (implying a false positive label from the previous stage) or that all object instances are already segmented.

Segmenting 2,500,000 object instances is an extremely time consuming task requiring over 22 worker hours per 1,000 segmentations. To minimize cost we only had a single worker segment each instance. However, when first completing the task, most workers produced only coarse instance outlines. As a consequence, we required all workers to complete a training task for each object category. The training task required workers to segment an object instance. Workers could not complete the task until their segmentation adequately matched the ground truth. The use of a training task vastly improved the quality of the workers (approximately 1 in 3 workers passed the training stage) and resulting segmentations. Example segmentations may be viewed in Fig. 2.6.

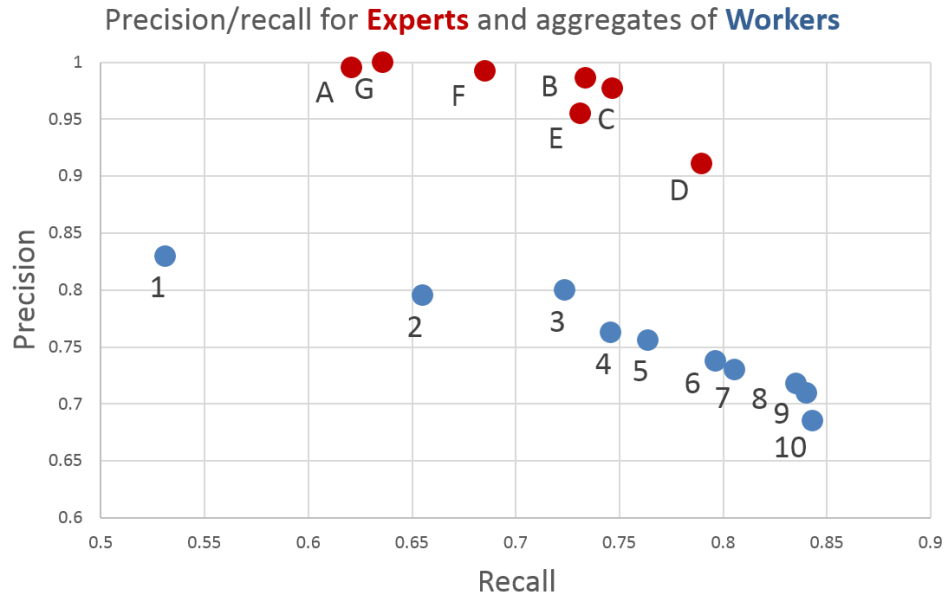
While the training task filtered out most bad workers, we also performed an explicit verification step on each segmented instance to ensure good quality. Multiple workers (3 to 5) were asked to judge each segmentation and indicate

whether it matched the instance well or not. Segmentations of insufficient quality were discarded and the corresponding instances added back to the pool of unsegmented objects. Finally, some approved workers consistently produced poor segmentations; all work obtained from such workers was discarded.

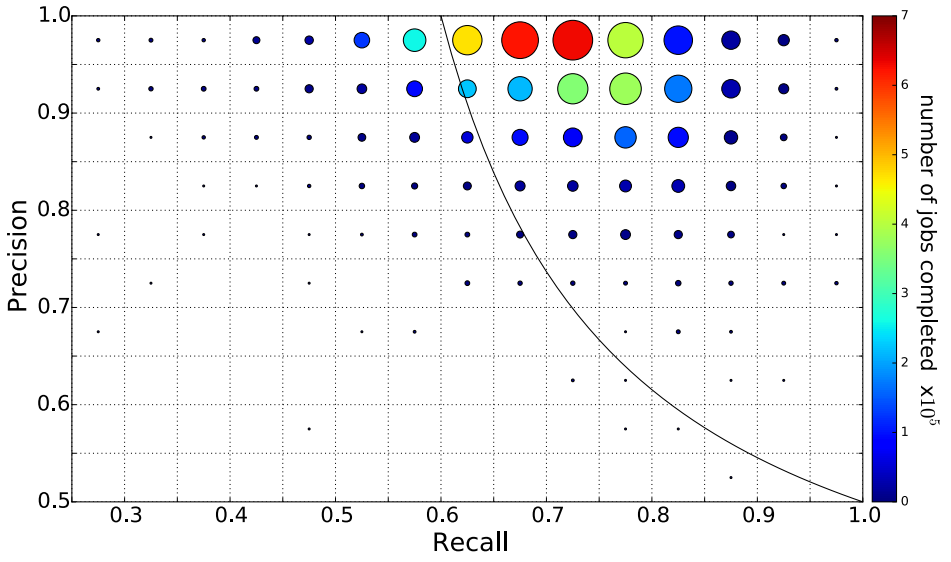
For images containing 10 object instances or fewer of a given category, every instance was individually segmented (note that in some images up to 15 instances were segmented). Occasionally the number of instances is drastically higher; for example, consider a dense crowd of people or a truckload of bananas. In such cases, many instances of the same category may be tightly grouped together and distinguishing individual instances is difficult. After 10-15 instances of a category were segmented in an image, the remaining instances were marked as "crowds" using a single (possibly multi-part) segment. For the purpose of evaluation, areas marked as crowds will be ignored and not affect a detector's score. Details are given in Appendix A.

2.4.4 Annotation Performance Analysis

We analyzed crowd worker quality on the category labeling task by comparing to dedicated expert workers, see Fig. 2.4(a). We compared precision and recall of seven expert workers (co-authors of the paper) with the results obtained by taking the union of one to ten AMT workers. Ground truth was computed using majority vote of the experts. For this task recall is of primary importance as false positives could be removed in later stages. Fig. 2.4(a) shows that the union of 8 AMT workers, the same number as was used to collect our labels, achieved greater recall than any of the expert workers. Note that worker recall saturates



(a)



(b)

Figure 2.4: Worker precision and recall for the category labeling task. (a) The union of multiple AMT workers (blue) has better recall than any expert (red). Ground truth was computed using majority vote of the experts. (b) Shows the number of workers (circle size) and average number of jobs per worker (circle color) for each precision/recall range. Most workers have high precision; such workers generally also complete more jobs. For this plot ground truth for each worker is the *union* of responses from all other AMT workers. See §2.4.4 for details.

at around 9-10 AMT workers.

Object category presence is often ambiguous. Indeed as Fig. 2.4(a) indicates, even dedicated experts often disagree on object presence, e.g. due to inherent ambiguity in the image or disagreement about category definitions. For any unambiguous examples having a probability of over 50% of being annotated, the probability all 8 annotators missing such a case is at most $.5^8 \approx .004$. Additionally, by observing how recall increased as we added annotators, we estimate that in practice over 99% of all object categories not later rejected as false positives are detected given 8 annotators. Note that a similar analysis may be done for instance spotting in which 8 annotators were also used.

Finally, Fig. 2.4(b) re-examines precision and recall of AMT workers on category labeling on a much larger set of images. The number of workers (circle size) and average number of jobs per worker (circle color) is shown for each precision/recall range. Unlike in Fig. 2.4(a), we used a leave-one-out evaluation procedure where a category was considered present if *any* of the remaining workers named the category. Therefore, overall worker precision is substantially higher. Workers who completed the most jobs also have the highest precision; all jobs from workers below the black line were rejected.

2.4.5 Caption Annotation

We added five written caption descriptions to each image in COCO. A full description of the caption statistics and how they were gathered will be provided shortly in a separate publication.

2.5 Dataset Statistics

Next, we analyze the properties of the Microsoft Common Objects in COntext (COCO) dataset in comparison to several other popular datasets. These include ImageNet [18], PASCAL VOC 2012 [27], and SUN [122]. Each of these datasets varies significantly in size, list of labeled categories and types of images. ImageNet was created to capture a large number of object categories, many of which are fine-grained. SUN focuses on labeling scene types and the objects that commonly occur in them. Finally, PASCAL VOC's primary application is object detection in natural images. COCO is designed for the detection and segmentation of objects occurring in their natural context.

The number of instances per category for all 91 categories is shown in Fig. 2.5(a). A summary of the datasets showing the number of object categories and the number of instances per category is shown in Fig. 2.5(d). While COCO has fewer categories than ImageNet and SUN, it has more instances per category which we hypothesize will be useful for learning complex models capable of precise localization. In comparison to PASCAL VOC, COCO has both more categories and instances.

An important property of our dataset is we strive to find non-iconic images containing objects in their natural context. The amount of contextual information present in an image can be estimated by examining the average number of object categories and instances per image, Fig. 2.5(b, c). For ImageNet we plot the object detection validation set, since the training data only has a single object labeled. On average our dataset contains 3.5 categories and 7.7 instances per image. In comparison ImageNet and PASCAL VOC both have less than 2

categories and 3 instances per image on average. Another interesting observation is only 10% of the images in COCO have only one category per image, in comparison, over 60% of images contain a single object category in ImageNet and PASCAL VOC. As expected, the SUN dataset has the most contextual information since it is scene-based and uses an unrestricted set of categories.

Finally, we analyze the average size of objects in the datasets. Generally smaller objects are harder to recognize and require more contextual reasoning to recognize. As shown in Fig. 2.5(e), the average sizes of objects is smaller for both COCO and SUN.

2.6 Dataset Splits

To accommodate a faster release schedule, we split the COCO dataset into two roughly equal parts. The first half of the dataset was released in 2014, the second half will be released in 2015. The 2014 release contains 82,783 training, 40,504 validation, and 40,775 testing images (approximately $\frac{1}{2}$ train, $\frac{1}{4}$ val, and $\frac{1}{4}$ test). There are nearly 270k segmented people and a total of 886k segmented object instances in the 2014 train+val data alone. The cumulative 2015 release will contain a total of 165,482 train, 81,208 val, and 81,434 test images. We took care to minimize the chance of near-duplicate images existing across splits by explicitly removing near duplicates (detected with [24]) and grouping images by photographer and date taken.

Following established protocol, annotations for train and validation data will be released, but not for test. We are currently finalizing the evaluation server for automatic evaluation on the test set. A full discussion of evaluation

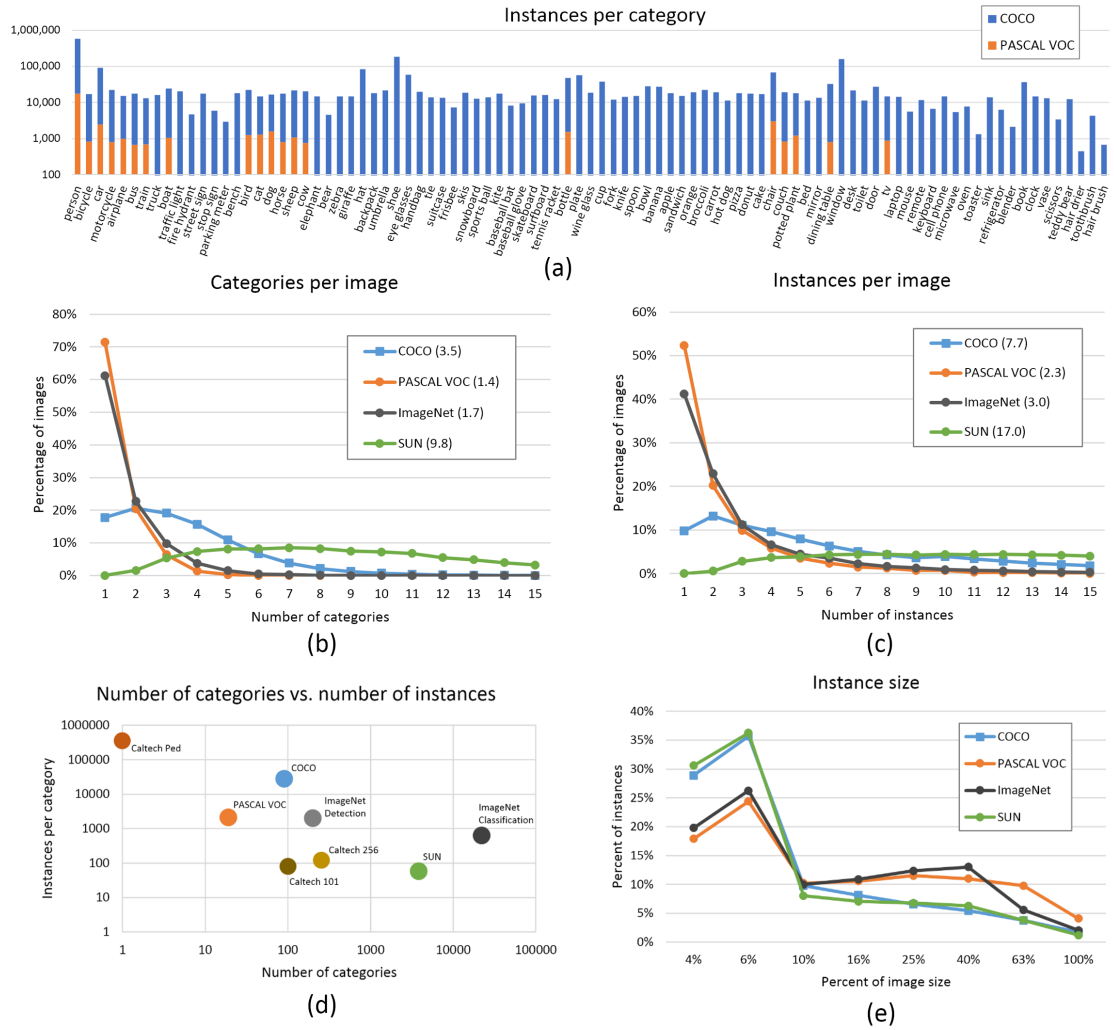


Figure 2.5: (a) Number of annotated instances per category for COCO and PASCAL VOC. (b,c) Number of annotated categories and annotated instances, respectively, per image for COCO, ImageNet Detection, PASCAL VOC and SUN (average number of categories and instances are shown in parentheses). (d) Number of categories vs. the number of instances per category for a number of popular object recognition datasets. (e) The distribution of instance sizes for the COCO, ImageNet Detection, PASCAL VOC and SUN datasets.

	plane	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	moto	person	plant	sheep	sofa	train	tv	avg.
DPMv5-P	**45.6**	49.0	11.0	**11.6**	**27.2**	50.5	**43.1**	**23.6**	**17.2**	23.2	**10.7**	**20.5**	42.5	**44.5**	**41.3**	**8.7**	**29.0**	**18.7**	**40.0**	34.5	**29.6**
DPMv5-C	43.7	**50.1**	**11.8**	2.4	21.4	**60.1**	35.6	16.0	11.4	**24.8**	5.3	9.4	**44.5**	41.0	35.8	6.3	28.3	13.3	38.8	**36.2**	26.8
DPMv5-P	35.1	17.9	3.7	2.3	**7**	45.4	**18.3**	8.6	**6.3**	17	4.8	**5.8**	35.3	25.4	**17.5**	4.1	**14.5**	9.6	31.7	27.9	16.9
DPMv5-C	**36.9**	**20.2**	**5.7**	**3.5**	6.6	**50.3**	16.1	**12.8**	4.5	**19.0**	**9.6**	4.0	**38.2**	**29.9**	15.9	**6.7**	13.8	**10.4**	**39.2**	**37.9**	**19.1**

Table 2.1: **Top**: Detection performance evaluated on **PASCAL VOC 2012**. DPMv5-P is the performance reported by Girshick et al. in VOC release 5. DPMv5-C uses the same implementation, but is trained with COCO. **Bottom**: Performance evaluated on **COCO** for DPM models trained with PASCAL VOC 2012 (DPMv5-P) and COCO (DPMv5-C). For DPMv5-C we used 5000 positive and 10000 negative training examples. While COCO is considerably more challenging than PASCAL, use of more training data coupled with more sophisticated approaches [61, 39, 102] should improve performance substantially.

metrics will be added once the evaluation server is complete.

Note that we have limited the 2014 release to a subset of 80 categories. We did not collect segmentations for the following 11 categories: hat, shoe, eyeglasses (too many instances), mirror, window, door, street sign (ambiguous and difficult to label), plate, desk (due to confusion with bowl and dining table, respectively) and blender, hair brush (too few instances). We may add segmentations for some of these categories in the cumulative 2015 release.

2.7 Algorithmic Analysis

Bounding-box detection For the following experiments we take a subset of 55,000 images from our dataset[1] and obtain tight-fitting bounding boxes from the annotated segmentation masks. We evaluate models tested on both COCO and PASCAL, see Table 2.1. We evaluate two different models. **DPMv5-P**: the

[1]These preliminary experiments were performed before our final split of the dataset intro train, val, and test. Baselines on the actual test set will be added once the evaluation server is complete.

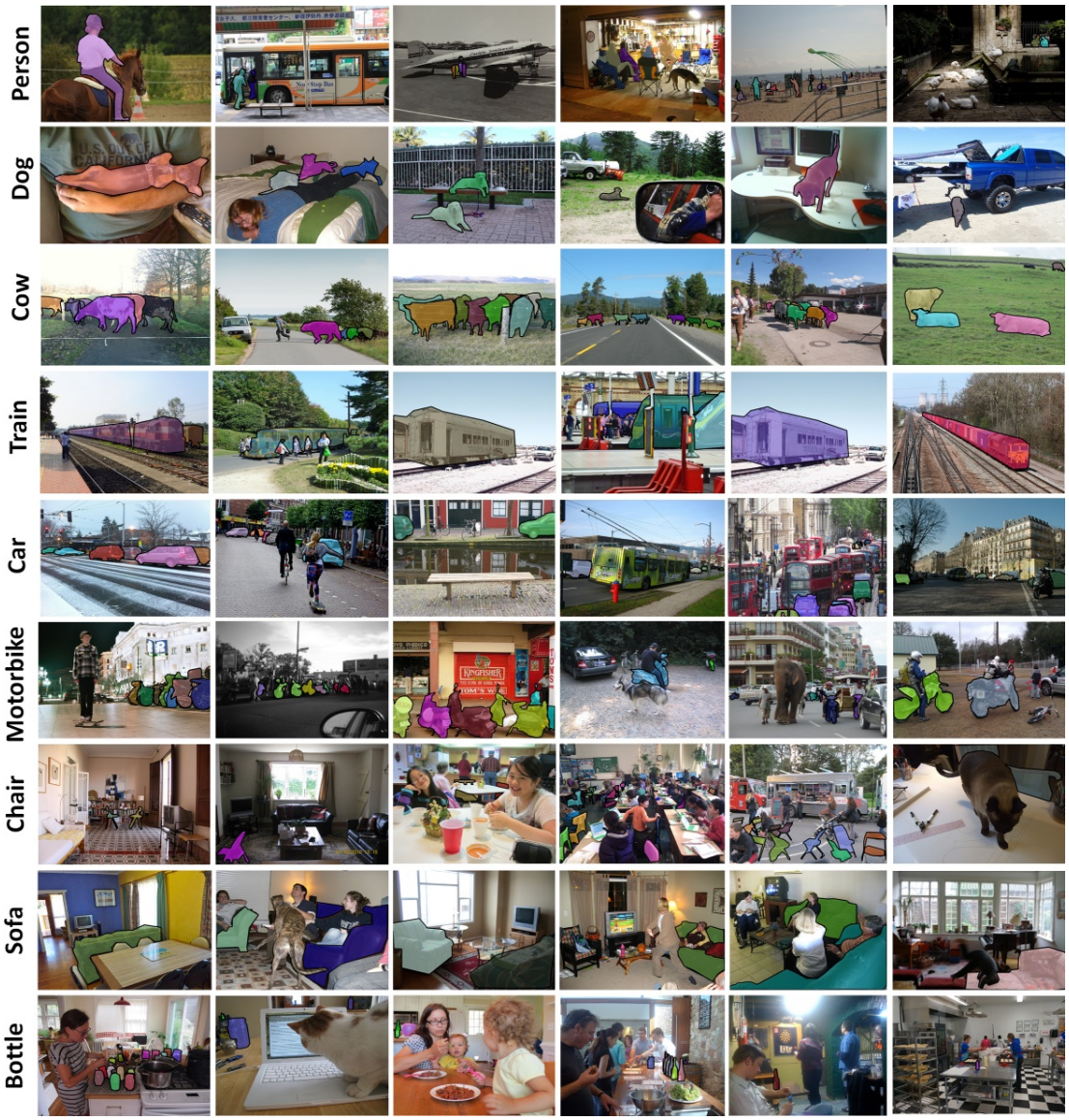


Figure 2.6: Samples of annotated images in the COCO dataset.

latest implementation of [33] (release 5 [40]) trained on PASCAL VOC 2012. **DPMv5-C**: the same implementation trained on COCO (5000 positive and 10000 negative images). We use the default parameter settings for training COCO models.

If we compare the average performance of DPMv5-P on PASCAL VOC and COCO, we find that average performance on COCO drops by nearly a *factor*

of 2, suggesting that COCO does include more difficult (non-iconic) images of objects that are partially occluded, amid clutter, etc. We notice a similar drop in performance for the model trained on COCO (DPMv5-C).

The effect on detection performance of training on PASCAL VOC or COCO may be analyzed by comparing DPMv5-P and DPMv5-C. They use the same implementation with different sources of training data. Table 2.1 shows DPMv5-C still outperforms DPMv5-P in 6 out of 20 categories when testing on PASCAL VOC. In some categories (e.g., dog, cat, people), models trained on COCO perform worse, while on others (e.g., bus, tv, horse), models trained on our data are better.

Consistent with past observations [130], we find that including difficult (non-iconic) images during training may not always help. Such examples may act as noise and pollute the learned model if the model is not rich enough to capture such appearance variability. Our dataset allows for the exploration of such issues.

Torralba and Efros [116] proposed a metric to measure cross-dataset generalization which computes the 'performance drop' for models that train on one dataset and test on another. The performance difference of the DPMv5-P models across the two datasets is 12.7 AP while the DPMv5-C models only have 7.7 AP difference. Moreover, overall performance is much lower on COCO. These observations support two hypotheses: 1) COCO is significantly more difficult than PASCAL VOC and 2) models trained on COCO can generalize better to easier datasets such as PASCAL VOC given more training data. To gain insight into the differences between the datasets, see Appendix B for visualizations of person and chair examples from the two datasets.

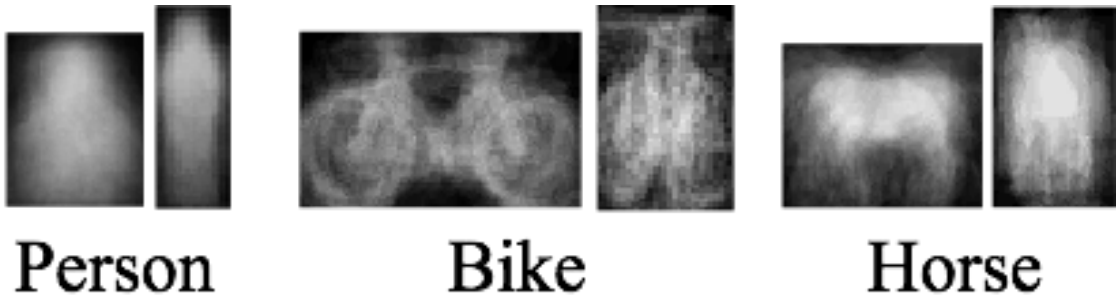


Figure 2.7: We visualize our mixture-specific shape masks. We paste thresholded shape masks on each candidate detection to generate candidate segments.

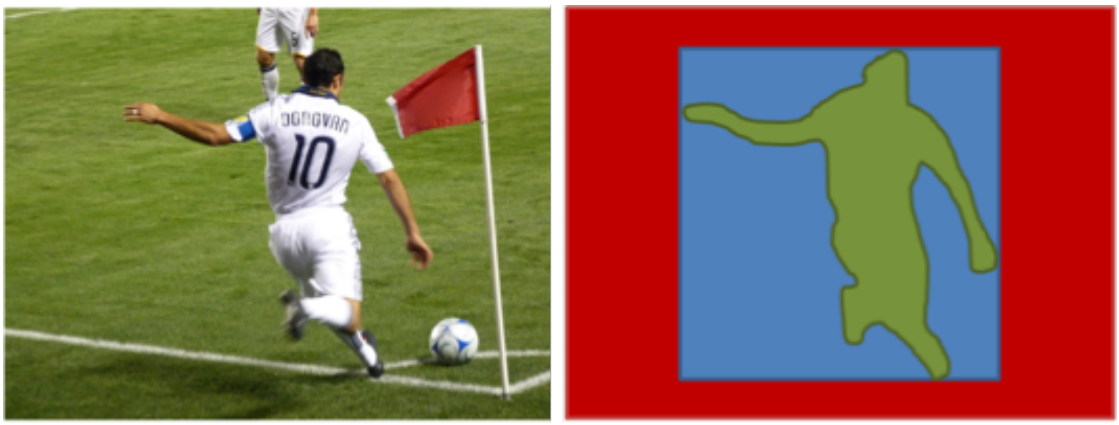


Figure 2.8: Evaluating instance detections with segmentation masks versus bounding boxes. Bounding boxes are a particularly crude approximation for articulated objects; in this case, the majority of the pixels in the (**blue**) tight-fitting bounding-box do not lie on the object. Our (**green**) instance-level segmentation masks allows for a more accurate measure of object detection and localization.

Generating segmentations from detections We now describe a simple method for generating object bounding boxes and segmentation masks, following prior work that produces segmentations from object detections [10, 124, 87, 16]. We learn aspect-specific pixel-level segmentation masks for different categories. These are readily learned by averaging together segmentation masks from aligned training instances. We learn different masks corresponding to the different mixtures in our DPM detector. Sample masks are visualized in Fig. 4.4.

Detection evaluated by segmentation Segmentation is a challenging task

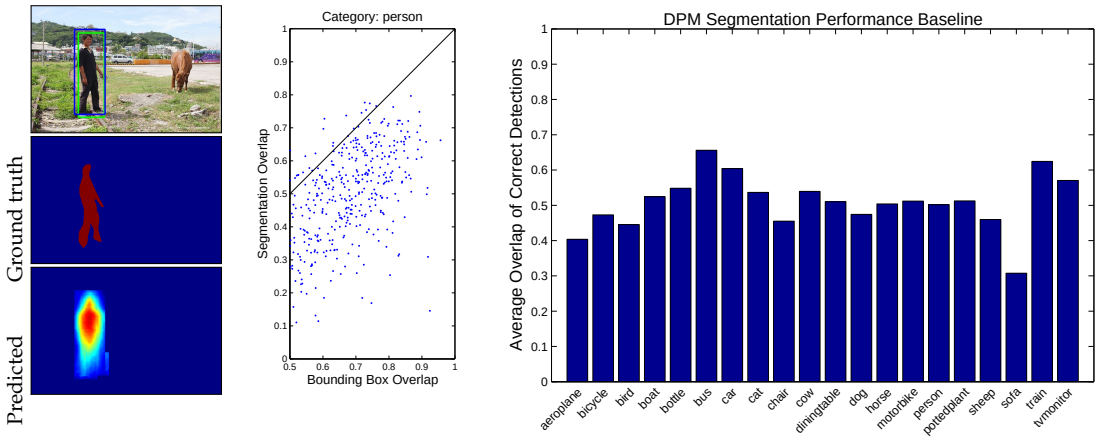


Figure 2.9: A predicted segmentation might not recover object detail even though detection and ground truth bounding boxes overlap well (left). Sampling from the person category illustrates that predicting segmentations from top-down projection of DPM part masks is difficult even for correct detections (center). Average segmentation overlap measured on COCO for the 20 PASCAL VOC categories demonstrates the difficulty of the problem (right).

even assuming a detector reports correct results as it requires fine localization of object part boundaries. To decouple segmentation evaluation from detection correctness, we benchmark segmentation quality using only correct detections. Specifically, given that the detector reports a correct bounding box, how well does the predicted segmentation of that object match the ground truth segmentation? As criterion for correct detection, we impose the standard requirement that intersection over union between predicted and ground truth boxes is at least 0.5. We then measure the intersection over union of the predicted and ground truth segmentation masks, see Fig. 2.8. To establish a baseline for our dataset, we project learned DPM part masks onto the image to create segmentation masks. Fig. 2.9 shows results of this segmentation baseline for the DPM learned on the 20 PASCAL categories and tested on our dataset.

2.8 Discussion

We introduced a new dataset for detecting and segmenting objects found in everyday life in their natural environments. Utilizing over 70,000 worker hours, a vast collection of object instances was gathered, annotated and organized to drive the advancement of object detection and segmentation algorithms. Emphasis was placed on finding non-iconic images of objects in natural environments and varied viewpoints. Dataset statistics indicate the images contain rich contextual information with many objects present per image.

There are several promising directions for future annotations on our dataset. We currently only label "things", but labeling "stuff" may also provide significant contextual information that may be useful for detection. Many object detection algorithms benefit from additional annotations, such as the amount an instance is occluded [20] or the location of keypoints on the object [8]. Finally, our dataset could provide a good benchmark for other types of labels, including scene types [122], attributes [82, 29] and full sentence written descriptions [88]. We are actively exploring adding various such annotations.

To download and learn more about COCO please see the project website[2]. COCO will evolve and grow over time; up to date information is available online.

[2] http://mscoco.org/

CHAPTER 3

LEARNING TO REFINE INSTANCE SEGMENTATION

3.1 Introduction

As object detection [33, 102, 114, 47, 39, 38, 91, 6] has rapidly progressed, there has been a renewed interest in object instance segmentation [66]. As the name implies, the goal is to both detect and segment each individual object. The task is related to both object detection with bounding boxes [66, 27, 18] and semantic segmentation [105, 27, 28, 83, 25, 128, 12, 100, 77]. It involves challenges from both domains, requiring accurate pixel-level object segmentation coupled with identification of each individual object instance.

A number of recent papers have explored the use convolutional neural networks (CNNs) [63] for object instance segmentation [43, 84, 14, 44]. Standard feedforward CNNs [61, 110, 113, 48] interleave convolutional layers (with pointwise nonlinearities) and pooling layers. Pooling controls model capacity and increases receptive field size, resulting in a coarse, highly-semantic feature representation. While effective and necessary for extracting object-level information, this general architecture results in low resolution features that are invariant to pixel-level variations. This is beneficial for classification and identifying object instances but poses challenge for pixel-labeling tasks. Hence, CNNs that utilize only upper network layers for object instance segmentation [43, 84, 14], as in Figure 3.1a, can effectively generate coarse object masks but have difficulty generating pixel-accurate segmentations.

For pixel-labeling tasks such as semantic segmentation and edge detection,

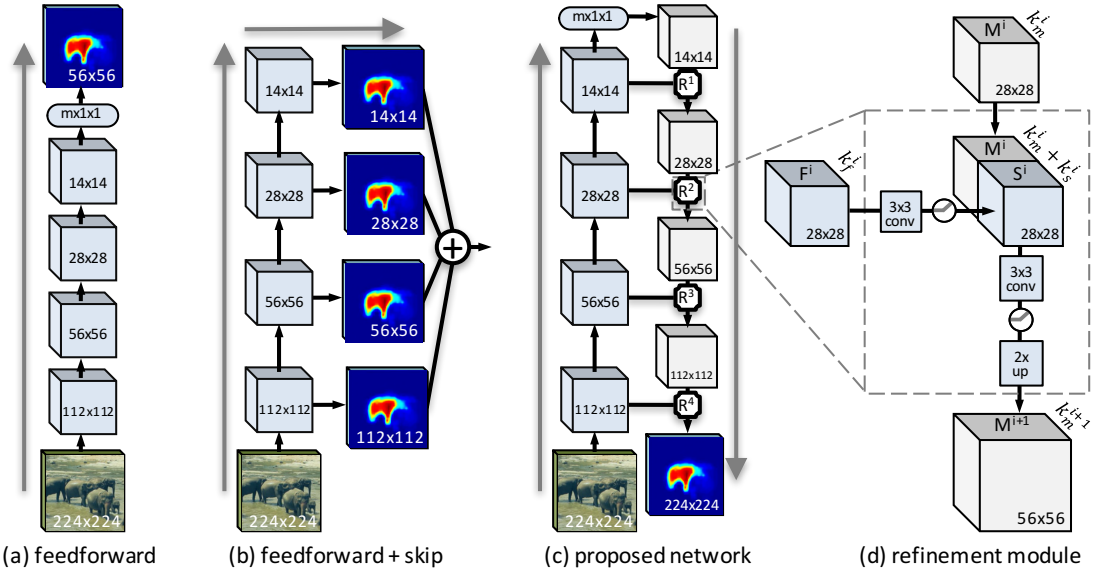


Figure 3.1: Architectures for object instance segmentation. (a) Feedforward nets, such as DeepMask [84], predict masks using only upper-layer CNN features, resulting in coarse pixel masks. (b) Common 'skip' architectures are equivalent to making independent predictions from each layer and averaging the results [73, 44, 123], such an approach is not well suited for object instance segmentation. (c,d) In this work we propose to augment feedforward nets with a novel top-down refinement approach. The resulting bottom-up/top-down architecture is capable of efficiently generating high-fidelity object masks.

'skip' connections [103, 73, 44, 123], as shown in Figure 3.1b, are popular. In practice, common skip architectures are equivalent to making independent predictions from each network layer and upsampling and averaging the results (see Fig. 2 in [44], Fig. 3 in [73], and Fig. 3 in [123]). This is effective for semantic segmentation as local receptive fields in early layers can provide sufficient data for pixel labeling. For object segmentation, however, it is necessary to differentiate between object instances, for which local receptive fields are insufficient (e.g. local patches of sheep fur can be labeled as such but without object-level information it can be difficult to determine if they belong to the same animal).

In this paper, we propose a novel CNN which efficiently merges the spatially rich information from low-level features with the high-level object knowledge

encoded in upper network layers. Rather than generating independent outputs from multiple network layers, our approach first generates a coarse *mask encoding* in a feedforward manner, which is simply a semantically meaningful feature map with multiple channels, then refines it by successively integrating information from earlier layers. Specifically, we introduce a *refinement module* and stack successive such modules together into a top-down refinement process. See Figures 3.1c and 3.1d. Each refinement module is responsible for 'inverting' the effect of pooling by taking a mask encoding generated in the top-down pass, along with the matching features from the bottom-up pass, and merging the information in both to generate a new mask encoding with double the spatial resolution. The process continues until full resolution is restored and the final output encodes the object mask. The refinement module is efficient and fully backpropable.

We apply our approach in the context of object proposal generation [2, 118, 131, 86, 58, 56, 53]. The seminal object detection work on R-CNN [39] follows a two-phase approach: first, an object proposal algorithm is used to find regions in images that may contain objects; second, a CNN assigns each proposal a category label. While originally object proposals were constructed from low-level grouping and saliency cues [53], recently CNNs have been adopted for this task [114, 91, 84], leading to massive improvements in detection accuracy. In particular, Pinheiro et al. [84] demonstrated how to adopt a CNN to generate rich object instance segmentations in an image. The proposed model, called DeepMask, predicts how likely an image patch is to fully contain a centered object and also outputs an associated segmentation mask for the object, if present. The model is run convolutionally to generate a dense set of object proposals for an image. DeepMask outperforms previous object segment proposal methods

by a substantial margin [84].

In this work we utilize the DeepMask architecture as our starting point for object instance segmentation due to its simplicity and effectiveness. We augment the basic DeepMask architecture with our refinement module (see Figure 3.1) and refer to the resulting approach as *SharpMask* to emphasize its ability to produce sharper, higher-fidelity object segmentation masks. In addition to the top-down refinement, we also revisit the basic bottom-up architecture of the DeepMask network and likewise optimize it for the segmentation task.

SharpMask improves segmentation mask quality relative to DeepMask. For object proposal generation, average recall on the COCO dataset [66] improves 10-20% and establishes the new state-of-the-art on this task. Moreover, we optimize our core architecture and improve speed by 50% with respect to DeepMask, with an average of .76s per image. Our fast model, which still outperforms previous results, runs at .46s, or, by using additional image scales, we can boost small object recall by ~2×. Finally we show SharpMask proposals substantially improve object detection results when coupled with the Fast R-CNN detector [38].

The paper is organized as follows: §3.2 presents related work, §3.3 introduces our novel top-down refinement network, §3.4 describes optimizations to the network architecture, and finally §3.5 validates our approach experimentally.

All source code for reproducing the methods in this paper will be released.

3.2 Related Work

Following their success in image classification [61, 110, 113, 48], CNNs have been adopted with great effect to pixel-labeling tasks such as depth estimation [25], optical flow [23], and semantic segmentation [28]. Below we describe architectural innovations for such tasks, and discuss how they relate to our approach. Aside from skip connections [103, 44, 73, 123], which were discussed in §3.1, these techniques can be roughly classified as multiscale architectures, deconvolutional networks, and graphical model networks. We discuss each in turn next. We emphasize, however, that most of these approaches are not applicable to our domain due to severe computational constraints: we must refine hundreds of proposals per image implying the marginal time per proposal must be minimal.

Multiscale architectures: [28, 25, 83] compute features over multiple rescaled versions of an image. Features can be computed independently at each scale [28], or the output from one scale can be used as additional input to the next finer scale [25, 83]. Our approach relies on similar intuition but does not require recomputing features at each image scale. This allows us to apply refinement efficiently to hundreds of locations per image as necessary for object proposal generation.

Deconvolutional networks: [127] proposed to invert the pooling process in a CNN to generate progressively higher resolution input images by storing the 'switch' variables from the pooling operation. Deconv networks have recently been applied successfully to semantic segmentation [77]. Deconv layers share similarities with our refinement module, however, 'switches' are commu-

nicated instead of the feature values, which limits the information that can be transferred. Finally, [23] proposed to progressively increase the resolution of an optical flow map. This can be seen as a special case of our refinement approach where: (1) the 'features' for refinement are set to be the flow field itself, (2) no feature transform is applied to the bottom-up features, and (3) the approach is applied monolithically to the entire image. Restricting our method in any of these ways would cause it to fail in our setting as discussed in §3.5.

Graphical model networks: a number of recent papers have proposed integrating graphical models into CNNs by demonstrating they can be formulated as recurrent nets [128, 12, 100]. Good results were demonstrated on semantic segmentation. While too slow to apply to multiple proposals per image, these approaches likewise attempt to sharpen a coarse segmentation mask.

3.3 Learning Mask Refinement

We apply our proposed bottom-up/top-down refinement architecture to object instance segmentation. Specifically, we focus on object proposal generation [53], which forms the cornerstone of modern object detection [39]. We note that although we test the proposed refinement architecture on the task of object segmentation, it could potentially be applied to other pixel-labeling tasks.

Object proposal algorithms aim to find diverse regions in an image which are likely to contain objects; both proposal recall and quality correlate strongly with detector performance [53]. We adopt the DeepMask network [84] as the starting point for proposal generation. DeepMask is trained to jointly generate a class-agnostic object mask and an associated 'objectness' score for each input

image patch. At inference time, the model is run convolutionally to generate a dense set of scored segmentation proposals. We refer readers to [84] for full details.

A simplified diagram of the segmentation branch of DeepMask is illustrated in Figure 3.1a. The network is trained to infer the mask for the object located in the center of the input patch. It contains a series of convolutional layers interleaved with pooling stages that reduce the spatial dimensions of the feature maps, followed by a fully connected layer to generate the object mask. Hence, each pixel prediction is based on a complete view of the object, however, its input feature resolution is low due to the multiple pooling stages.

As a result, DeepMask generates masks that are accurate on the object level but only coarsely align with object boundaries, see Figure 3.2a. In order to obtain higher-quality masks, we augment the basic DeepMask architecture with our refinement approach. We refer to the resulting method as *SharpMask* to emphasize its ability to produce sharper, pixel-accurate object masks, see Figure 3.2b. We begin with a high-level overview of our approach followed by further details.

3.3.1 Refinement Overview

Our goal is to efficiently merge the spatially rich information from low-level features with the high-level semantic information encoded in upper network layers. Three principles guide our approach: (1) object-level information is often necessary to segment an object, (2) given object-level information, segmentation should proceed in a top-down fashion, successively integrating informa-

(a) DeepMask Output (b) SharpMask Output

Figure 3.2: Qualitative comparison of DeepMask versus SharpMask segmentations. Proposals with highest IoU to the ground truth are shown for each method. Both DeepMask and SharpMask generate object masks that capture the general shape of the objects. However, SharpMask improves the masks near object boundaries.

tion from earlier layers, and (3) the approach should invert the loss of resolution from pooling (with the final output matching the resolution of the input).

To satisfy these principles, we augment standard feedforward nets with a top-down refinement process. An overview of our approach is shown in Figure 3.1c. We introduce a 'refinement module' R that is responsible for inverting the effect of pooling and doubling the resolution of the input mask encoding. Each module R^i takes as input a mask encoding M^i generated in the top-down pass, along with matching features F^i generated in the bottom-up pass, and learns to merge the information to generate a new upsampled object encoding M^{i+1}. In other words: $M^{i+1} = R^i(M^i, F^i)$, see Figure 3.1d. Multiple such modules are stacked (one module per pooling layer). The final output of our network is a pixel labeling of the same resolution as the input image. We present full details next.

3.3.2 Refinement Details

The feedforward pathway of our network outputs a 'mask encoding' M^1, or simply, a low-resolution but semantically meaningful feature map with k_m^1 channels. M^1 serves as the input to the top-down refinement module, which is responsible for progressively increasing the mask encoding's resolution. Note that using $k_m^1 > 1$ allows the mask encoding to capture more information than a simple segmentation mask, which proves to be key for obtaining good accuracy.

Each refinement module R^i aggregates information from a coarse mask encoding M^i and features F^i from the corresponding layer of the bottom-up computation (we always use the last convolutional layer prior to pooling). By construction, M^i and F^i have the same spatial dimensions; the goal of R^i is to generate a new mask encoding M^{i+1} with double spatial resolution based on inputs M^i and F^i. We denote this via $M^{i+1} = R^i(M^i, F^i)$. This process is applied iteratively n times (where n is the number of pooling stages) until the feature map has the same dimensions as the input image patch. Each module R^i has separate parameters, allowing the network to learn stage-specific refinements.

The refinement module aims to enhance the mask encoding M^i using features F^i. As M^i and F^i have the same spatial dimensions, one option is to first simply concatenate M^i and F^i. However, directly concatenating F^i with M^i poses two challenges. Let k_m^i and k_f^i be the number of channels in M^i and F^i respectively. Typically, k_f^i can be quite large in modern CNNs, so using F^i directly would be computationally expensive. Second, typically $k_f^i \gg k_m^i$, so directly concatenating the features maps risks drowning out the signal in M^i.

Instead, we opt to first reduce the number of channels k_f^i (but preserving the

spatial dimensions) of these features through a 3×3 convolutional module (plus ReLU), generating 'skip' features S^i, with $k_s^i \ll k_f^i$ channels. This substantially reduces computational requirements, moreover, it allows the network to transform F^i into a form S^i more suitable for use in refinement. An important but subtle point is that during full image inference, as with the features F^i, skip features are shared by overlapping image patches, making them highly efficient to compute. In contrast, the remaining computations of R^i are patch dependent as they depend on the local mask M^i and hence cannot be shared across locations.

The refinement module concatenates mask encoding M^i with skip features S^i resulting in a feature map with $k_m^i + k_s^i$ channels, and applies another 3×3 convolution (plus ReLU) to the result. Finally, the output is upsampled using bilinear upsampling by a factor of 2, resulting in a new mask encoding M^{i+1} with k_m^{i+1} channels (k_m^{i+1} is determined by the number of 3×3 kernels used for the convolution). As with the convolution for generating the skip features, this transformation is used to simultaneously learn a nonlinear mask encoding from the concatenated features and to control the capacity of the model. Please see Figure 3.1d for a complete overview of the refinement module R. Further optimizations to R are possible, for details see Figure 3.7.

Note that the refinement module uses only convolution, ReLU, bilinear upsampling, and concatenation, hence it is fully backpropable and highly efficient. In §3.5.2, we analyze different architecture choices for the refinement module in terms of performance and speed. As a general design principle, we aim to keep k_s^i and k_m^i large enough to capture rich information but small enough to keep computation low. In particular, we can start with a fairly large number of channels but as spatial resolution is increased the number of channels should

decrease. This reverses the typical design of feedforward networks where spatial resolution decreases while the number of channels increases with increasing depth.

3.3.3 Training and Inference

We train SharpMask with an identical data definition and loss function as the original DeepMask model. Each training sample is a triplet containing an input patch, a label specifying if the input patch contains a centered object at the correct scale, and for positive samples a binary object mask. The network trunk parameters are initialized with a network that was pre-trained on ImageNet [18]. All the other layers are initialized randomly from a uniform distribution.

Training proceeds in two stages: first, the model is trained to jointly infer a coarse pixel-wise segmentation mask and an object score, second, the feedforward path is 'frozen' and the refinement modules trained. The first training stage is identical to [84]. Once learning of the first stage converges, the final mask prediction layer of the feedforward network is removed and replaced with a linear layer that generates a mask encoding M^1 in place of the actual mask output. We then add the refinement modules to the network and train using standard stochastic gradient descent, backpropagating the error only on the horizontal and vertical convolution layers on each of the n refinement modules.

This two-stage training procedure was selected for three reasons. First, we found it led to faster convergence. Second, at inference time, a *single* network trained in this manner can be used to generate either a coarse mask using the forward path only or a sharp mask using our bottom-up/top-down approach.

Third, we found the gains of fine-tuning through the entire network to be minimal once the forward branch had converged.

During full-image inference, similarly to [84], most computation for neighboring windows is shared through use of convolution, including for skip layers S^i. However, as discussed, the refinement modules receive a unique input M^1 at each spatial location, hence, computation proceeds independently at each location for this stage. Rather than refine every proposal, we simply refine only the most promising locations. Specifically, we select the top N scoring proposal windows and apply the refinement in a batch mode to these top N locations.

To further clarify all implementation details, full source code will be released.

3.4 Feedforward Architecture

While the focus of our work is on top-down mask refinement, to obtain a better understanding of object segmentation we also explore factors that effect a feedforward network's ability to generate accurate object masks. In the next two subsections we carefully examine the design of the network 'trunk' and 'head'.

3.4.1 Trunk Architecture

We begin by identifying model bottlenecks. DeepMask spends 40% of its time for feature extraction, 40% for mask prediction, and 20% for score prediction. Given the time of feature extraction, increasing model depth or breadth can in-

cur a non-trivial computational cost. Simply upgrading the 11-layer VGG-A model [110] used in [84] to the 16-layer VGG-D model can double run time. Recently He et al. [48] introduced Residual Networks (ResNet) and showed excellent results. In this work, we use the 50-layer ResNet model pre-trained on ImageNet, which achieves the accuracy of VGG-D but with the inference time of VGG-A.

We explore models with varying input size **W**, number of pooling layers **P**, stride density **S**, model depth **D**, and final number of features channels **F**. These factors are intertwined but we can achieve significant insight by a targeted study.

Input size W: Given a minimum object size O, the input image needs to be upsampled by W/O to detect small objects. Hence, reducing W improves speed of both mask prediction and inference for small objects. However, smaller W reduces the input resolution which in turn lowers the accuracy of mask prediction. Moreover, reducing W decreases stride density S which further harms accuracy.

Pooling layers P: Assuming 2×2 pooling, the final kernel width is $W/2^P$. During inference, this necessitates convolving with a large $W/2^P$ kernel in order to aggregate information (e.g., 14×14 for DeepMask). However, while more pooling P results in faster computation, it also results in loss of feature resolution.

Stride density S: We define the stride density to be S=W/stride (where typically stride is 2^P). The smaller the stride, the denser the overlap with ground truth locations. We found that the stride density is key for mask prediction. Doubling the stride while keeping W constant greatly reduces performance as

the model must be more spatially invariant relative to a fixed object size.

Depth D: For typical networks [61, 110, 113, 48], spatial resolution decreases with increasing D while the number of features channels F increases. In the context of instance segmentation, reducing spatial resolution hurts performance. One possible direction is to start with lower layers that have less pooling and increase the depth of the model without reducing spatial resolution or increasing F. This would require training networks from scratch which we leave to future work.

Feature channels F: The high dimensional features at the top layer introduce a bottleneck for feature aggregation. An efficient approach is to first apply dimensionality reduction before feature aggregation. We adopt 1×1 convolution to reduce F and show that we can achieve large speedups in this manner.

In §3.5.1 and Table 3.1 we examine various choices for W, P, S, D, and F.

3.4.2 Head Architecture

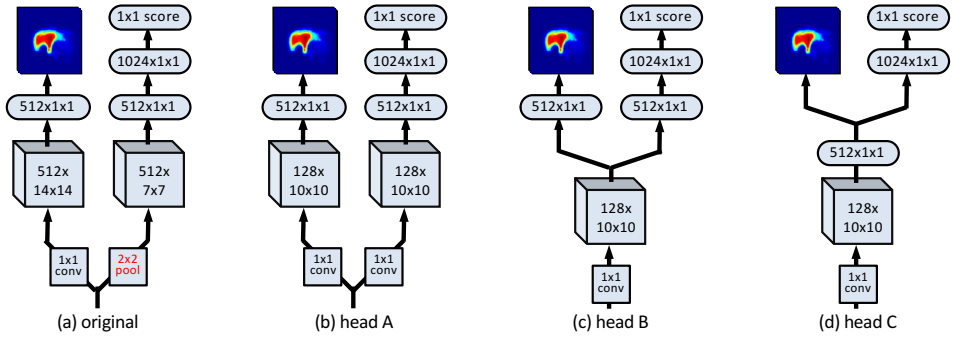


Figure 3.3: Network head architecture. (a) The original DeepMask head. (b-d) Various head options with increasing simplicity and speed. The heads share identical pathways for mask prediction but have progressively simplified score branches.

We also examine the 'head' of the DeepMask model, focusing on score prediction. Our goal is to simplify the head and further improve inference speed.

In DeepMask, the mask and scoring heads branch after the final $512 \times 14 \times 14$ feature map (see Figure 3.3a). Both mask and score prediction require a large convolution, and in addition, the score branch requires an extra pooling step and hence interleaving to match the stride of the mask network during inference. Overall, this leads to a fairly inelegant and slow inference procedure.

We propose a sequence of simplified network structures that have identical mask branches but that share progressively more computation. A series of model heads A-C is detailed in Figure 3.3. Head A removes the need for interleaving in DeepMask by removing max pooling and replacing the $512 \times 7 \times 7$ convolutions by $128 \times 10 \times 10$ convolutions; overall this network is much faster. Head B simplifies this by having the $128 \times 10 \times 10$ features shared by both the mask and score branch. Finally, model C further reduces computation by having the score prediction utilize the same low rank $512 \times 1 \times 1$ features used for the mask.

In §3.5.1 we evaluate these variants in terms of performance and speed.

3.5 Experiments

We train our model on the training set of the COCO dataset [66], which contains 80k training images and 500k instance annotations. For most of our experiments, results are reported on the first 5k COCO validation images. Mask accuracy is measured by Intersection over Union (IoU) which is the ratio of the

Figure 3.4: SharpMask proposals with highest IoU to the ground truth on selected COCO images. Missed objects (no matching proposals with IoU > 0.5) are marked in red. The last row shows a number of failure cases.

intersection of the predicted mask and ground truth annotation to their union. A common method for summarizing object proposal accuracy is using the average recall (AR) between IoU 0.5 and .95 for a fixed number of proposals. Hosang et al. [53] show that AR correlates well with object detector performance.

Our results are measured in terms of AR at 10, 100, and 1000 proposals and

averaged across all counts (AUC). As the COCO dataset contains objects in a wide range of scales, it is also common practice to divide objects into roughly equally sized sets according to object pixel area a: small ($a < 32^2$), medium ($32^2 \leq a \leq 96^2$), and large ($a > 96^2$) objects, and report accuracy at each scale.

We use a different subset of the COCO validation set to decide architecture choices and hyper-parameter selection. We use a learning rate of 1e-3 for training the refinement stage, which takes about 2 days to train on an Nvidia Tesla K40m GPU. To mitigate the mismatch of per-patch training with convolutional inference, we found that training deeper model such as ResNet requires adding extra image content (32 pixels) surrounding the training patches and using reflective-padding instead of 0-padding at every convolutional layer. Finally, following [84], we binarize our continuous mask prediction using a threshold of 0.2.

	W	P	D	S	kernel	F	AR	AR^S	AR^M	AR^L	time
DeepMask	224	4	8	14	512x14x14	512	36.6	18.2	48.7	50.6	1.32s
W160-P4-D8-VGG	160	4	8	10	1024x10x10	512	35.5	15.1	47.5	53.2	.58s
W160-P4-D39	160	4	39	10	1024x10x10	512	37.0	15.9	50.5	53.9	.58s
W160-P4-D39-F128	160	4	39	10	1024x10x10	128	36.9	15.6	49.9	54.8	.45s
W112-P4-D39	112	4	39	7	1024x7x7	512	30.8	11.2	42.3	47.8	.31s
W112-P3-D21	112	3	21	14	512x14x14	512	36.7	16.7	49.1	53.1	.75s
W112-P3-D21-F128	112	3	21	14	512x14x14	128	36.1	16.3	48.4	52.2	.33s
SharpMask	160	4	39	10	1024x10x10	128	39.3	18.1	52.1	57.1	.75s

Table 3.1: Model performance (upper bound on AR) for varying input size W, number of pooling layers P, stride density S, depth D, and features channels F. See §3.4.1 and §3.5.1 for details. Timing is for multiscale inference excluding the time for score prediction. Total time for DeepMask & SharpMask is 1.59s & .76s.

3.5.1 Architecture Optimization

We begin by reporting our optimizations of the feedforward model. For our initial results, we measure AR for densely computed masks (~10^4 proposals per image). This allows us to factor out the effect of objectness score prediction and focus exclusively on evaluating mask quality. In our experiments, AR across all proposals is highly correlated (see Figure 3.6), hence this upper bound on AR is predictive of performance at more realistic settings (e.g. at AR^{100}).

Trunk Architecture: We begin by investigating effect of the network trunk parameters described in §3.4.1 with the goal of optimizing both speed and accuracy. Performance of a number of representative models is shown in Table 3.1. First, replacing the 224×224 DeepMask VGG-A model with a 160×160 version is much faster (over 2×). Surprisingly, accuracy loss for this model, W160-P4-D8-VGG, is only minor, partially due to an improved learning schedule. Upgrading to a ResNet trunk, W160-P4-D39, restores accuracy and keeps speed identical. We found that reducing the feature dimension to 128 (-F128) shows almost no loss, but improves speed. Finally, as input size is a bottleneck, we also tested a number of W112 models. Nevertheless, overall, W160-P4-D39-F128 gave the best tradeoff between speed and accuracy.

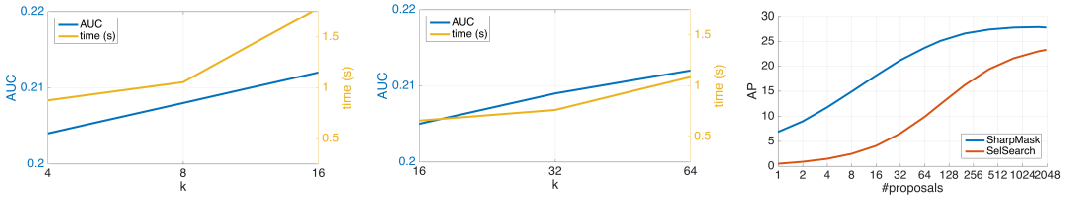
Head Architecture: In Table 3.2 we evaluate the performance of the various network heads in Figure 3.3 (using standard AR, not upper-bound AR as in Table 3.1). Head A is already substantially faster than DeepMask. All heads achieve similar accuracy with a decreasing inference time as the score branch shares progressively more computation with the mask. Interestingly, head C is able to predict both the score and mask from a single compact 512 dimensional vector. We chose this variant due to its simplicity and speed.

DeepMask-ours: Based on all of these observations, we combine the W160-P4-D39-F128 trunk with the C head. We refer to the resulting architecture as *DeepMask-ours*. DeepMask-ours is over 3× faster than the original DeepMask (.46s per image versus 1.59s) and also more accurate. Moreover, model parameter count is reduced from ~75M to ~17M. For all SharpMask experiments, we adopt DeepMask-ours as the base feedforward architecture.

	AR^{10}	AR^{100}	AR^{1K}	AUC^{S}	AUC^{M}	AUC^{L}	AUC	mask	score	total
DeepMask	12.6	24.5	33.1	2.3	26.6	33.6	18.3	1.32s	.27s	1.59s
head A	14.0	25.8	33.4	2.2	27.3	36.6	19.3	.45s	.06s	.51s
head B	14.0	25.4	33.0	2.0	27.0	36.9	19.1	.45s	.05s	.50s
head C	14.4	25.8	33.1	2.2	27.3	37.4	19.4	.45s	.01s	.46s

Table 3.2: All model variants of the head have similar performance. Head C is a win in terms of both simplicity and speed. See Figure 3.3 for head definitions.

3.5.2 SharpMask Analysis



(a) $k_m^i = k_s^i = k$ (b) $k_m^i = k_s^i = \frac{k}{2^{i-1}}$ (c) detection perf.

Figure 3.5: (a-b) Performance and inference time for multiple SharpMask variants. (c) Fast R-CNN detection performance versus number and type of proposals.

We now analyze different parameter settings for our top-down refinement network. As described in §3.3, each of the four refinement modules R^i in SharpMask is controlled by two parameters k_m^i and k_s^i, which denote the size of the mask encoding M^i and skip encoding S^i, respectively. These parameters control network capacity and effect inference speed. We experiment with two different

schedules for these parameters: (a) $k_m^i = k_s^i = k$ and (b) $k_m^i = k_s^i = \frac{k}{2^{i-1}}$ for each $i \leq 4$.

Figure 3.5(a-b) shows performance for the two schedules for different k both in terms of AUC and inference time (measured when refining the top 500 proposals per image, at which point object detection performance saturates, see Figure 3.5c). We consistently observe higher performance as we increase the capacity, with no sign of overfitting. Parameter schedule b, in particular with $k = 32$, has the best trade-off between performance and speed, so we chose this as our final model.

We note that we were unable to obtain good results with schedule a for $k \leq 2$, indicating the importance of using sufficiently large k. Also, we observed that a single 3×3 convolution encounters learning difficulties when ($k_s^i \ll k_f^i$). Therefore, in all experiments we used a sequence of two 3×3 convolutions (followed by ReLUs) to generate S^i from F^i, reducing F^i to 64 channels first followed by a further reduction to k_s^i channels.

Finally, we performed two additional ablation studies. First, we removed all downward convs, set $k_m^i = k_s^i = 1$, and averaged the output of all layers. Second, we kept the vertical convs but removed all horizontal convs. These two variants are related to 'skip' and 'deconv' networks, respectively. Neither setup showed meaningful improvement over the baseline feedforward network. In short, we found that both horizontal and vertical connections were necessary for this task.

	Box Proposals				Segmentation Proposals						
	AR^{10}	AR^{100}	AR^{1K}	AUC	AR^{10}	AR^{100}	AR^{1K}	AUC^{S}	AUC^{M}	AUC^{L}	AUC
EdgeBoxes [131]	7.4	17.8	33.8	13.9	—	—	—	—	—	—	—
Geodesic [58]	4.0	18.0	35.9	12.6	2.3	12.3	25.3	1.3	8.6	20.5	8.5
Rigor [56]	—	13.3	33.7	10.1	—	9.4	25.3	2.2	6.0	17.8	7.4
SelectiveSearch [118]	5.2	16.3	35.7	12.6	2.5	9.5	23.0	0.6	5.5	21.4	7.4
MCG [86]	10.1	24.6	39.8	18.0	7.7	18.6	29.9	3.1	12.9	32.4	13.7
RPN [6, 91]	12.8	29.2	42.6	21.4	—	—	—	—	—	—	—
DeepMask [84]	15.3	31.3	44.6	23.3	12.6	24.5	33.1	2.3	26.6	33.6	18.3
DeepMaskZoom [84]	15.0	32.6	48.2	24.2	12.7	26.1	36.6	6.8	26.3	30.8	19.4
DeepMask-ours	18.7	34.9	46.5	26.2	14.4	25.8	33.1	2.2	27.3	37.4	19.4
SharpMask	19.7	36.4	48.2	27.4	15.6	27.6	35.5	2.5	29.1	40.4	20.9
SharpMaskZoom	20.1	39.4	52.8	29.1	16.1	30.3	39.2	6.9	29.7	38.4	22.4
$SharpMaskZoom^{2}$	19.2	39.9	55.0	29.2	15.4	30.7	40.8	10.6	27.3	36.0	22.5

Table 3.3: Results on the COCO validation set on box and segmentation proposals. AR at different proposals counts is reported and also AUC (AR averaged across all proposal counts). For segmentation proposals, we also report AUC at multiple scales. SharpMask has largest for segmentation proposals and large objects.

3.5.3 Comparison with State of the Art

Table 3.3 compares the performance of our model, SharpMask, to other existing methods on the COCO dataset. We compare results both on box and segmentation proposals (for box proposals we extract tight bounding boxes surrounding our segmentation masks). SharpMask achieves the state of the art in all metrics for both speed and accuracy by a large margin. Additionally, because SharpMask has a smaller input size, it can be applied to an additional one to two scales (*SharpMaskZoom*) and achieves a large boost in AR for small objects.

Our feedforward architecture improvements, *DeepMask-ours*, alone, improve over the original DeepMask, in particular for bounding box proposals. Not only is the new baseline more accurate, with our architecture optimization to the trunk and head of the network (see §3.4), speed is improved to .46s per image. We emphasize that DeepMask was the previous state-of-the-art on this

task, outperforming all bottom-up proposal methods as well as Region Proposal Networks (RPN) [91] (we obtained improved RPN proposals from the authors of [6]).

We train SharpMask using DeepMask-ours as the feedforward network. As the two networks have an identical score branch, we can disentangle the performance improvements achieved by our top-down refinement approach. Once again, we observe a considerable boost in performance on AR due to the top-down refinement. We note that improvement for segmentation predictions is bigger than box predictions, which is not surprising, as sharpening masks might not change the tight box around the objects in many examples. Inference for SharpMask is .76s per image, over 2× faster than DeepMask; moreover, the refinement modules require fewer than 3M additional parameters.

In Figures 3.2 and 3.9 we show direct comparison between SharpMask and DeepMask and we can see SharpMask generates higher-fidelity masks that more accurately delineate object boundaries. In Figures 3.4 and 3.8, we show more qualitative results. Additional detailed performance plots are shown in Figure 3.6.

3.5.4 Object Detection

In this section, we use SharpMask in the Fast R-CNN pipeline [38] and analyze the improvements of using our proposals for object detection. In the following experiments we coupled SharpMask proposals with two classifiers: VGG [110] and MultiPathNet (MPN) [126], which introduces a number of improvements to the VGG classifier. In future work we will also test our proposals

with ResNets [48].

First, Fig. 3.5c shows the comparison of bounding box detection results for SharpMask and SelSearch [118] on the COCO val set with the MPN classifier applied to both. SharpMask achieves 28 AP, which is 5 AP higher than SelSearch. Also, performance converges using only ~500 SharpMask proposals per image.

Next, Table 3.4 top shows results of various baselines without bells and whistles, trained on the train set only. SharpMask achieves top results with the VGG classifier, outperforming both RPN [91] and SelSearch [118].

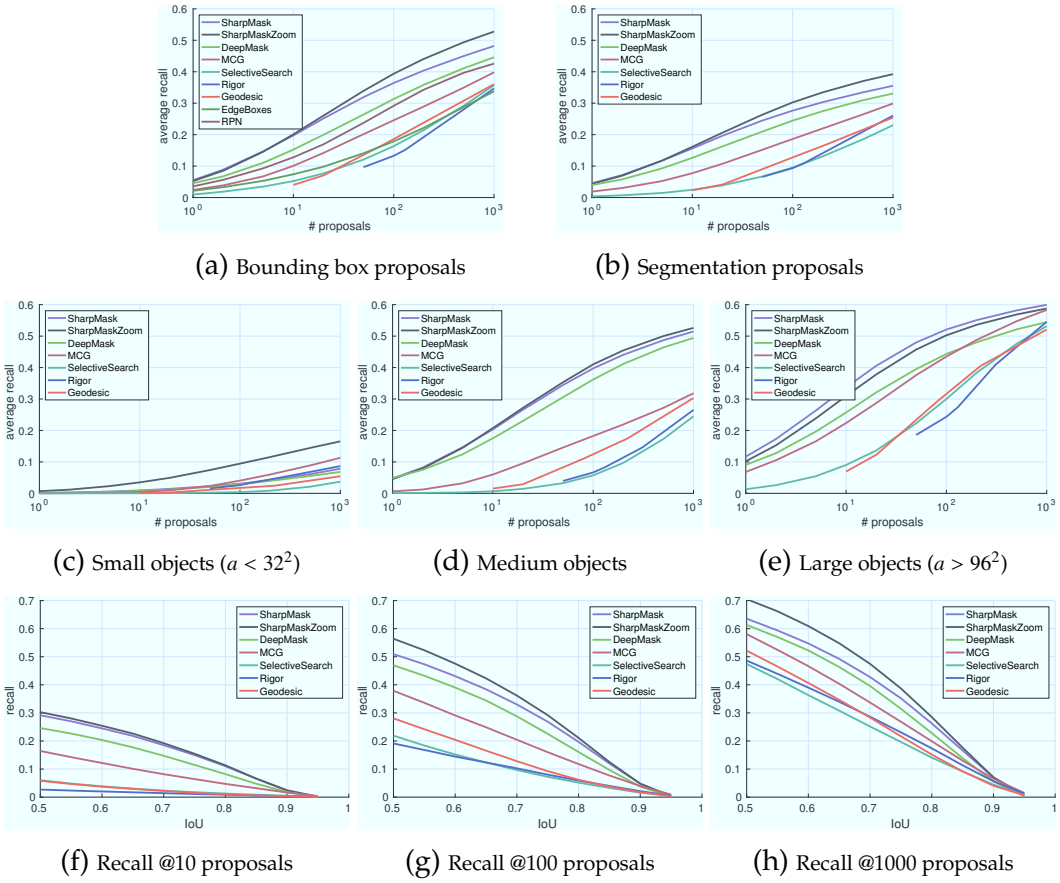
Finally, Table 3.4 middle/bottom shows results from the 2015 COCO detection challenges. The performance is reported with model ensembling and the MPN classifier. The ensemble model achieve 33.5 AP for boxes and 25.1 AP for segments, and achieved second place in the challenges. Note that for the challenges, both SharpMask and MPN used the VGG trunk (ResNets were concurrent work, and won the competitions). We have not re-run our model with ensembling and additional bells and whistles after integrating ResNets into SharpMask.

3.6 Conclusion

In this paper, we introduce a novel architecture for object instance segmentation, based on an augmentation of feedforward networks with top-down refinement modules. Our model achieves a new state of the art for object proposals generation, both in terms of performance and speed. The proposed refinement approach is general and could be applied to other pixel-labeling tasks.

	AP	AP^{50}	AP^{75}	AP^{S}	AP^{M}	AP^{L}	AR^{1}	AR^{10}	AR^{100}	AR^{S}	AR^{M}	AR^{L}
SelSearch + VGG [38]	19.3	39.3	—	—	—	—	—	—	—	—	—	—
RPN + VGG [91]	21.9	42.7	—	—	—	—	—	—	—	—	—	—
SharpMask + VGG	25.2	43.4	—	—	—	—	—	—	—	—	—	—
ResNet++ [48]	28.2	51.5	27.9	9.3	30.6	45.2	25.7	37.4	38.2	16.8	43.9	57.6
SharpMask+MPN[126]	25.1	45.8	24.8	7.4	29.2	39.1	24.1	36.8	38.7	17.3	46.9	53.9
ResNet++ [48]	37.3	58.9	39.9	18.3	41.9	52.4	32.1	47.7	49.1	27.3	55.6	67.9
SharpMask+MPN[126]	33.5	52.6	36.6	13.9	37.8	47.7	30.2	46.2	48.5	24.1	56.1	66.4
ION [6]	31.0	53.3	31.8	12.3	33.2	44.7	27.9	43.1	45.7	23.8	50.4	62.8

Table 3.4: **Top**: COCO bounding box results of various baselines without bells and whistles, trained on the train set only, and reported on test-dev (results for [38, 91] obtained from original papers). We denote methods using 'proposal+classifier' notation for clarity. SharpMask achieves top results, outperforming both RPN and SelSearch proposals. **Middle**: Winners of the 2015 COCO segmentation challenge. **Bottom**: Winners of the 2015 COCO bounding box challenge.



(a) Bounding box proposals (b) Segmentation proposals

(c) Small objects ($a < 32^2$) (d) Medium objects (e) Large objects ($a > 96^2$)

(f) Recall @10 proposals (g) Recall @100 proposals (h) Recall @1000 proposals

Figure 3.6: (a-b) Average recall versus number of box and segment proposals on COCO. (c-e) AR versus number of proposals for different object scales on segment proposals. (f-h) Recall versus IoU threshold for different number of segment proposals.

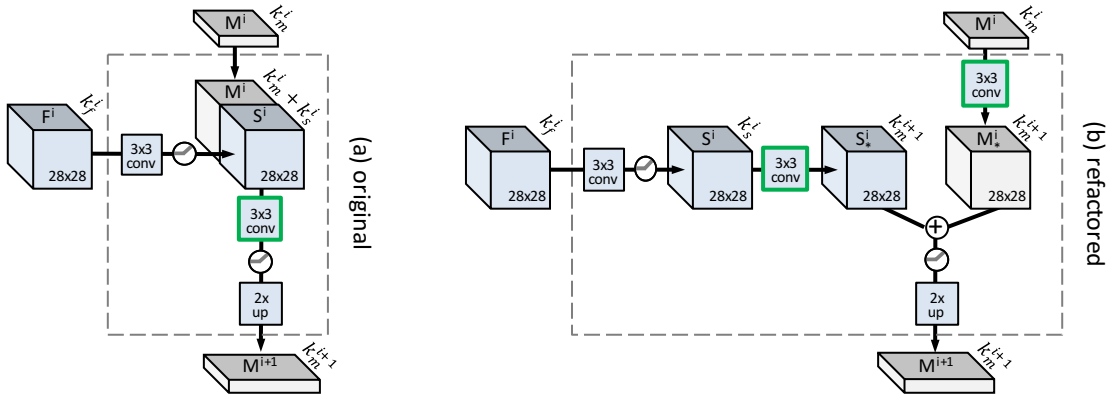


Figure 3.7: (a) Original refinement model. (b) Refactored but *equivalent* model that leads to a more efficient implementation. The models are equivalent as concatenating along depth and convolving along the spatial dimensions can be rewritten as two separate spatial convolutions followed by addition. The green 'conv' boxes denote the corresponding convolutions (note also the placement of the ReLUs). The refactored model is more efficient as skip features (both S^i and S^i_*) are shared by overlapping refinement windows (while M^i and M^i_* are not). Finally, observe that setting $k^i_m = 1, \forall i$, and removing the top-down convolution would transform our refactored model into a standard 'skip' architecture (however, using $k^i_m = 1$ is not effective in our setting).

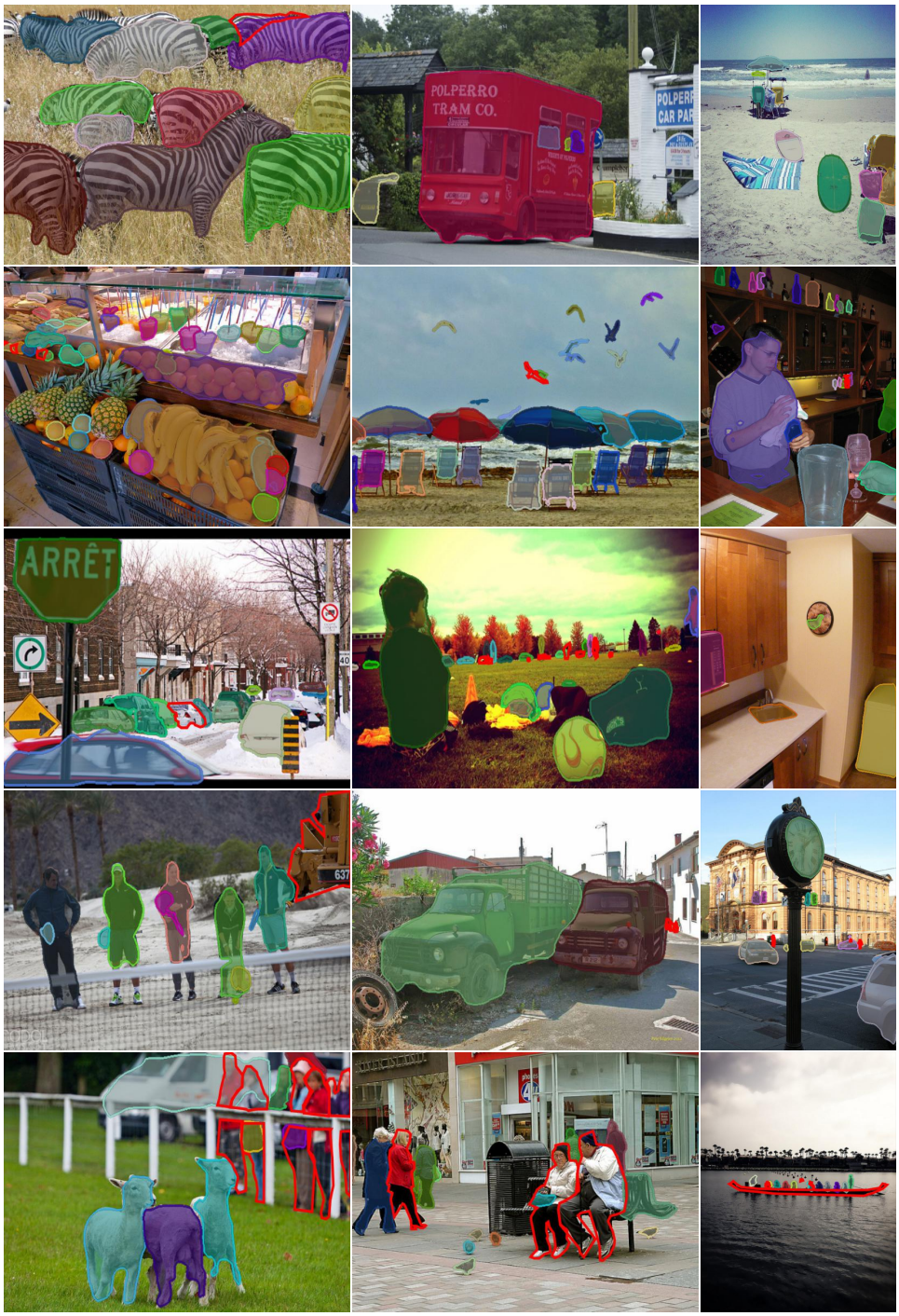


Figure 3.8: More selected qualitative results (see also Figure 3.4).

(a) DeepMask Output (b) SharpMask Output

Figure 3.9: More selected qualitative comparisons (see also Figure 3.2).

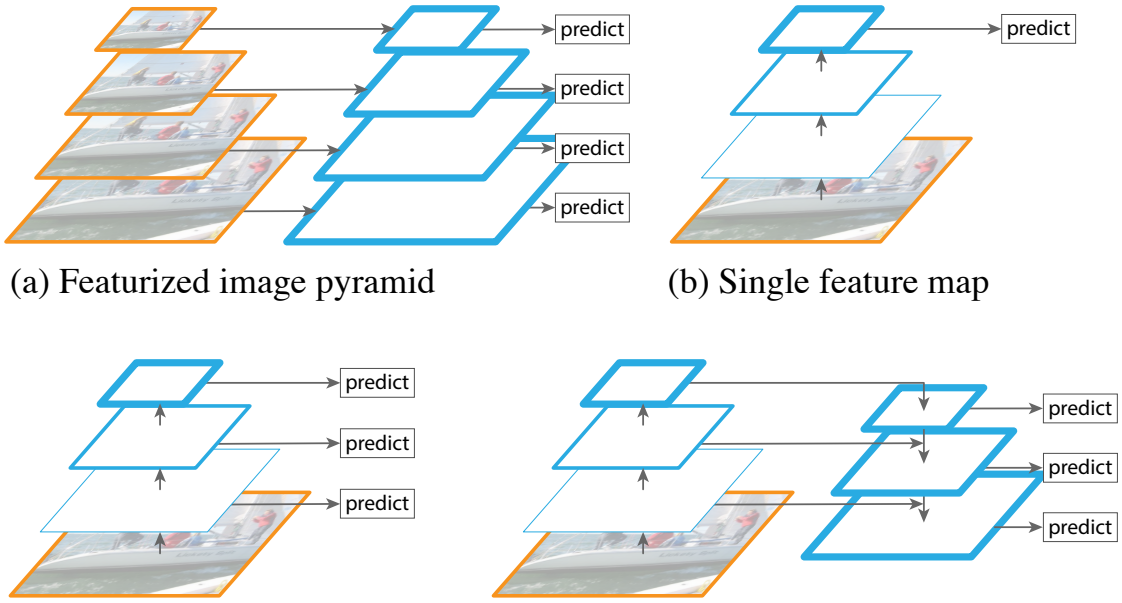
CHAPTER 4

LEARNING MULTISCALE FEATURE RRESENTATIONS

4.1 Introduction

Recognizing objects at vastly different scales is a fundamental challenge in computer vision. *Feature pyramids built upon image pyramids* (for short we call these *featurized image pyramids*) form the basis of a standard solution [1] (Fig. 4.1(a)). These pyramids are scale-invariant in the sense that an object's scale change is offset by shifting its level in the pyramid. Intuitively, this property enables a model to detect objects across a large range of scales by scanning the model over both positions and pyramid levels.

Featurized image pyramids were heavily used in the era of hand-engineered features [17, 74]. They were so critical that object detectors like DPM [33] required dense scale sampling to achieve good results (*e.g.*,, 10 scales per octave). For recognition tasks, engineered features have largely been replaced with features computed by deep convolutional networks (ConvNets) [61, 65]. Aside from being capable of representing higher-level semantics, ConvNets are also more robust to variance in scale and thus facilitate recognition from features computed on a single input scale [47, 41, 92] (Fig. 4.1(b)). But even with this robustness, pyramids are still needed to get the most accurate results. All recent top entries in the ImageNet [97] and COCO [69] detection challenges use multi-scale testing on featurized image pyramids (*e.g.*,, [48, 107]). The principle advantage of featurizing each level of an image pyramid is that it produces a multi-scale feature representation in which *all levels are semantically strong*, including the high-resolution levels.



(a) Featurized image pyramid (b) Single feature map

(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Figure 4.1: (a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow. (b) Recent detection systems have opted to use only single scale features for faster detection. (c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid. (d) Our proposed Feature Pyramid Network (FPN) is fast like (b) and (c), but more accurate. In this figure, feature maps are indicate by blue outlines and thicker outlines denote semantically stronger features.

Nevertheless, featurizing each level of an image pyramid has obvious limitations. Inference time increases considerably (*e.g.*,, by four times [41]), making this approach impractical for real applications. Moreover, training deep networks end-to-end on an image pyramid is infeasible in terms of memory, and so, if exploited, image pyramids are used only at test time [47, 41, 48, 107], which creates an inconsistency between train/test-time inference. For these reasons, Fast and Faster R-CNN [41, 92] opt to not use featurized image pyramids under default settings.

However, image pyramids are not the only way to compute a multi-scale feature representation. A deep ConvNet computes a *feature hierarchy* layer by layer,

and with subsampling layers the feature hierarchy has an inherent multi-scale, pyramidal shape. This in-network feature hierarchy produces feature maps of different spatial resolutions, but introduces large semantic gaps caused by different depths. The high-resolution maps have low-level features that harm their representational capacity for object recognition.

The Single Shot Detector (SSD) [71] is one of the first attempts at using a ConvNet's pyramidal feature hierarchy as if it were a featurized image pyramid (Fig. 4.1(c)). Ideally, the SSD-style pyramid would reuse the multi-scale feature maps from different layers computed in the forward pass and thus come free of cost. But to avoid using low-level features SSD foregoes reusing already computed layers and instead builds the pyramid starting from high up in the network (*e.g.*,, conv4_3 of VGG nets [110]) and then by adding several new layers. Thus it misses the opportunity to reuse the higher-resolution maps of the feature hierarchy. We show that these are important for detecting small objects.

The goal of this paper is to naturally leverage the pyramidal shape of a ConvNet's feature hierarchy while creating a feature pyramid that has strong semantics at all scales. To achieve this goal, we rely on an architecture that combines low-resolution, semantically strong features with high-resolution, semantically weak features via a top-down pathway and lateral connections (Fig. 4.1(d)). The result is a feature pyramid that has rich semantics at all levels and is built quickly from a single input image scale. In other words, we show how to create in-network feature pyramids that can be used to replace featurized image pyramids without sacrificing representational power, speed, or memory.

Similar architectures adopting top-down and skip connections are popular

in recent research [85, 52, 35, 76]. Their goals are to produce a single high-level feature map of a fine resolution on which the predictions are to be made (Fig. 4.2 top). On the contrary, our method leverages the architecture as a feature pyramid where predictions (*e.g.*,, object detections) are independently made on each level (Fig. 4.2 bottom). Our model echoes a featurized image pyramid, which has not been explored in these works.

We evaluate our method, called a Feature Pyramid Network (FPN), in various systems for detection and segmentation [41, 92, 84]. Without bells and whistles, we report a state-of-the-art single-model result on the challenging COCO detection benchmark [69] simply based on FPN and a basic Faster R-CNN detector [92], surpassing all existing heavily-engineered single-model entries of competition winners. In ablation experiments, we find that for bounding box proposals, FPN significantly increases the Average Recall (AR) by 8.0 points; for object detection, it improves the COCO-style Average Precision (AP) by 2.3 points and PASCAL-style AP by 3.8 points, over a strong single-scale baseline of Faster R-CNN on ResNets [48]. Our method is also easily extended to mask proposals and improves both instance segmentation AR and speed over state-of-the-art methods that heavily depend on image pyramids.

In addition, our pyramid structure can be trained end-to-end with all scales and is used consistently at train/test time, which would be memory-infeasible using image pyramids. As a result, FPNs are able to achieve higher accuracy than all existing state-of-the-art methods. Moreover, this improvement is achieved without increasing testing time over the single-scale baseline. We believe these advances will facilitate future research and applications. Our code will be made publicly available.

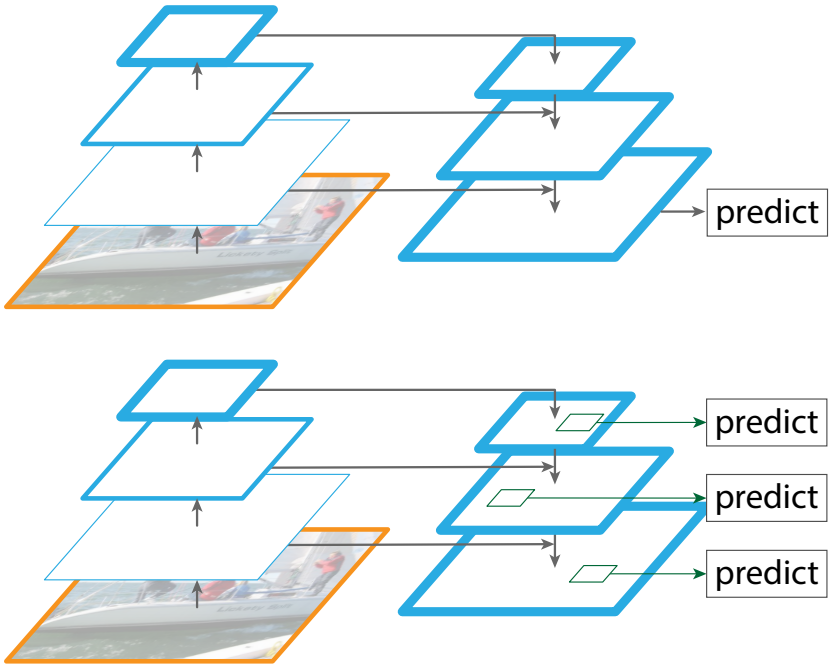


Figure 4.2: Top: a top-down architecture with skip connections, where predictions are made on the finest level (*e.g.*,, [85]). Bottom: our model that has a similar structure but leverages it as a *feature pyramid*, with predictions made independently at all levels.

4.2 Related Work

Hand-engineered features and early neural networks. SIFT features [74] were originally extracted at scale-space extrema and used for feature point matching. HOG features [17], and later SIFT features as well, were computed densely over entire image pyramids. These HOG and SIFT pyramids have been used in numerous works for image classification, object detection, human pose estimation, and more. There has also been significant interest in computing featurized image pyramids quickly. Dollár *et al.*[21] demonstrated fast pyramid computation by first computing a sparsely sampled (in scale) pyramid and then interpolating missing levels. Before HOG and SIFT, early work on face detection with ConvNets [119, 95] computed shallow networks over image pyramids to detect

faces across scales.

Deep ConvNet object detectors. With the development of modern deep ConvNets [61], object detectors like OverFeat [104] and R-CNN [39] showed dramatic improvements in accuracy. OverFeat adopted a strategy similar to early neural network face detectors by applying a ConvNet as a sliding window detector on an image pyramid. R-CNN adopted a region proposal-based strategy [118] in which each proposal was scale-normalized before classifying with a ConvNet. SPPnet [47] demonstrated that such region-based detectors could be applied much more efficiently on feature maps extracted on a single image scale. Recent and more accurate detection methods like Fast R-CNN [41] and Faster R-CNN [92] advocate using features computed from a single scale, because it offers a good trade-off between accuracy and speed. Multi-scale detection, however, still performs better, especially for small objects.

Methods using multiple layers. A number of recent approaches improve detection and segmentation by using different layers in a ConvNet. FCN [73] sums partial scores for each category over multiple scales to compute semantic segmentations. Hypercolumns [44] uses a similar method for object instance segmentation. Several other approaches (HyperNet [57], ParseNet [72], and ION [6]) concatenate features of multiple layers before computing predictions, which is equivalent to summing transformed features. SSD [71] and MS-CNN [11] predict objects at multiple layers of the feature hierarchy without combining features or scores.

There are recent methods exploiting lateral/skip connections that associate low-level feature maps across resolutions and semantic levels, including U-Net [94] and SharpMask [85] for segmentation, Recombinator networks [52] for face

detection, and Stacked Hourglass networks [76] for keypoint estimation. Ghiasi *et al*.[35] present a Laplacian pyramid presentation for FCNs to progressively refine segmentation. Although these methods adopt architectures with pyramidal shapes, they are unlike featurized image pyramids [17, 33, 104] where predictions are made independently at all levels, see Fig. 4.2. In fact, for the pyramidal architecture in Fig. 4.2 (top), image pyramids are still needed to recognize objects across multiple scales [85].

4.3 Feature Pyramid Networks

Our goal is to leverage a ConvNet's pyramidal feature hierarchy, which has semantics from low to high levels, and build a feature pyramid with high-level semantics throughout. The resulting *Feature Pyramid Network* is general-purpose and in this paper we focus on sliding window proposers (Region Proposal Network, RPN for short) [92] and region-based detectors (Fast R-CNN) [41]. We also generalize FPNs to instance segmentation proposals in Sec. 4.6.

Our method takes a single-scale image of an arbitrary size as input, and outputs proportionally sized feature maps at multiple levels, in a fully convolutional fashion. This process is independent of the backbone convolutional architectures (*e.g.*,, [61, 110, 48]), and in this paper we present results using ResNets [48]. The construction of our pyramid involves a bottom-up pathway, a top-down pathway, and lateral connections, as introduced in the following.

Bottom-up pathway. The bottom-up pathway is the feedforward computation of the backbone ConvNet, which computes a feature hierarchy consisting of

feature maps at several scales with a scaling step of 2. There are often many layers producing output maps of the same size and we say these layers are in the same network *stage*. For our feature pyramid, we define one pyramid level for each stage. We choose the output of the last layer of each stage as our reference set of feature maps, which we will enrich to create our pyramid. This choice is natural since the deepest layer of each stage should have the strongest features.

Specifically, for ResNets [48] we use the feature activations output by each stage's last residual block. We denote the output of these last residual blocks as $\{C_2, C_3, C_4, C_5\}$ for conv2, conv3, conv4, and conv5 outputs, and note that they have strides of {4, 8, 16, 32} pixels with respect to the input image. We do not include conv1 into the pyramid due to its large memory footprint.

Top-down pathway and lateral connections. The top-down pathway hallucinates higher resolution features by upsampling spatially coarser, but semantically stronger, feature maps from higher pyramid levels. These features are then enhanced with features from the bottom-up pathway via lateral connections. Each lateral connection merges feature maps of the same spatial size from the bottom-up pathway and the top-down pathway. The bottom-up feature map is of lower-level semantics, but its activations are more accurately localized as it was subsampled fewer times.

Fig. 4.3 shows the building block that constructs our top-down feature maps. With a coarser-resolution feature map, we upsample the spatial resolution by a factor of 2 (using nearest neighbor upsampling for simplicity). The upsampled map is then merged with the corresponding bottom-up map (which undergoes a 1×1 convolutional layer to reduce channel dimensions) by element-wise addition. This process is iterated until the finest resolution map is generated. To

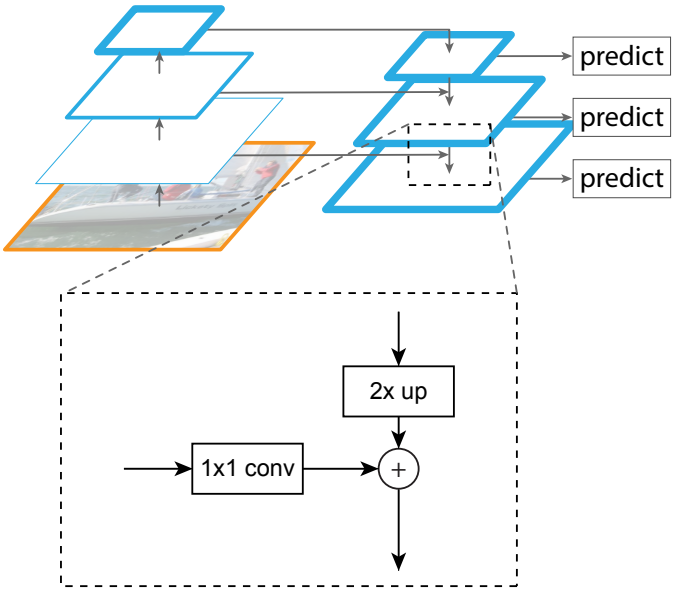


Figure 4.3: A building block illustrating the lateral connection and the top-down pathway, merged by addition.

start the iteration, we simply attach a 1×1 convolutional layer on C_5 to produce the coarsest resolution map. Finally, we append a 3×3 convolution on each merged map to generate the final feature map, which is to reduce the aliasing effect of upsampling. This final set of feature maps is called $\{P_2, P_3, P_4, P_5\}$, corresponding to $\{C_2, C_3, C_4, C_5\}$ that are respectively of the same spatial sizes.

Because all levels of the pyramid use shared classifiers/regressors as in a traditional featurized image pyramid, we fix the feature dimension (numbers of channels, denoted as d) in all the feature maps. We set $d = 256$ in this paper and thus all extra convolutional layers have 256-channel outputs. There are no non-linearities in these extra layers, which we have empirically found to have minor impacts.

Simplicity is central to our design and we have found that our model is robust to many design choices. We have experimented with more sophisticated blocks (*e.g.*,, using multi-layer residual blocks [48] as the connections) and ob-

served marginally better results. Designing better connection modules is not the focus of this paper, so we opt for the simple design described above.

4.4 Applications

Our method is a generic solution for building feature pyramids inside deep ConvNets. In the following we adopt our method in RPN [92] for bounding box proposal generation and in Fast R-CNN [41] for object detection. To demonstrate the simplicity and effectiveness of our method, we make minimal modifications to the original systems of [92, 41] when adapting them to our feature pyramid.

4.4.1 Feature Pyramid Networks for RPN

RPN [92] is a sliding-window class-agnostic object detector. In the original RPN design, a small subnetwork is evaluated on dense 3×3 sliding windows, on top of a single-scale convolutional feature map, performing object/non-object binary classification and bounding box regression. This is realized by a 3×3 convolutional layer followed by two sibling 1×1 convolutions for classification and regression, which we refer to as a network *head*. The object/non-object criterion and bounding box regression target are defined with respect to a set of reference boxes called *anchors* [92]. The anchors are of multiple pre-defined scales and aspect ratios in order to cover objects of different shapes.

We adapt RPN by replacing the single-scale feature map with our FPN. We attach a head of the same design (3×3 conv and two sibling 1×1 convs) to each

level on our feature pyramid. Because the head slides densely over all locations in all pyramid levels, it is not necessary to have multi-scale anchors on a specific level. Instead, we assign anchors of a single scale to each level. Formally, we define the anchors to have areas of $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ pixels on $\{P_2, P_3, P_4, P_5, P_6\}$ respectively.[1] As in [92] we also use anchors of multiple aspect ratios {1:2, 1:1, 2:1} at each level. So in total there are 15 anchors over the pyramid.

We assign training labels to the anchors based on their Intersection-over-Union (IoU) ratios with ground-truth bounding boxes as in [92]. Formally, an anchor is assigned a positive label if it has the highest IoU for a given ground-truth box or an IoU over 0.7 with any ground-truth box, and a negative label if it has IoU lower than 0.3 for all ground-truth boxes. Note that scales of ground-truth boxes are not explicitly used to assign them to the levels of the pyramid; instead, ground-truth boxes are associated with anchors, which have been assigned to pyramid levels. As such, we introduce no extra rules in addition to those in [92].

We note that the parameters of the heads are shared across all feature pyramid levels; we have also evaluated the alternative without sharing parameters and observed similar accuracy. The good performance of sharing parameters indicates that all levels of our pyramid share similar semantic levels. This advantage is analogous to that of using a featurized image pyramid, where a common head classifier can be applied to features computed at any image scale.

With the above adaptations, RPN can be naturally trained and tested with our FPN, in the same fashion as in [92]. We elaborate on the implementation

[1]Here we introduce P_6 only for covering a larger anchor scale of 512^2. P_6 is simply a stride two subsampling of P_5. P_6 is not used by the Fast R-CNN detector in the next section.

details in the experiments.

4.4.2 Feature Pyramid Networks for Fast R-CNN

Fast R-CNN [41] is a region-based object detector in which Region-of-Interest (RoI) pooling is used to extract features. Fast R-CNN is most commonly performed on a single-scale feature map. To use it with our FPN, we need to assign RoIs of different scales to the pyramid levels.

We view our feature pyramid as if it were produced from an image pyramid. Thus we can adapt the assignment strategy of region-based detectors [47, 41] in the case when they are run on image pyramids. Formally, we assign an RoI of width w and height h (on the input image to the network) to the level P_k of our feature pyramid by:

$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor. \tag{4.1}$$

Here 224 is the canonical ImageNet pre-training size, and k_0 is the target level on which an RoI with $w \times h = 224^2$ should be mapped into. Analogous to the ResNet-based Faster R-CNN system [48] that uses C_4 as the single-scale feature map, we set k_0 to 4. Intuitively, Eqn. (4.1) means that if the RoI's scale becomes smaller (say, 1/2 of 224), it should be mapped into a finer-resolution level (say, $k = 3$).

We attach predictor heads (in Fast R-CNN the heads are class-specific classifiers and bounding box regressors) to all RoIs of all levels. Again, the heads all share parameters, regardless of their levels. In [48], a ResNet's conv5 layers (a 9-layer deep subnetwork) are adopted as the head on top of the conv4 features, but our method has already harnessed conv5 to construct the feature pyramid.

So unlike [48], we simply adopt RoI pooling to extract 7×7 features, and attach two hidden 1,024-d fully-connected (*fc*) layers (each followed by ReLU) before the final classification and bounding box regression layers. These layers are randomly initialized, as there are no pre-trained *fc* layers available in ResNets. Note that compared to the standard conv5 head, our 2-*fc* MLP head is lighter weight and faster.

Based on these adaptations, we can train and test Fast R-CNN on top of the feature pyramid. Implementation details are given in the experimental section.

4.5 Experiments on Object Detection

We perform experiments on the 80 category COCO detection dataset [69]. We train using the union of 80k train images and a 35k subset of val images (`trainval35k` [6]), and report ablations on a 5k subset of val images (`minival`). We also report final results on the standard test set (`test-std`) [69] which has no disclosed labels.

As is common practice [39], all network backbones are pre-trained on the ImageNet1k classification set [97] and then fine-tuned on the detection dataset. We use the pre-trained ResNet-50 and ResNet-101 models that are publicly available.[2] Our code is a reimplementation of `py-faster-rcnn`[3] using Caffe2.[4]

[2] `https://github.com/kaiminghe/deep-residual-networks`

[3] `https://github.com/rbgirshick/py-faster-rcnn`

[4] `https://github.com/caffe2/caffe2`

4.5.1 Region Proposal with RPN

We evaluate the COCO-style Average Recall (AR) and AR on small, medium, and large objects (AR_s, AR_m, and AR_l) following the definitions in [69]. We report results for 100 and 1000 proposals per images (AR^{100} and AR^{1k}).

Implementation details. All architectures in Table 4.1 are trained end-to-end. The input image is resized such that its shorter side has 800 pixels. We adopt synchronized SGD training on 8 GPUs. A mini-batch involves 2 images per GPU and 256 anchors per image. We use a weight decay of 0.0001 and a momentum of 0.9. The learning rate is 0.02 for the first 30k mini-batches and 0.002 for the next 10k. For all RPN experiments (including baselines), we include the anchor boxes that are outside the image for training, which is unlike [92] where these anchor boxes are ignored. Other implementation details are as in [92]. Training RPN with FPN on 8 GPUs takes about 8 hours on COCO.

Ablation Experiments

Comparisons with baselines. For fair comparisons with original RPNs [92], we run two baselines (Table 4.1(a, b)) using the single-scale map of C_4 (the same as [48]) or C_5, both using the same hyper-parameters as ours, including using 5 scale anchors of $\{32^2, 64^2, 128^2, 256^2, 512^2\}$. Table 4.1 (b) shows no advantage over (a), indicating that a single higher-level feature map is not enough because there is a trade-off between coarser resolutions and stronger semantics.

Placing FPN in RPN improves AR^{1k} to 56.3 (Table 4.1 (c)), which is **8.0** points increase over the single-scale RPN baseline (Table 4.1 (a)). In addition, the per-

RPN	feature	# anchors	lateral?	top-down?	AR^{100}	AR^{1k}	AR^{1k}_s	AR^{1k}_m	AR^{1k}_l
(a) baseline on conv4	C_4	47k			36.1	48.3	32.0	58.7	62.2
(b) baseline on conv5	C_5	12k			36.3	44.9	25.3	55.5	64.2
(c) **FPN**	$\{P_k\}$	200k	✓	✓	**44.0**	**56.3**	**44.9**	**63.4**	66.2
Ablation experiments follow:									
(d) bottom-up pyramid	$\{P_k\}$	200k	✓		37.4	49.5	30.5	59.9	**68.0**
(e) top-down pyramid, w/o lateral	$\{P_k\}$	200k		✓	34.5	46.1	26.5	57.4	64.7
(f) only finest level	P_2	750k	✓	✓	38.4	51.3	35.1	59.7	67.6

Table 4.1: Bounding box proposal results using RPN [92], evaluated on the COCO `minival` set. All models are trained on `trainval35k`. The columns "lateral" and "top-down" denote the presence of lateral and top-down connections, respectively. The column "feature" denotes the feature maps on which the heads are attached. All results are based on ResNet-50 and share the same hyper-parameters.

Fast R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(a) baseline on conv4	RPN, $\{P_k\}$	C_4	conv5			54.7	31.9	15.7	36.5	45.5
(b) baseline on conv5	RPN, $\{P_k\}$	C_5	2*fc*			52.9	28.8	11.9	32.4	43.4
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	**45.8**
Ablation experiments follow:										
(d) bottom-up pyramid	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*	✓		44.9	24.9	10.9	24.4	38.5
(e) top-down pyramid, w/o lateral	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*		✓	54.0	31.3	13.3	35.2	45.3
(f) only finest level	RPN, $\{P_k\}$	P_2	2*fc*	✓	✓	56.3	33.4	17.3	37.3	45.6

Table 4.2: Object detection results using **Fast R-CNN** [41] on *a fixed set of proposals* (RPN, $\{P_k\}$, Table 4.1(c)), evaluated on the COCO `minival` set. Models are trained on the `trainval35k` set. All results are based on ResNet-50 and share the same hyper-parameters.

Faster R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(*) baseline [48]†	RPN, C_4	C_4	conv5			47.3	26.3	-	-	-
(a) baseline on conv4	RPN, C_4	C_4	conv5			53.1	31.6	13.2	35.6	**47.1**
(b) baseline on conv5	RPN, C_5	C_5	2*fc*			51.7	28.0	9.6	31.9	43.1
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	45.8

Table 4.3: Object detection results using **Faster R-CNN** [92] evaluated on the COCO `minival` set. *The backbone network for RPN are consistent with Fast R-CNN.* Models are trained on the `trainval35k` set and use ResNet-50. †Provided by authors of [48].

formance on small objects (AR^{1k}_s) is boosted by a large margin of 12.9 points. Our pyramid representation greatly improves RPN's robustness to object scale variation.

How important is top-down enrichment? Table 4.1(d) shows the results of our feature pyramid without the top-down pathway. With this modification, the 1×1 lateral connections followed by 3×3 convolutions are attached to the bottom-up pyramid. This architecture simulates the effect of reusing the pyramidal feature hierarchy (Fig. 4.1(b)).

The results in Table 4.1(d) are just on par with the RPN baseline and lag far behind ours. We conjecture that this is because there are large semantic gaps between different levels on the bottom-up pyramid (Fig. 4.1(b)), especially for very deep ResNets. We have also evaluated a variant of Table 4.1(d) without sharing the parameters of the heads, but observed similarly degraded performance. This issue cannot be simply remedied by level-specific heads.

How important are lateral connections? Table 4.1(e) shows the ablation results of a top-down feature pyramid without the 1×1 lateral connections. This top-down pyramid has strong semantic features and fine resolutions. But we argue that the locations of these features are not precise, because these maps have been downsampled and upsampled several times. More precise locations of features can be directly passed from the finer levels of the bottom-up maps via the lateral connections to the top-down maps. As a results, FPN has an AR^{1k} score 10 points higher than Table 4.1(e).

How important are pyramid representations? Instead of resorting to pyramid representations, one can attach the head to the highest-resolution, strongly se-

mantic feature maps of P_2 (*i.e.*,, the finest level in our pyramids). Similar to the single-scale baselines, we assign all anchors to the P_2 feature map. This variant (Table 4.1(f)) is better than the baseline but inferior to our approach. RPN is a sliding window detector with a fixed window size, so scanning over pyramid levels can increase its robustness to scale variance.

In addition, we note that using P_2 alone leads to more anchors (750k, Table 4.1(f)) caused by its large spatial resolution. This result suggests that a larger number of anchors is not sufficient in itself to improve accuracy.

4.5.2 Object Detection with Fast/Faster R-CNN

Next we investigate FPN for region-based (non-sliding window) detectors. We evaluate object detection by the COCO-style Average Precision (AP) and PASCAL-style AP (at a single IoU threshold of 0.5). We also report COCO AP on objects of small, medium, and large sizes (namely, AP_s, AP_m, and AP_l) following the definitions in [69].

Implementation details. The input image is resized such that its shorter side has 800 pixels. Synchronized SGD is used to train the model on 8 GPUs. Each mini-batch involves 2 image per GPU and 512 RoIs per image. We use a weight decay of 0.0001 and a momentum of 0.9. The learning rate is 0.02 for the first 60k mini-batches and 0.002 for the next 20k. We use 2000 RoIs per image for training and 1000 for testing. Training Fast R-CNN with FPN takes about 10 hours on the COCO dataset.

method	backbone	competition	image pyramid	test-dev					test-std				
				$AP_{@.5}$	AP	AP_s	AP_m	AP_l	$AP_{@.5}$	AP	AP_s	AP_m	AP_l
ours	ResNet-101	-		**59.1**	**36.2**	**18.2**	**39.0**	48.2	**58.5**	**35.8**	**17.5**	**38.7**	47.8
*Competition-winning **single-model** results follow:*													
G-RMI†	Inception-ResNet	2016		-	34.7	-	-	-	-	-	-	-	-
AttractioNet‡ [37]	VGG16 + Wide ResNet§	2016	✓	53.4	35.7	15.6	38.0	**52.7**	52.9	35.3	14.7	37.6	**51.9**
Faster R-CNN [48]	ResNet-101	2015	✓	55.7	34.9	15.6	38.7	50.9	-	-	-	-	-
Multipath [126]	VGG-16	2015		49.6	31.5	-	-	-	-	-	-	-	-
ION‡ [6]	VGG-16	2015		53.4	31.2	12.8	32.9	45.2	52.9	30.7	11.8	32.8	44.8

Table 4.4: Comparisons of **single-model** results on the COCO detection benchmark. Some results were not available on the `test-std` set, so we also include the `test-dev` results (and for Multipath [126] on `minival`). †: `http://image-net.org/challenges/talks/2016/GRMI-COCO-slidedeck.pdf`. ‡: `http://mscoco.org/dataset/\#detections-leaderboard`. §: This entry of AttractioNet [37] adopts VGG-16 for proposals and Wide ResNet [125] for object detection, so is not strictly a single-model result.

Fast R-CNN (on fixed proposals)

To better investigate FPN's effects on the region-based detector alone, we conduct ablations of Fast R-CNN on *a fixed set of proposals*. We choose to freeze the proposals as computed by RPN on FPN (Table 4.1(c)), because it has good performance on small objects that are to be recognized by the detector. For simplicity we do not share features between Fast R-CNN and RPN, except when specified.

As a ResNet-based Fast R-CNN baseline, following [48], we adopt RoI pooling with an output size of 14×14 and attach all conv5 layers as the hidden layers of the head. This gives an AP of 31.9 in Table 4.2(a). Table 4.2(b) is a baseline exploiting an MLP head with 2 hidden *fc* layers, similar to the head in our architecture. It gets an AP of 28.8, indicating that the 2-*fc* head does not give us any orthogonal advantage over the baseline in Table 4.2(a).

Table 4.2(c) shows the results of our FPN in Fast R-CNN. Comparing with the baseline in Table 4.2(a), our method improves AP by 2.0 points and small object AP by 2.1 points. Comparing with the baseline that also adopts a 2*fc* head (Table 4.2(b)), our method improves AP by 5.1 points.[5] These comparisons indicate that our feature pyramid is superior to single-scale features for a *region-based* object detector.

Table 4.2(d) and (e) show that removing top-down connections or removing lateral connections leads to inferior results, similar to what we have observed in the above subsection for RPN. It is noteworthy that removing top-down connections (Table 4.2(d)) significantly degrades the accuracy, suggesting that Fast R-CNN suffers from using the low-level features at the high-resolution maps.

In Table 4.2(f), we adopt Fast R-CNN on the single finest scale feature map of P_2. Its result (33.4 AP) is marginally worse than that of using all pyramid levels (33.9 AP, Table 4.2(c)). We argue that this is because RoI pooling is a warping-like operation, which is less sensitive to the region's scales. Despite the good accuracy of this variant, it is based on the RPN proposals of $\{P_k\}$ and has thus already benefited from the pyramid representation.

Faster R-CNN (on consistent proposals)

In the above we used a fixed set of proposals to investigate the detectors. But in a Faster R-CNN system [92], the RPN and Fast R-CNN must use *the same network backbone* in order to make feature sharing possible. Table 4.3 shows the comparisons between our method and two baselines, all using *consistent* backbone

[5] We expect a stronger architecture of the head [93] will improve upon our results, which is beyond the focus of this paper.

	ResNet-50		ResNet-101	
share features?	$AP_{@0.5}$	AP	$AP_{@0.5}$	AP
no	56.9	33.9	58.0	35.0
yes	**57.2**	**34.3**	**58.2**	**35.2**

Table 4.5: More object detection results using Faster R-CNN and our FPNs, evaluated on `minival`. Sharing features increases train time by 1.5× (using 4-step training [92]), but reduces test time.

architectures for RPN and Fast R-CNN. Table 4.3(a) shows our reproduction of the baseline Faster R-CNN system as described in [48]. Under controlled settings, our FPN (Table 4.3(c)) is better than this strong baseline by **2.3** points AP and **3.8** points AP@0.5.

Note that Table 4.3(a) and (b) are baselines that are much stronger than the baseline provided by He *et al*.[48] in Table 4.3(*). We find the following implementations contribute to the gap: (i) We use an image scale of 800 pixels instead of 600 in [41, 48]; (ii) We train with 512 RoIs per image which accelerate convergence, in contrast to 64 RoIs in [41, 48]; (iii) We use 5 scale anchors instead of 4 in [48] (adding 32^2); (iv) At test time we use 1000 proposals per image instead of 300 in [48]. So comparing with He *et al*.'s ResNet-50 Faster R-CNN baseline in Table 4.3(*), our method improves AP by 7.6 points and AP@0.5 by 9.6 points.

Sharing features. In the above, for simplicity we do not share the features between RPN and Fast R-CNN. In Table 4.5, we evaluate sharing features following the 4-step training described in [92]. Similar to [92], we find that sharing features improves accuracy by a small margin. Feature sharing also reduces the testing time.

Running time. With feature sharing, our FPN-based Faster R-CNN system has inference time of 0.148 seconds per image on a single NVIDIA M40 GPU for

ResNet-50, and 0.172 seconds for ResNet-101.[6] As a comparison, the single-scale ResNet-50 baseline in Table 4.3(a) runs at 0.32 seconds. Our method introduces small extra cost by the extra layers in the FPN, but has a lighter weight head. Overall our system is faster than the ResNet-based Faster R-CNN counterpart. We believe the efficiency and simplicity of our method will benefit future research and applications.

Comparing with COCO Competition Winners

We find that our ResNet-101 model in Table 4.5 is not sufficiently trained with the default learning rate schedule. So we increase the number of mini-batches by 2× at each learning rate when training the Fast R-CNN step. This increases AP on `minival` to 35.6, without sharing features. This model is the one we submitted to the COCO detection leaderboard, shown in Table 4.4. We have not evaluated its feature-sharing version due to limited time, which should be slightly better as implied by Table 4.5.

Table 4.4 compares our method with the *single-model* results of the COCO competition winners, including the 2016 winner G-RMI and the 2015 winner Faster R-CNN+++. *Without adding bells and whistles*, our single-model entry has surpassed these strong, heavily engineered competitors. On the `test-dev` set, our method increases over the existing best results by **0.5** points of AP (36.2 *v.s.*35.7) and **3.4** points of AP@0.5 (59.1 *v.s.*55.7). It is worth noting that our method does not rely on image pyramids and only uses a single input image scale, but still has outstanding AP on small-scale objects. This could only be achieved by high-resolution image inputs with previous methods.

[6]These runtimes are updated from an earlier version of this paper.

Moreover, our method does *not* exploit many popular improvements, such as iterative regression [36], hard negative mining [107], context modeling [48], stronger data augmentation [71], etc.. These improvements are complementary to FPNs and should boost accuracy further.

Recently, FPN has enabled new top results in *all* tracks of the COCO competition, including detection, instance segmentation, and keypoint estimation. See [46] for details.

4.6 Extensions: Segmentation Proposals

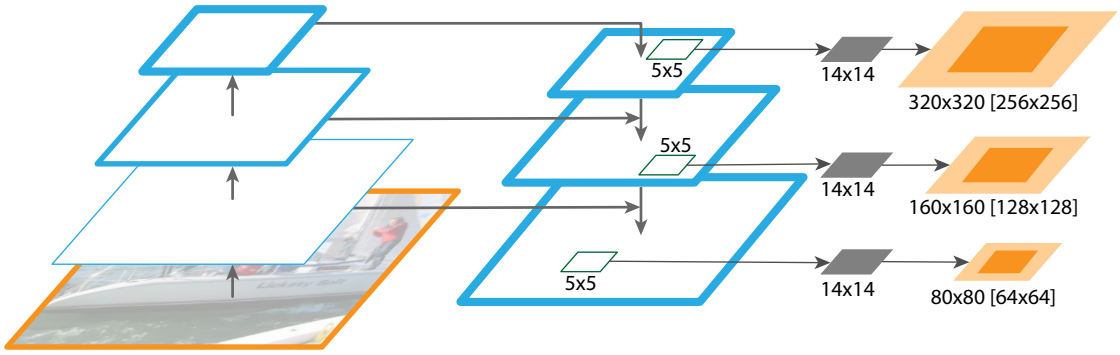


Figure 4.4: FPN for object segment proposals. The feature pyramid is constructed with identical structure as for object detection. We apply a small MLP on 5x5 windows to generate dense object segments with output dimension of 14x14. Shown in orange are the size of the image regions the mask corresponds to for each pyramid level (levels P_{3-5} are shown here). Both the corresponding image region size (light orange) and canonical object size (dark orange) are shown. Half octaves are handled by an MLP on 7x7 windows ($7 \approx 5\sqrt{2}$), not shown here. Details are in the appendix.

Our method is a generic pyramid representation and can be used in applications other than object detection. In this section we use FPNs to generate segmentation proposals, following the DeepMask/SharpMask framework [84, 85].

DeepMask/SharpMask were trained on image crops for predicting instance

segments and object/non-object scores. At inference time, these models are run convolutionally to generate dense proposals in an image. To generate segments at multiple scales, image pyramids are necessary [84, 85].

It is easy to adapt FPN to generate mask proposals. We use a fully convolutional setup for both training and inference. We construct our feature pyramid as in Sec. 4.5.1 and set $d = 128$. On top of each level of the feature pyramid, we apply a small 5x5 MLP to predict 14x14 masks and object scores in a fully convolutional fashion, see Fig. 4.4. Additionally, motivated by the use of 2 scales per octave in the image pyramid of [84, 85], we use a second MLP of input size 7x7 to handle half octaves. The two MLPs play a similar role as anchors in RPN. The architecture is trained end-to-end; full implementation details are given in the appendix.

	image pyramid	AR	AR_s	AR_m	AR_l	time (s)
DeepMask [84]	✓	37.1	15.8	50.1	54.9	0.49
SharpMask [85]	✓	39.8	17.4	53.1	59.1	0.77
InstanceFCN [13]	✓	39.2	–	–	–	1.50†
FPN Mask Results:						
single MLP [5x5]		43.4	32.5	49.2	53.7	**0.15**
single MLP [7x7]		43.5	30.0	49.6	57.8	0.19
dual MLP [5x5, 7x7]		45.7	31.9	51.5	60.8	0.24
+ 2x mask resolution		46.7	31.7	53.1	63.2	0.25
+ 2x train schedule		**48.1**	**32.6**	**54.2**	**65.6**	0.25

Table 4.6: Instance segmentation proposals evaluated on the first 5k COCO `val` images. All models are trained on the `train` set. DeepMask, SharpMask, and FPN use ResNet-50 while InstanceFCN uses VGG-16. DeepMask and SharpMask performance is computed with models available from `https://github.com/facebookresearch/deepmask` (both are the 'zoom' variants). †Runtimes are measured on an NVIDIA M40 GPU, except the InstanceFCN timing which is based on the slower K40.

4.6.1 Segmentation Proposal Results

Results are shown in Table 4.6. We report segment AR and segment AR on small, medium, and large objects, always for 1000 proposals. Our baseline FPN model with a single 5x5 MLP achieves an AR of 43.4. Switching to a slightly larger 7x7 MLP leaves accuracy largely unchanged. Using both MLPs together increases accuracy to 45.7 AR. Increasing mask output size from 14x14 to 28x28 increases AR another point (larger sizes begin to degrade accuracy). Finally, doubling the training iterations increases AR to 48.1.

We also report comparisons to DeepMask [84], SharpMask [85], and InstanceFCN [13], the previous state of the art methods in mask proposal generation. We outperform the accuracy of these approaches by over **8.3** points AR. In particular, we nearly double the accuracy on small objects.

Existing mask proposal methods [84, 85, 13] are based on densely sampled image pyramids (*e.g.*,, scaled by $2^{\{-2:0.5:1\}}$ in [84, 85]), making them computationally expensive. Our approach, based on FPNs, is substantially faster (our models run at 6 to 7 FPS). These results demonstrate that our model is a generic feature extractor and can replace image pyramids for other multi-scale detection problems.

4.7 Conclusion

We have presented a clean and simple framework for building feature pyramids inside ConvNets. Our method shows significant improvements over several strong baselines and competition winners. Thus, it provides a practical

solution for research and applications of feature pyramids, without the need of computing image pyramids. Finally, our study suggests that despite the strong representational power of deep ConvNets and their implicit robustness to scale variation, it is still critical to explicitly address multi-scale problems using pyramid representations.

CHAPTER 5

FOCAL LOSS FOR DENSE OBJECT DETECTION

5.1 Introduction

Current state-of-the-art object detectors are based on a two-stage, proposal-driven mechanism. As popularized in the R-CNN framework [39], the first stage generates a *sparse* set of candidate object locations and the second stage classifies each candidate location as one of the foreground classes or as background using a convolutional neural network. Through a sequence of advances [41, 92, 67, 46], this two-stage framework consistently achieves top accuracy on the challenging COCO benchmark [69].

Despite the success of two-stage detectors, a natural question to ask is: could a simple one-stage detector achieve similar accuracy? One stage detectors are applied over a regular, *dense* sampling of object locations, scales, and aspect ratios. Recent work on one-stage detectors, such as YOLO [89, 90] and SSD [71, 34], demonstrates promising results, yielding faster detectors with accuracy within 10-40% relative to state-of-the-art two-stage methods.

This paper pushes the envelop further: we present a one-stage object detector that, for the first time, matches the state-of-the-art COCO AP of more complex two-stage detectors, such as the Feature Pyramid Network (FPN) [67] or Mask R-CNN [46] variants of Faster R-CNN [92]. To achieve this result, we identify class imbalance during training as the main obstacle impeding one-stage detector from achieving state-of-the-art accuracy and propose a new loss function that eliminates this barrier.

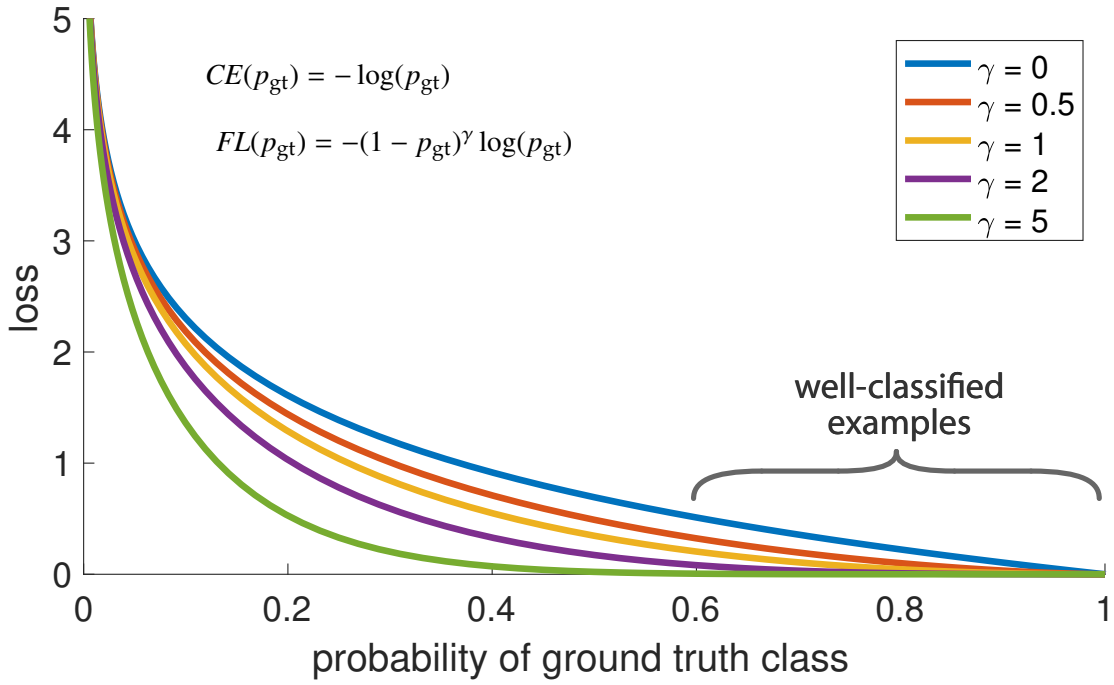


Figure 5.1: We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_{gt})^{\gamma}$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_{gt} > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

Class imbalance is addressed in R-CNN-like detectors by a two-stage cascade and sampling heuristics. The proposal stage (*e.g.*,, Selective Search [118], EdgeBoxes [131], DeepMask [84, 85], RPN [92]) rapidly narrows down the number of candidate object locations to a small number (*e.g.*,, 1-2k), filtering out most background samples. In the second classification stage, sampling heuristics, such as a fixed foreground-to-background ratio (1:3), or online hard example mining (OHEM) [107], are performed to maintain a manageable balance between foreground and background.

In contrast, a one-stage detector must process a much larger set of candidate object locations regularly sampled across an image. In practice this often amounts to enumerating ~100k locations that densely cover spatial positions,

scales, and aspect ratios. While similar sampling heuristics may also be applied, they are inefficient as the training procedure is still dominated by easily classified background examples. This inefficiency is a classic problem in object detection that is typically addressed via techniques such as bootstrapping [112, 95] or hard example mining [120, 32, 107].

In this paper, we propose a new loss function that acts as a more effective alternative to previous approaches for dealing with class imbalance. The loss function is a dynamically scaled cross entropy loss, where the scaling factor decays to zero as confidence in the correct class increases, see Figure 5.1. Intuitively, this scaling factor can automatically down-weight the contribution of easy examples during training and rapidly focus the model on hard examples. Experiments show that our proposed *Focal Loss* enables us to train a high-accuracy, one-stage detector that significantly outperforms the alternatives of training with the sampling heuristics or hard example mining, the previous state-of-the-art techniques for training one-stage detectors. Finally, we note that the exact form of the focal loss is not crucial, and we show other instantiations can achieve similar results.

To demonstrate the effectiveness of the proposed focal loss, we design a simple and intuitive one-stage object detector called *RetinaNet*, named for its dense sampling of object locations in an input image. Its design features an efficient in-network feature pyramid and use of anchor boxes. It draws on a variety of recent ideas from [71, 26, 92, 67]. RetinaNet is efficient and accurate; our best model, based on a ResNet-101-FPN backbone, achieves a COCO `test-dev` AP of 38.1 while running at 5 fps, surpassing the previously best published single-model results from both one and two-stage detectors (see Figure 5.4).

5.2 Related Work

Classic Object Detectors: The sliding-window paradigm, in which a classifier is applied on a dense image grid, has a long and rich history. One of the earliest successes is the classic work of LeCun *et al*.who applied convolutional neural networks to handwritten digit recognition [65, 119]. Viola and Jones [120] used boosted object detectors for face detection, leading to widespread adoption of such models. The introduction of HOG [17] and integral channel features [22] gave rise to effective methods for pedestrian detection. DPMs [32] helped extend dense detectors to more general object categories and had top results on PASCAL [27] for many years. While the sliding-window approach was the leading detection paradigm in classic computer vision, with the resurgence of deep learning [61], two-stage detectors, described next, quickly came to dominate object detection.

Two-stage Detectors: The dominant paradigm in modern object detection is based on a two-stage approach. As pioneered in the Selective Search work [118], the first stage generates a sparse set of candidate proposals that should contain all objects while filtering out the majority of negative locations, and the second stage classifies the proposals into foreground classes / background. R-CNN [39] upgraded the second-stage classifier to a convolutional network yielding large gains in accuracy and ushering in the modern era of object detection. R-CNN was improved over the years, both in terms of speed [47, 41] and by using learned object proposals [26, 84, 92] in place of bottom-up segmentation proposals. Region Proposal Networks (RPN) integrated proposal generation with the second-stage classifier into a single convolution network, forming the Faster R-CNN framework [92]. Numerous extensions to this framework have been

proposed, *e.g.*,[67, 107, 108, 48, 46] among others.

One-stage Detectors: OverFeat [104] was one of the first modern one-stage object detector based on deep networks. More recently SSD [71, 34] and YOLO [89, 90] have renewed interest in one-stage methods. These detectors have been tuned for speed but their accuracy trails that of two-stage methods. SSD has a 10-20% lower AP, while YOLO focuses on an even more extreme speed/accuracy trade-off. See Figure 5.4. In contrast, the aim of this work is to understand if one-stage detectors can match or surpass the accuracy of two-stage detectors while running at a similar speed. This is motivated by recent work showing that two-stage detectors can be made fast by simply reducing input image resolution and the number of proposals [55].

The design of our RetinaNet detector shares many similarities with previous dense detectors, in particular the concept of 'anchors' introduced by RPN [92] and use of features pyramids as in SSD [71] and FPN [67]. We emphasize that our simple detector achieves top results not based on innovations in network design but due to our novel loss.

Class Imbalance: Both classic one-stage object detection methods, like boosted detectors [120, 22] and DPMs [32], and more recent methods, like SSD [71], face a massive class imbalance during training. These detectors sample 10^4-10^5 candidate locations per image but only a few locations contain objects. This imbalance causes two problems: (1) training is inefficient as most locations are easy negatives that contribute no useful learning signal; (2) en masse, the easy negatives can overwhelm training and lead to degenerate models. A common solution is to perform some form of hard negative mining [112, 120, 32, 107, 71] that samples hard examples during training. In contrast, we show that our pro-

posed focal loss naturally handles the class imbalance faced by a one-stage detector and allows us to efficiently train on all examples without sampling and without easy negatives overwhelming the loss and computed gradients.

Robust Estimation: There has been much interest in designing robust loss functions (*e.g.*,, Huber loss [45]) that reduce the contribution of *outliers* by down-weighting the loss of examples with large errors (hard examples). In contrast, rather than addressing outliers, our focal loss is designed to address class imbalance by down-weighting *inliers* (easy examples) such that their contribution to the total loss is small even if their number is large. In other words, the focal loss perform the *opposite* role of a robust loss: it focuses training on a sparse set of hard examples.

5.3 Focal Loss

The *Focal Loss* is designed to address the one-stage object detection scenario in which there is an extreme imbalance between foreground and background classes during training (*e.g.*,, 1:1000). We introduce the focal loss starting from the cross entropy (CE) loss for binary classification:

$$CE(p, y) = -y \log(p) - (1 - y) \log(1 - p). \tag{5.1}$$

In the above $y \in \{0, 1\}$ specifies the ground-truth class and $p \in [0, 1]$ is the model's estimated probability for class with label $y = 1$. For notational convenience, we introduce p_{gt}:

$$p_{\text{gt}} = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise,} \end{cases} \tag{5.2}$$

and rewrite $CE(p, y) = CE(p_{gt}) = -\log(p_{gt})$.

The CE loss can be seen as the blue (top) curve in Figure 5.1. One notable property of this loss, which can be easily seen in its plot, is that even examples that are easily classified ($p_{gt} \gg .5$) incur a loss with non-trivial magnitude. When summed over a large number of easy examples, these small loss values can overwhelm the rare class.

5.3.1 Balanced Cross Entropy

A common method for addressing class imbalance is to introduce a weighting factor $\alpha \in [0, 1]$ for class 1 and $1-\alpha$ for class 0. In practice α may be set by inverse class frequency or treated as a hyperparameter to set by cross validation. For notational convenience, we define α_{gt} analogously to how we defined p_{gt}. We write the α-balanced CE loss as:

$$CE(p_{gt}) = -\alpha_{gt} \log(p_{gt}). \tag{5.3}$$

This loss is a simple extension to CE that we consider as an experimental baseline for our proposed focal loss.

5.3.2 Focal Loss Definition

As our experiments will show, the severe foreground-background class imbalance encountered during training of dense detectors overwhelms the cross entropy loss. Easily classified negatives comprise the majority of the loss and dominate the computed gradient. While α provides one way to adjust the cross en-

tropy, it is not sufficient. Instead, we propose to reshape the loss function to down-weight easy examples and thus focus training on hard negatives.

More formally, we propose to add a modulating factor $(1 - p_{\text{gt}})^{\gamma}$ to the cross entropy loss, with tunable *focusing* parameter $\gamma \geq 0$. We define the focal loss as:

$$FL(p_{\text{gt}}) = -(1 - p_{\text{gt}})^{\gamma} \log(p_{\text{gt}}). \tag{5.4}$$

The focal loss is visualized for several values of $\gamma \in [0, 5]$ in Figure 5.1. We note two properties of the focal loss. (1) When an example is misclassified and p_{gt} is small, the modulating factor is near 1 and the loss is unaffected. As $p_{\text{gt}} \to 1$, the factor goes to 0 and the loss for well-classified examples is down-weighted. (2) The focusing parameter γ smoothly adjusts the rate at which easy examples are down-weighted. When $\gamma = 0$, FL is equivalent to CE, and as γ is increased the effect of the modulating factor is likewise increased (we found $\gamma = 2$ to work best in our experiments).

Intuitively, the modulating factor reduces the loss contribution from easy examples and extends the range in which an example receives low loss. For instance, with $\gamma = 2$, an example classified with $p_{\text{gt}} = 0.9$ would have $100x$ lower loss compared with CE and with $p_{\text{gt}} \approx 0.968$ it would have $1000x$ lower loss. This in turn increases the importance of correcting misclassified examples (whose loss is scaled down by at most $4x$ for $p_{\text{gt}} \leq .5$ and $\gamma = 2$).

In practice we use an α-balanced variant of the focal loss:

$$FL(p_{\text{gt}}) = -\alpha_{\text{gt}}(1 - p_{\text{gt}})^{\gamma} \log(p_{\text{gt}}). \tag{5.5}$$

We adopt this form in our experiments as it yields slightly improved accuracy over the non-α-balanced form. Finally, we note that the implementation of the

loss layer combines the sigmoid operation for computing p with the loss computation, resulting in greater numerical stability.

While in our main experimental results we use the focal loss definition above, its precise form is not crucial. In the appendix we consider alternate instantiations of the focal loss and demonstrate that these can be equally effective.

5.3.3 Class Imbalance and Model Initialization

Binary classification models are by default initialized to have equal probability of outputting either $y = 0$ or 1. Under such an initialization, in the presence of class imbalance, the loss due to the frequent class can dominate total loss and cause instability in early training. To counter this, we introduce the concept of a 'prior' for the value of p estimated by the model for the rare class (foreground) *at the start of training*. We denote the prior by π and set it so that the model's estimated p for examples of the rare class is low, *e.g.*,0.01. We note that this is a change in model initialization (see §5.4.2) and *not* of the loss function. We found this to improve training stability for both the cross entropy and focal loss in the case of heavy class imbalance.

5.3.4 Class Imbalance and Two-stage Detectors

Two-stage detectors are often trained with the cross entropy loss without use of α-balancing or our proposed loss. Instead, they address class imbalance through two mechanisms: (1) a two-stage cascade and (2) biased minibatch sampling.

The first cascade stage is an object proposal mechanism [118, 84, 92] that reduces the nearly infinite set of possible object locations down to one or two thousand. Importantly, the selected proposals are not random, but are likely to correspond to true object locations, which removes the vast majority of easy negatives. When training the second stage, biased sampling is typically used to construct minibatches that contain, for instance, a 1:3 ratio of positive to negative examples. This ratio is like an implicit α-balancing factor that is implemented via sampling. Our proposed focal loss is designed to address these mechanisms in a one-stage detection system directly via the loss function.

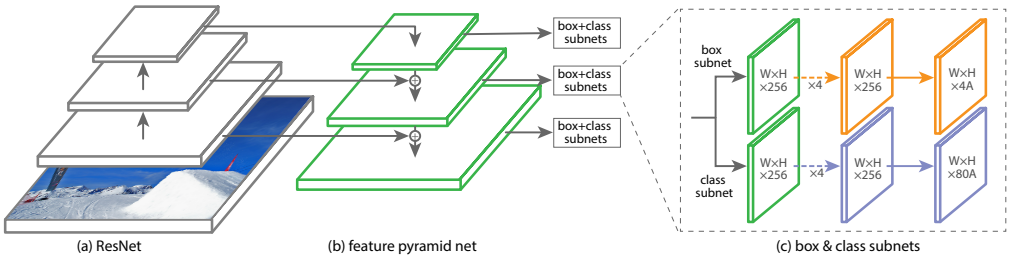


Figure 5.2: The one-stage **RetinaNet** network architecture uses a Feature Pyramid Network (FPN) [67] (left side) backbone to generate a rich, multi-scale convolutional feature pyramid. To this backbone RetinaNet attaches two subnetworks (right side), one for classifying anchor boxes and one for regressing from anchor boxes to ground-truth object boxes. The network design is intentionally simple and intuitive, which enables this work to focus on a novel loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN [67].

5.4 RetinaNet Detector

RetinaNet is a single, unified network composed of a *backbone* network and two task-specific *subnetworks*. The backbone is responsible for computing a convolutional feature map over an entire input image and is an off-the-self convolutional network. The first subnet performs convolutional object classification

on the backbone's output; the second subnet performs convolutional bounding box regression. The two subnetworks feature a simple design that we propose specifically for one-stage, dense detection (see Figure 5.2). While there are many possible choices for the details of these components, most of the design parameters are not particularly sensitive to exact values. We show the effect of alternatives with lesion and sensitivity studies in §5.5.1.

Backbone: We adopt the Feature Pyramid Network (FPN) from [67] as the backbone convolutional network used in our experiments. In brief, FPN augments a standard convolutional network with a top-down pathway and lateral connections so the network efficiently constructs a rich, multi-scale feature pyramid from a single resolution input image (Figure 5.2 (a+b)). FPN improves dense, multi-scale predictions from fully convolutional networks (FCN) [73] as demonstrated by its application to RPN and convolutional DeepMask-style [84] mask proposal generation.

Following [67], we refer to the multi-scale feature maps output by the backbone as P_l, where l indicates the pyramid level (*e.g.*,, P_2, P_3, ...). In our experiments we build FPN on top of the ResNet-50 and ResNet-101 architectures [48] and note two minor changes compared to [67]: (1) rather than computing P_6 by a $2x$ subsampling of P_5, we instead apply a $3x3$ / stride 2 conv layer to C_5 (the output of the final residual block in ResNet-50/101); (2) we construct an additional even coarser pyramid level, P_7, by applying ReLU followed by a $3x3$ / stride 2 conv layer to P_6. The number of filters in these new conv layers is set so P_6 and P_7 have the same number of channels as the other FPN levels. The addition of P_7 improves large object AP while introducing minimal extra computation. While many design choices are not critical, we emphasize the use of

an FPN-like backbone is; preliminary experiments using features from a single network level showed a substantially lower AP.

Anchors: We use translation-invariant anchor boxes as described for the FPN variant of RPN in [67], aside from a two minor differences. First, our model places anchors from P_3 onwards, instead of P_2 as in [67] (and therefore we do not compute P_2). This choice reduces computation time, since P_2 has roughly $3x$ as many spatial locations as P_3 to P_7 *combined*. Starting anchors from P_3 increases the finest anchor stride from 4 to 8 pixels, which we found does not significantly reduce AP. Second, on each pyramid level we use three anchor aspect ratios, as in [67], but unlike [67] we use three anchor scales spaced a factor of $2^{1/3}$ apart on each level. These intermediate scales provide denser coverage and improve AP. Our anchors cover the scale range 32 - 813 pixels with respect to the network's input image.

Object Classification Subnet: During inference, for each anchor the model outputs K binomial distributions (each a probability between 0 and 1) for each of the K object classes. These predictions are the output of a specially designed object classification subnet that is a small FCN attached to each FPN level (P_3 to P_7). Parameters of this subnet are shared across all pyramid levels. The classification subnet has a simple design: taking an input feature map (*e.g.*,, P_4) with C channels, the subnet applies four $3x3$ conv layers, each with C filters and each followed by ReLU activations. A final $3x3$ conv layer with AK filters, followed by sigmoid activations, is used to output the AK binomial predictions, where A is the number of anchors co-located at each spatial position (see Figure 5.2 (c) bottom). Typical values used in our experiments are $C = 256$ and $A = 9$.

In contrast to RPN, our object classification subnet is deeper, uses only $3x3$

convs, and is not shared with the box regression subnet (described next). We found these higher-level design decisions to be more important than specific values of hyperparameters.

Box Regression Subnet: In parallel with the object classification subnet, we attach another small FCN to each FPN level for the purpose of regressing the geometric transformation from each anchor to a nearby ground-truth object, if one exists. The design of the box regression subnet is identical to the classification subnet with the exception that it terminates in $4A$ linear outputs per spatial location (see Figure 5.2 (c) top). For each of the A co-located anchors, these 4 outputs predict the relative shift in anchor box position and log scale needed to reconstruct a ground-truth box from the anchor. We use the standard parameterization from R-CNN, Fast R-CNN, and RPN [39]. We note that unlike most recent work employing bounding box regressors, we use a class-agnostic regressor which uses fewer parameters and we found it to work about as well as a class-aware box regressor in practice.

Inference: The RetinaNet network forms single FCN comprised of a ResNet-FPN backbone, an object classification subnet, and a box regression subnet. As such, inference is performed by simply forwarding a test image through the network. To improve inference speed, we only decode the box predictions from the top-scoring 1k predictions, per FPN level, after thresholding detector confidence at 0.05. Finally, the top predictions from all levels are merged and non-maximum suppression with a threshold of 0.5 is applied to yield the final detections.

5.4.1 Initialization

We experiment with ResNet-50-FPN and ResNet-101-FPN backbones. The underlying RN-50 and RN-101 models are pre-trained on ImageNet1k; we use the models released by [48]. New layers added for FPN are initialized as described in [67]. The new conv layers in the RetinaNet subnets are initialized with bias $b = 0$ and a Gaussian weight fill with $\sigma = 0.01$, with one exception: for the final conv layer of the object classification subnet, we set the bias initialization to $b = -\log((1-\pi)/\pi)$, where π (set to 0.01 in most experiments) specifies that at the start of training every anchors should be labeled as background with a confidence of approximately $1 - \pi$. As explained in §5.4.1, this initialization prevents the large number of background anchors from generating a huge, destabilizing loss value at the very start of training. The object classification subnet and the box regression subnet, though sharing a common structure, use separate parameters unless otherwise stated.

5.4.2 Training

Focal Loss: We use the focal loss introduced in this paper as the loss on the output of the classification subnet. We find that $\gamma = 2$ works well in practice and the RetinaNet is relatively robust to $\gamma \in [0.5, 5]$. We emphasize that when training RetinaNet, the focal loss is applied to *all* ~100k anchors in each sampled image. This stands in contrast to standard methods that use heuristic sampling (RPN) or hard example mining (OHEM, SSD) to select a limited set example anchors (*e.g.*,, 256) for each minibatch image. We also note that α also has a stable range, but it interacts with γ making it necessary to select the two together

α	AP	AP_{50}	AP_{75}
.10	0.0	0.0	0.0
.25	10.8	16.0	11.7
.50	30.2	46.7	32.8
.75	31.1	49.4	33.0
.90	30.8	49.7	32.3
.99	28.7	47.4	29.9
.999	25.1	41.7	26.1

(a) **Varying α for CE loss** ($\gamma = 0$)

γ	α	AP	AP_{50}	AP_{75}
0	.75	31.1	49.4	33.0
0.1	.75	31.4	49.9	33.1
0.2	.75	31.9	50.7	33.4
0.5	.50	32.9	51.7	35.2
1.0	.25	33.7	52.0	36.2
2.0	.25	**34.0**	**52.5**	**36.5**
5.0	.25	32.2	49.6	34.8

(b) **Varying anchor scales and aspects**

#sc	#ar	AP	AP_{50}	AP_{75}
1	1	30.3	49.0	31.8
2	1	31.9	50.0	34.0
3	1	31.8	49.4	33.7
1	3	32.4	52.3	33.9
2	3	**34.2**	**53.1**	**36.5**
3	3	34.0	52.5	**36.5**
4	3	33.8	52.1	36.2

(c) **Varying γ for FL** (w. optimal α)

method	batch size	nms thr	AP	AP_{50}	AP_{75}
OHEM	128	.7	31.1	47.2	33.2
OHEM	256	.7	31.8	48.8	33.9
OHEM	512	.7	30.6	47.0	32.6
OHEM	128	.5	32.8	50.3	35.1
OHEM	256	.5	31.0	47.4	33.0
OHEM	512	.5	27.6	42.0	29.2
OHEM 1:3	128	.5	31.1	47.2	33.2
OHEM 1:3	256	.5	28.3	42.4	30.3
OHEM 1:3	512	.5	24.0	35.5	25.8
FL	n/a	n/a	**36.0**	**54.9**	**38.7**

(d) **FL *v.s.* OHEM** baselines (with ResNet-101-FPN)

model	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	time
RN50-400	30.5	47.8	32.7	11.2	33.8	46.1	64
RN50-500	32.5	50.9	34.8	13.9	35.8	46.7	72
RN50-600	34.3	53.2	36.9	16.2	37.4	47.4	98
RN50-700	35.1	54.2	37.7	18.0	39.3	46.4	121
RN50-800	35.7	55.0	38.5	18.9	38.9	46.3	153
RN101-400	31.9	49.5	34.1	11.6	35.8	48.5	81
RN101-500	34.4	53.1	36.8	14.7	38.5	49.1	90
RN101-600	36.0	55.2	38.7	17.4	39.6	49.7	122
RN101-700	37.1	56.6	39.8	19.1	40.6	49.4	154
RN101-800	37.8	57.5	40.8	20.2	41.1	49.2	198

(e) **Accuracy/speed trade-off** RetinaNet (on `test-dev`)

Table 5.1: **Ablation and sensitivity experiments for RetinaNet.** All models are trained on `trainval35k` and tested on `minival` unless noted. If not specified, default values are: $\gamma = 2$; anchors for 3 scales and 3 aspect ratios; ResNet-50-FPN backbone; and a 600 pixel train and test image scale. We compare RetinaNet trained with our proposed focal loss to strong baselines including α-balanced cross entropy loss (a) and two variants of online hard example mining (OHEM) [107, 71] (d). We find that the focal loss strongly outperforms the best results from either approach by about 3 points AP. Table (e) illustrates the accuracy/speed trade-off provided by RetinaNet on `test-dev`.

(see Table 5.1a and Table 5.1c).

Implementation Details: Unless otherwise stated, we set the α-balancing factor to 0.25 for the foreground class. The total focal loss for one image is computed as the sum of the individual balanced anchor focal losses divided by the number of anchors assigned to a ground-truth box. To improve numerical stability we clip probabilities estimated by the model at a threshold τ, using $p_{\text{gt}} = \max(p_{\text{gt}}, \tau)$, prior to computing the focal loss. With a small value of $\tau = 0.00001$ our model achieves the same accuracy as with a larger value of $\tau = 0.01$, however it will occasionally diverge early in training. With the larger value, training is always stable. We emphasize that good results are achieved with $\tau = 0.00001$ and therefore clipping is not responsible for addressing the class imbalance problem.

Label Assignment: For each image, each anchor is assigned a length K vector of classification targets and a 4-vector of box regression targets. We use the assignment rule from RPN [92] but with adjusted thresholds for object detection (RPN was optimized for proposal generation). Specifically, anchors are assigned to ground-truth boxes using an intersection-over-union (iou) threshold of 0.5; and to background if their iou is in [0, 0.4). As each anchor is assigned to at most one ground-truth box, we set the corresponding entry in its length K label vector to 1 and all other entries to 0. If an anchor is unassigned, which may happen with overlap in [0.4, 0.5), then all labels entries are set to -1, which flags it as an ignored example. Box regression targets are computed as the transformation between each anchor and its assigned ground-truth box, or omitted (zero loss) if there is no assignment, just as in RPN.

Optimization: RetinaNet is trained with stochastic gradient descent (SGD). We use synchronized SGD over 8 GPUs with a total of 16 images per minibatch (2 images per GPU). Unless otherwise specified, all models are trained for 90k iteration with an initial learning rate of 0.01, which is then divided by 10 at 60k and again at 80k iterations. Weight decay of 0.0001 and momentum of 0.9 are used. The training loss is the sum the focal loss and the standard smooth L_1 loss used for box regression [41]. Training time ranges between 10 and 35 hours for the models in Table 5.1e.

5.5 Experiments

We present experimental results on the bounding box detection track of the challenging COCO benchmark [69]. For training, we follow common practice [6, 67] and use the COCO `trainval35k` split (union of 80k images from `train` and a random 35k subset of images from the 40k image `val` split). We report lesion and sensitivity studies by evaluating on the `minival` split (the remaining 5k images from `val`). For our main results, we report COCO AP on the `test-dev` split, which has no public labels and requires use of the evaluation server. If accepted, we will post results on `test-std` to the leaderboard, as recommended.

5.5.1 Training Dense Detection

We run numerous experiments to analyze the behavior of the loss function for dense detection along with various optimization strategies. For all experiments we use depth 50 or 101 ResNets [48] with a Feature Pyramid Network (FPN) [67]

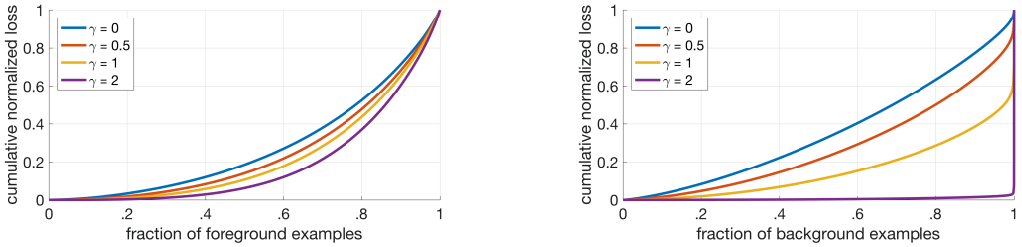


Figure 5.3: Cumulative distribution functions of the normalized loss for positive and negative samples for different values of γ for a *converged* model. The effect of changing γ on the distribution of the loss for positive examples is minor. For negatives, however, increasing γ heavily concentrates the loss on hard examples, focusing nearly all attention away from easy negatives.

constructed on top. For all ablation studies we use an image scale of 600 pixels for training and testing.

Network Initialization: Our first attempt to train RetinaNet uses standard cross entropy (CE) loss without any modifications to the initialization or learning strategy. This fails quickly, with the network diverging during training. However, simply initializing the last layer of our model such that the prior probability of detecting an object is $\pi = .01$ (see §5.4.1) enables effective learning. Training RetinaNet with ResNet-50 and this initialization already yields a respectable AP of 30.2 on COCO. Results are insensitive to the exact value of π so we use $\pi = .01$ for all experiments.

Balanced Cross Entropy: Our next attempt to improve learning involved using the α-balanced CE loss (CE_α) described in §5.3.1. Results for various α are shown in Table 5.1a. Setting $\alpha = .75$ gives a gain of 0.9 points AP.

Focal Loss: Results using our proposed focal loss are shown in Table 5.1c. The focal loss introduces one new hyperparameter, the closing rate of the gate γ, that controls the strength of the gating mechanism. When $\gamma = 0$, our loss is equivalent to the CE loss. As γ increases, the shape of the loss changes so that

"easy" examples with low loss get further discounted, see Figure 5.1. FL shows large gains over CE as γ is increased. With $\gamma = 2$, FL *yields a 2.9 AP improvement over the standard CE loss*.

For the experiments in Table 5.1c, for a fair comparison we find the best α for each γ. We observe that lower α's are selected for higher γ's (as easy negatives are down-weighted, less emphasis needs to be placed on the positives). Overall, however, the benefit of changing γ is much larger, and indeed the best α's ranged in just [.25,.75] (we tested $\alpha \in [.01, .999]$). We use $\gamma = 2.0$ with $\alpha = .25$ for all experiments but $\alpha = .5$ works nearly as well (.4 AP lower).

Analysis of the Focal Loss: To understand the FL better, we analyze the empirical distribution of the loss of a *converged* model. For this, we take take our default ResNet-101 600 pixel model trained with $\gamma = 2$ (which achieves 36.0 AP). We apply this model to a large number of random images and sample the predicted probability for ~10^7 negative windows and ~10^5 positive windows. Next, separately for positives and negatives, we compute the FL for these samples, and normalize the loss such that it sums to one. Given the normalized loss, we can sort the loss from lowest to highest and plot its cumulative distribution function (CDF) for both positive and negative samples and for different settings for γ (even though model was trained w $\gamma = 2$).

Cumulative distribution functions for positive and negative samples are shown in Figure 5.3. If we observe the positive samples, we see that the CDF looks fairly similar for different values of γ. For example, approximately 20% of the hardest positive samples account for roughly half of the positive loss, as γ increases more of the loss gets concentrated in the top 20% of examples, but the effect is minor.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [48]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [67]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [55]	Inception-ResNet-v2 [115]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [108]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [90]	DarkNet-19 [90]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [71, 34]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [34]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	**38.1**	**57.9**	**41.2**	**20.1**	**41.3**	49.8

Table 5.2: **Object detection** *single-model* results (bounding box AP), *v.s.*state-of-the-art on COCO `test-dev`. We show results for our ResNet-101 model that operates on 800 pixel images, trained for 50% longer than the same model from Table 5.1e. Our model achieves top results, outperforming both one-stage and two-stage models. For a detailed breakdown of speed versus accuracy see Table 5.1e and Figure 5.4.

The effect of γ on negative samples is dramatically different. For $\gamma = 0$, the positive and negative CDFs are quite similar. However, as γ increases, substantially more weight becomes concentrated on the hard negative examples. In fact, with $\gamma = 2$ (our default setting), the vast majority of the loss comes from a small fraction of samples. As can be seen, FL can effectively discount the effect of easy negatives, focusing all attention on the hard negative samples.

Online Hard Example Mining (OHEM): [107] proposed to improve training of two-stage object detectors by constructing mini-batches using high-loss examples. Specifically, in OHEM each example is assigned a score based on its loss, Non-Maximum Suppression (nms) is then applied, and a batch is constructed with the highest-scoring examples. The nms threshold and batch size are tunable parameters. Like our focal loss, OHEM puts more emphasis on misclassified examples. Unlike our approach, it completely discards easy examples. We adapt OHEM to our setting of one-stage detection which has extreme class imbalance.

We additionally implement a variant of OHEM motivated by SSD [71]: after applying nms to all examples, the mini-batch is constructed to enforce a 1:3 ratio between positives and negatives. The motivation for this is to help ensure each mini batch has sufficient positives. Results for the original OHEM strategy and the 'OHEM 1:3' strategy for selected batch sizes and nms thresholds are shown in Table 5.1d. These results use ResNet-101, our baseline trained with FL achieves 36.0 AP for this setting. In contrast, the best setting for OHEM (no 1:3 ratio, batch size 128, nms of .5) achieves 32.8 AP. This is a gap of 3.2 AP, showing FL is much more effective than OHEM for training dense detectors. We note that we tried other parameter setting and variants for OHEM but did not achieve better results.

Hinge Loss: Finally, in early experiments, we attempted to train with the hinge loss [45], which sets loss to 0 beyond a certain value of the loss. However, this was unstable and we did not manage to obtain meaningful results.

5.5.2 Model Architecture Design

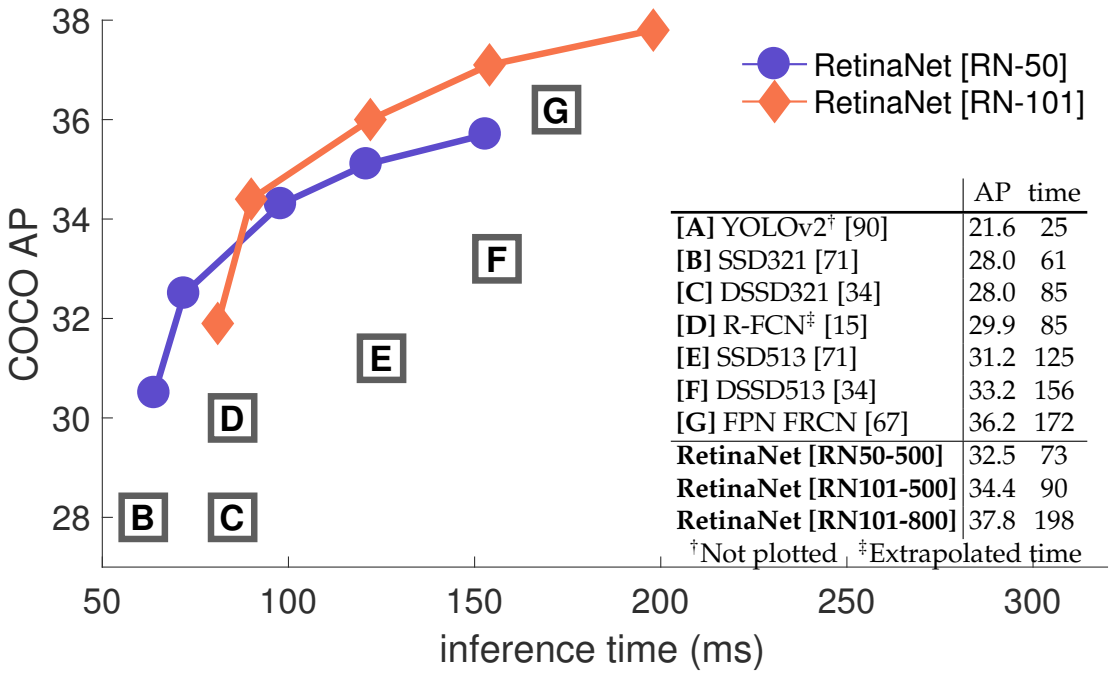
Anchor Density: One of the most important design factors in a one-stage detection system is how densely it covers the space of possible image rectangles. Two-stage detectors can classify rectangles at any position, scale, and aspect ratio using a region pooling operation [41]. In contrast, as one-stage detectors use a fixed sampling grid, a popular approach for achieving high coverage of image rectangles in these approaches is to use multiple 'anchors' at each spatial position to cover boxes of various scales and aspect ratios.

We sweep over the number of scale and aspect ratio anchors used at each

spatial position and each pyramid level in FPN. We consider cases from a single square anchor at each position to 12 anchors per position spanning 4 sub-octave scales ($2^{k/4}$, for $k \leq 3$) and 3 aspect ratios (0.5, 1, 2). Results using ResNet-50 are shown in Table 5.1b. A surprisingly good AP (30.3) is achieved using just one square anchor. However, the AP can be improved by nearly 4 points (to 34.0) when using 3 scales and 3 aspect ratios per position. We used this setting for all other experiments in this work.

Finally, we note that increasing beyond 6-9 anchors did not shown further gains. Thus while two-stage systems can classify arbitrary rectangles in an image, the saturation of performance w.r.tdensity implies the higher potential density of two-stage systems may not offer an advantage.

Speed versus Accuracy: Larger backbone networks yield higher accuracy, but also slower inference speeds. Likewise for input image resolution. We show the impact of these two factors in Table 5.1e. In Figure 5.4 we plot the speed/accuracy trade-off curve for RetinaNet and compare it to recent methods using public numbers on COCO `test-dev`. The plot reveals that RetinaNet, enabled by our focal loss, forms an upper envelope over all existing methods, discounting the low-accuracy regime. Remarkably, RetinaNet with ResNet-101-FPN and a 600 pixel image scale matches the accuracy of the recently published ResNet-101-FPN Faster R-CNN [67], while running in 122 ms per image compared to 240 ms (both measured on an Nvidia M40 GPU). Using larger image sizes allows RetinaNet to surpass the accuracy of all two-stage approaches, while still being faster. At the higher speed side of the curve there is only one operating point (500 pixel input) at which the ResNet-50-FPN backbone dominates the ResNet-101-FPN backbone. Addressing the high frame rate



	AP	time
[A] YOLOv2† [90]	21.6	25
[B] SSD321 [71]	28.0	61
[C] DSSD321 [34]	28.0	85
[D] R-FCN‡ [15]	29.9	85
[E] SSD513 [71]	31.2	125
[F] DSSD513 [34]	33.2	156
[G] FPN FRCN [67]	36.2	172
RetinaNet [RN50-500]	32.5	73
RetinaNet [RN101-500]	34.4	90
RetinaNet [RN101-800]	37.8	198

†Not plotted ‡Extrapolated time

Figure 5.4: Speed (ms) versus accuracy (AP) on COCO `test-dev`. Enabled by the focal loss, our simple one-stage *RetinaNet* detector outperforms all previous one-stage and two-stage detectors, including the best reported Faster R-CNN [92] system from [67]. We show variants of RetinaNet with ResNet-50 (blue circles) and RN-101 (orange diamonds) each at five scales (400-800 pixels). Ignoring the low-accuracy regime (AP<25), RetinaNet forms an upper envelop over all existing detectors. Details are given in §5.5.

regime will likely require special network architecture design, as in [90], rather than use of an off-the-shelf model and is beyond the scope of this work.

5.5.3 Comparison to State of the Art

We evaluate RetinaNet on the bounding box detection task of the challenging COCO dataset and compare `test-dev` results to recent state-of-the-art methods including both one-stage and two-stage models. Results are presented in Table 5.2. Inference times are not available for all models (*e.g.*,, [108]), so we focus on accuracy. Compared to existing one-stage methods, our approach achieves

a healthy 4.9 point gap (38.1 *v.s.*33.2) with the closest competitor, DSSD [34], while also being faster (see Figure 5.4). Compared to recent two-stage methods, RetinaNet achieves a 1.3 point gap above the top-performing TDM-based Faster R-CNN model [108].

5.6 Conclusion

In this work, we identify class imbalance as the primary obstacle preventing one-stage object detectors from surpassing top-performing, two-stage methods, such as Faster R-CNN variants. Motivated by classic work in statistics on loss function shaping, we propose the focal loss which applies a gating mechanism to cross entropy loss in order to modulate its effect on easy examples. This intuitive approach is highly effective at addressing the class imbalance problem in object detection. We demonstrate its efficacy by designing a simple and intuitive one-stage detector and report extensive experimental analysis showing that it achieves state-of-the-art accuracy and run time on the challenging COCO dataset.

CHAPTER 6

CONCLUSION

We discuss the challenges of detecting common objects in everyday scenes. We approach the challenges through the combined efforts of dataset creation and algorithm designs. We create COCO dataset that enables research on detecting objects in non-iconic views, recognizing objects in context, and precise 2D localization. We propose using top-down and lateral connections to generate semantic strong multiscale representations. The SharpMask is proposed to accurately localize instance with detailed pixel segmentation masks. Using the same intuition, FPN is designed and demonstrates the ability to learn generic multiscale feature representations. We show that FPN can be applied for various object detection applications, including instance segmentation, box localization, object proposals, and greatly improve performance for both speed and accuracy of existing methods. Finally, we identify the extreme class imbalance is an inherent challenge for training object detectors and propose Focal Loss to address the issue. Combining the focal loss and FPN, we design a single-stage dense object detector RetinaNet, which achieves state-of-the-art performance for both speed and accuracy on COCO dataset.

APPENDIX A

ANNOTATION USER INTERFACES

We describe and visualize our user interfaces for collecting non-iconic images, category labeling, instance spotting, instance segmentation, segmentation verification and finally crowd labeling.

Non-iconic Image Collection Flickr provides a rich image collection associated with text captions. However, captions might be inaccurate and images may be iconic. To construct a high-quality set of non-iconic images, we first collected candidate images by searching for pairs of object categories, or pairs of object and scene categories. We then created an AMT filtering task that allowed users to remove invalid or iconic images from a grid of 128 candidates, Fig. A.1. We found the choice of instructions to be crucial, and so provided users with examples of iconic and non-iconic images. Some categories rarely co-occurred with others. In such cases, we collected candidates using only the object category as the search term, but apply a similar filtering step, Fig. A.1(b).

Category Labeling Fig. A.3(a) shows our interface for category labeling. We designed the labeling task to encourage workers to annotate all categories present in the image. Workers annotate categories by dragging and dropping icons from the bottom category panel onto a corresponding object instance. Only a single instance of each object category needs to be annotated in the image. We group icons by the super-categories from Fig. A.2, allowing workers to quickly skip categories that are unlikely to be present.

Instance Spotting Fig. A.3(b) depicts our interface for labeling all instances

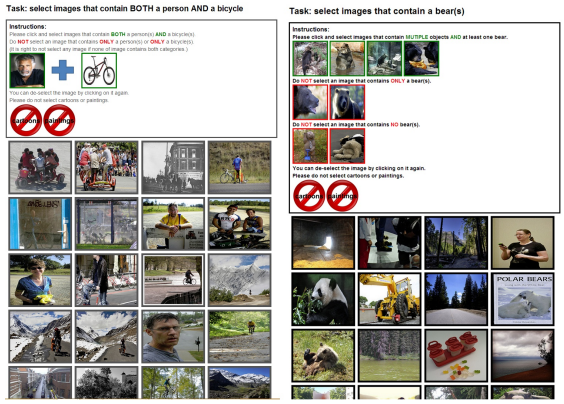


Figure A.1: User interfaces for non-iconic image collection. (a) Interface for selecting non-iconic images containing pairs of objects. (b) Interface for selecting non-iconic images for categories that rarely co-occurred with others.

of a given category. The interface is initialized with a blinking icon specifying a single instance obtained from the previous category-labeling stage. Workers are then asked to spot and click on up to 10 total instances of the given category, placing a single cross anywhere within the region of each instance. In order to spot small objects, we found it crucial to include a "magnifying glass" feature that doubles the resolution of a worker's currently selected region.

Instance Segmentation Fig. A.3(c) shows our user interface for instance segmentation. We modified source code from the OpenSurfaces project [5], which defines a single AMT task for segmenting multiple regions of a homogenous material in real-scenes. In our case, we define a single task for segmenting a single object instance labeled from the previous annotation stage. To aid the segmentation process, we added a visualization of the object category icon to remind workers of the category to be segmented. Crucially, we also added zoom-in functionality to allow for efficient annotation of small objects and curved boundaries. In the previous annotation stage, to ensure high coverage of all object instances, we used multiple workers to label all instances per image. We

would like to segment *all* such object instances, but instance annotations across different workers may refer to different or redundant instances. To resolve this correspondence ambiguity, we sequentially post AMT segmentation tasks, ignoring instance annotations that are already covered by an existing segmentation mask.

Segmentation Verification Fig. A.3(d) shows our user interface for segmentation verification. Due to the time consuming nature of the previous task, each object instance is segmented only once. The purpose of the verification stage is therefore to ensure that each segmented instance from the previous stage is of sufficiently high quality. Workers are shown a grid of 64 segmentations and asked to select poor quality segmentations. Four of the 64 segmentation are known to be bad; a worker must identify 3 of the 4 known bad segmentations to complete the task. Each segmentation is initially shown to 3 annotators. If any of the annotators indicates the segmentation is bad, it is shown to 2 additional workers. At this point, any segmentation that doesn't receive at least 4 of 5 favorable votes is discarded and the corresponding instance added back to the pool of unsegmented objects. Examples of borderline cases that either passed (4/5 votes) or were rejected (3/5 votes) are shown in Fig. B.3.

Crowd Labeling Fig. A.3(e) shows our user interface for crowd labeling. As discussed, for images containing ten object instances or fewer of a given category, every object instance was individually segmented. In some images, however, the number of instances of a given category is much higher. In such cases crowd labeling provided a more efficient method for annotation. Rather than requiring workers to draw exact polygonal masks around each object instance, we allow workers to "paint" all pixels belonging to the category in question.

Crowd labeling is similar to semantic segmentation as object instance are not individually identified. We emphasize that crowd labeling is only necessary for images containing more than ten object instances of a given category.

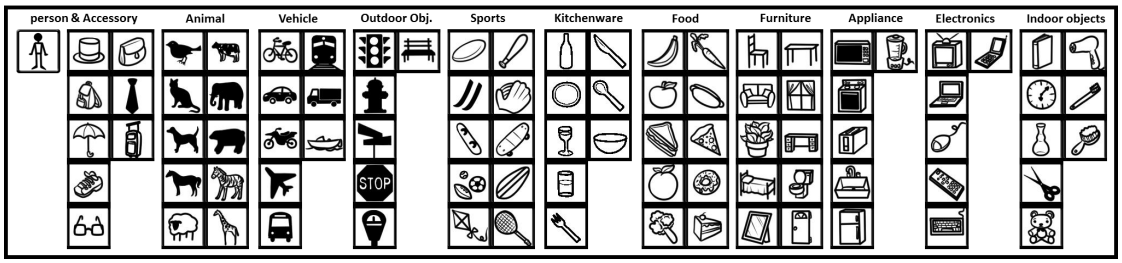


Figure A.2: Icons of 91 categories in the COCO dataset grouped by 11 super-categories. We use these icons in our annotation pipeline to help workers quickly reference the indicated object category.

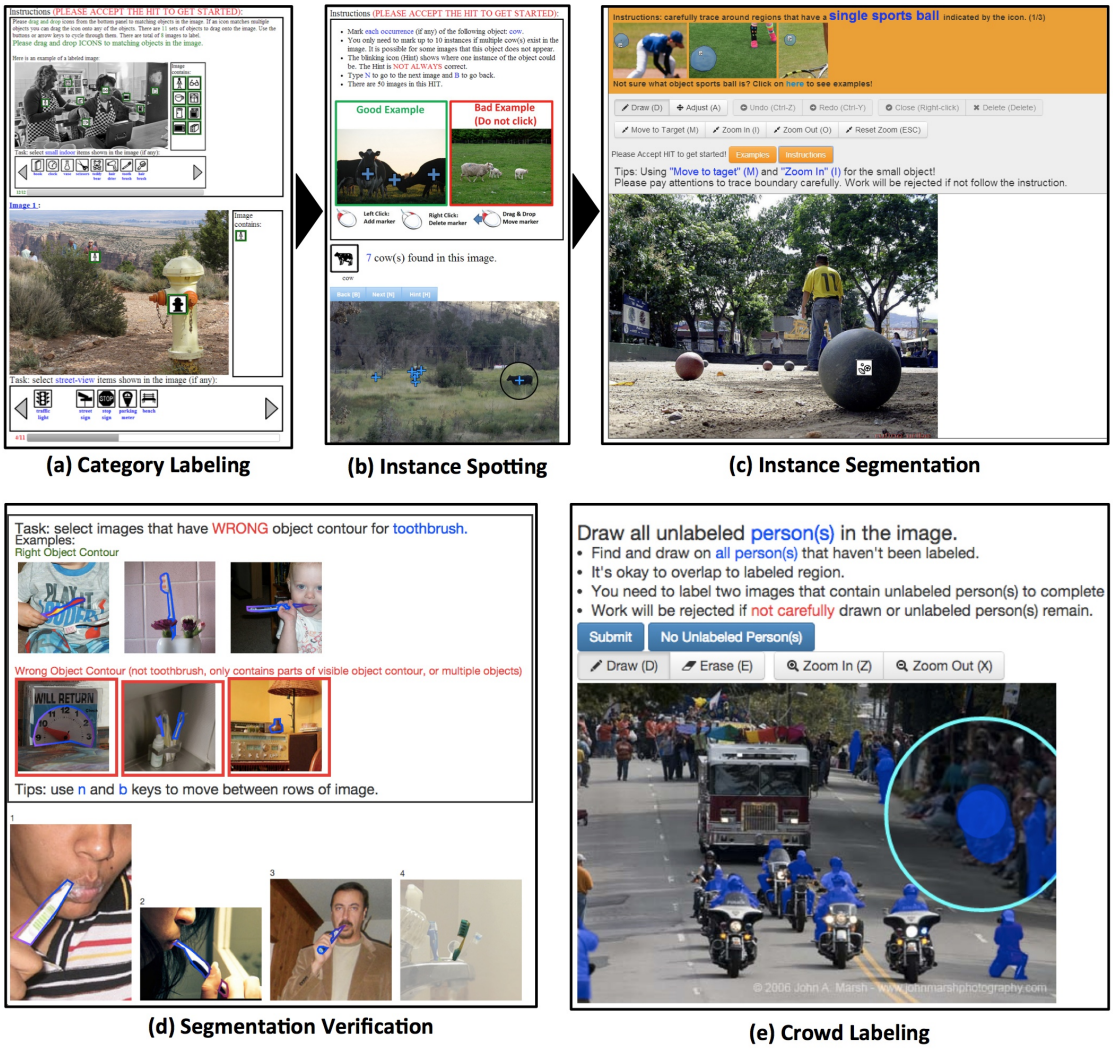


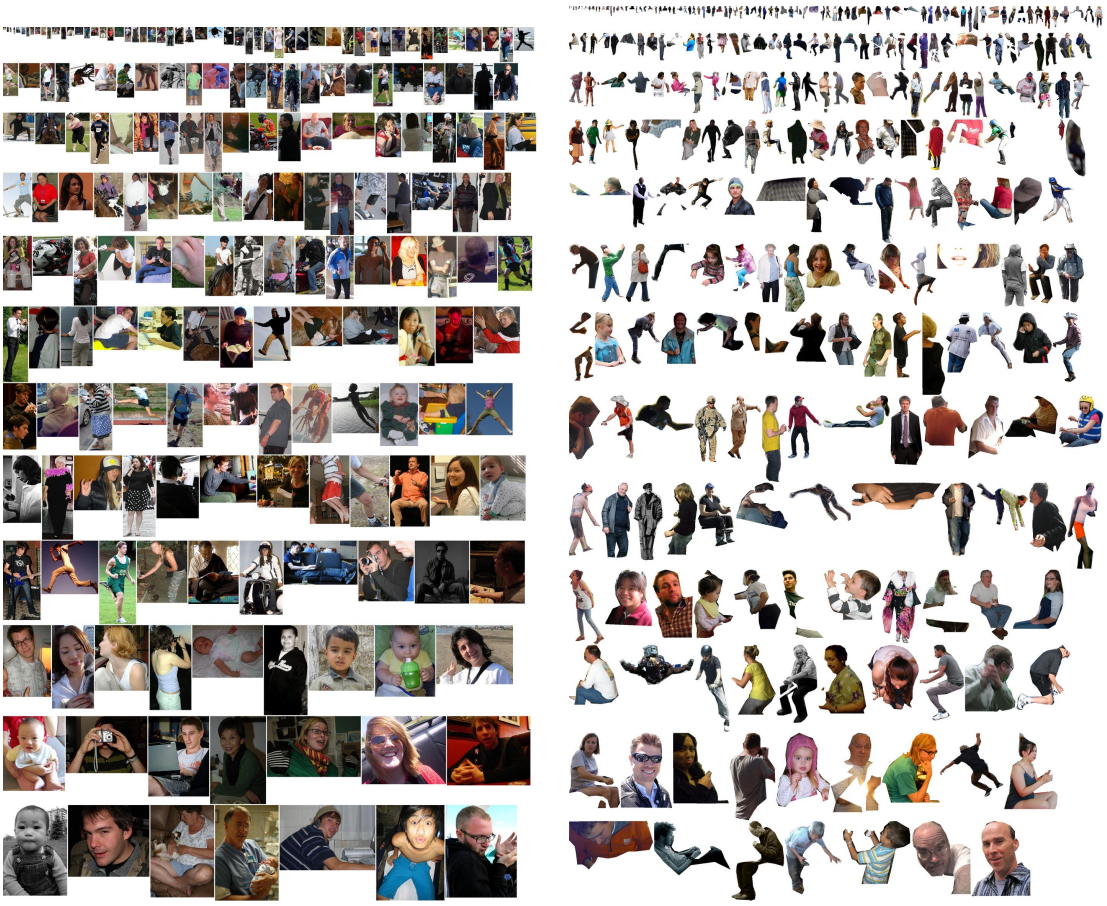
Figure A.3: User interfaces for collecting instance annotations, see text for details.

APPENDIX B

OBJECT & SCENE CATEGORIES IN COCO DATASET

Our dataset contains 91 object categories (the 2014 release contains segmentation masks for 80 of these categories). We began with a list of frequent object categories taken from WordNet, LabelMe, SUN and other sources as well as categories derived from a free recall experiment with young children. The authors then voted on the resulting 272 categories with the aim of sampling a diverse and computationally challenging set of categories; see §2.3 for details. The list in Table B.1 enumerates those 272 categories in descending order of votes. As discussed, the final selection of 91 categories attempts to pick categories with high votes, while keeping the number of categories per super-category (animals, vehicles, furniture, etc.) balanced.

As discussed in §2.3, in addition to using object-object queries to gather non-iconic images, object-scene queries also proved effective. For this task we selected a subset of 40 scene categories from the SUN dataset that frequently co-occurred with object categories of interest. Table B.2 enumerates the 40 scene categories (evenly split between indoor and outdoor scenes).



(a) PASCAL VOC. (b) COCO.

Figure B.1: Random person instances from PASCAL VOC and COCO. At most one instance is sampled per image.

person	**bicycle**	**car**	**motorcycle**	**bird**	**cat**	**dog**	**horse**	**sheep**	**bottle**
chair	**couch**	**potted plant**	**tv**	**cow**	**airplane**	**hat***	license plate	**bed**	**laptop**
fridge	**microwave**	**sink**	**oven**	**toaster**	**bus**	**train**	**mirror***	**dining table**	**elephant**
banana	bread	**toilet**	**book**	**boat**	**plate***	**cell phone**	**mouse**	**remote**	**clock**
face	hand	**apple**	**keyboard**	**backpack**	steering wheel	**wine glass**	chicken	**zebra**	**shoe***
eye	mouth	**scissors**	**truck**	**traffic light**	**eyeglasses***	**cup**	**blender***	**hair drier**	wheel
street sign*	**umbrella**	**door***	**fire hydrant**	**bowl**	teapot	**fork**	**knife**	**spoon**	**bear**
headlights	**window***	**desk***	computer	**refrigerator**	**pizza**	squirrel	duck	**frisbee**	guitar
nose	**teddy bear**	**tie**	**stop sign**	**surfboard**	**sandwich**	pen/pencil	**kite**	**orange**	**toothbrush**
printer	pans	head	**sports ball**	**broccoli**	**suitcase**	**carrot**	chandelier	**parking meter**	fish
handbag	**hot dog**	stapler	basketball hoop	**donut**	**vase**	**baseball bat**	**baseball glove**	**giraffe**	jacket
skis	**snowboard**	table lamp	egg	door handle	power outlet	hair	tiger	table	coffee table
skateboard	helicopter	tomato	tree	bunny	pillow	**tennis racket**	**cake**	feet	**bench**
chopping board	washer	lion	monkey	**hair brush***	light switch	arms	legs	house	cheese
goat	magazine	key	picture frame	cupcake	fan (ceil/floor)	frogs	rabbit	owl	scarf
ears	home phone	pig	strawberries	pumpkin	van	kangaroo	rhinoceros	sailboat	deer
playing cards	towel	hyppo	can	dollar bill	doll	soup	meat	window	muffins
tire	necklace	tablet	corn	ladder	pineapple	candle	desktop	carpet	cookie
toy cars	bracelet	bat	balloon	gloves	milk	pants	wheelchair	building	bacon
box	platypus	pancake	cabinet	whale	dryer	torso	lizard	shirt	shorts
pasta	grapes	shark	swan	fingers	towel	side table	gate	beans	flip flops
moon	road/street	fountain	fax machine	bat	hot air balloon	cereal	seahorse	rocket	cabinets
basketball	telephone	movie (disc)	football	goose	long sleeve shirt	short sleeve shirt	raft	rooster	copier
radio	fences	goal net	toys	engine	soccer ball	field goal posts	socks	tennis net	seats
elbows	aardvark	dinosaur	unicycle	honey	legos	fly	roof	baseball	mat
ipad	iphone	hoop	hen	back	table cloth	soccer nets	turkey	pajamas	underpants
goldfish	robot	crusher	animal crackers	basketball court	horn	firefly	armpits	nectar	super hero costume
jetpack	robots								

Table B.1: Candidate category list (272). **Bold**: selected categories (91). **Bold***: omitted categories in 2014 release (11).

(a) PASCAL VOC. (b) COCO.

Figure B.2: Random chair instances from PASCAL VOC and COCO. At most one instance is sampled per image.

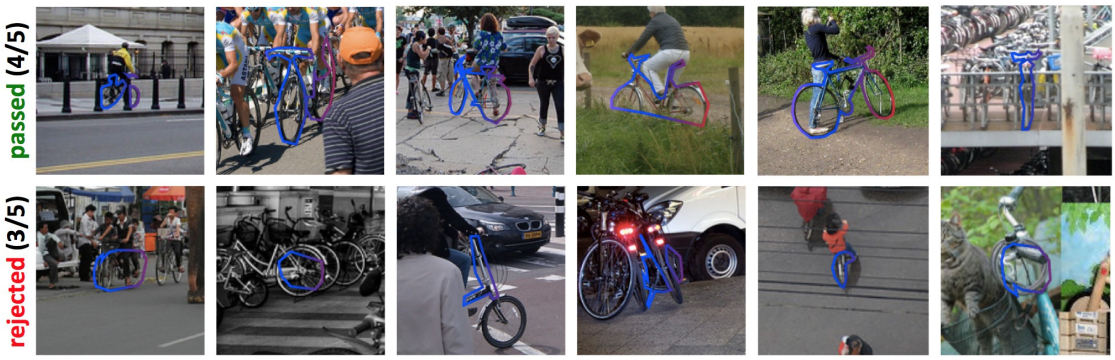


Figure B.3: Examples of borderline segmentations that passed (top) or were rejected (bottom) in the verification stage.

library	church	office	restaurant	kitchen	living room	bathroom	factory	campus	bedroom
child's room	dining room	auditorium	shop	home	hotel	classroom	cafeteria	hospital room	food court
street	park	beach	river	village	valley	market	harbor	yard	parking lot
lighthouse	railway	playground	swimming pool	forest	gas station	garden	farm	mountain	plaza

Table B.2: Scene category list.

BIBLIOGRAPHY

[1] Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. Pyramid methods in image processing. *RCA engineer*, 1984.

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

[3] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *PAMI*, 33(5):898–916, 2011.

[4] S. Baker, D. Scharstein, J.P. Lewis, S. Roth, M.J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. *IJCV*, 92(1):1–31, 2011.

[5] S. Bell, P. Upchurch, N. Snavely, and K. Bala. OpenSurfaces: A richly annotated catalog of surface appearance. *SIGGRAPH*, 32(4), 2013.

[6] Sean Bell, C. Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural nets. In *CVPR*, 2016.

[7] T. Berg and A. Berg. Finding iconic images. In *CVPR*, 2009.

[8] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3D human pose annotations. In *ICCV*, 2009.

[9] G.J. Brostow, J. Fauqueur, and R. Cipolla. Semantic object classes in video: A high-definition ground truth database. *PRL*, 30(2):88–97, 2009.

[10] T. Brox, L. Bourdev, S. Maji, and J. Malik. Object segmentation by alignment of poselet activations to image contours. In *CVPR*, 2011.

[11] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV*, 2016.

[12] LC. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep conv. nets and fully connected CRFs. In *ICLR*, 2015.

[13] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *ECCV*, 2016.

[14] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *CVPR*, 2016.

[15] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. In *NIPS*, 2016.

[16] Q. Dai and D. Hoiem. Learning to localize detected objects. In *CVPR*, 2012.

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

[18] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[19] J. Deng, O. Russakovsky, J. Krause, M. Bernstein, A. Berg, and L. Fei-Fei. Scalable multi-label annotation. In *CHI*, 2014.

[20] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *PAMI*, 34, 2012.

[21] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *TPAMI*, 2014.

[22] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. 2009.

[23] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015.

[24] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid. Evaluation of gist descriptors for web-scale image search. In *CIVR*, 2009.

[25] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, 2015.

[26] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. In *CVPR*, 2014.

[27] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL visual object classes (VOC) challenge. *IJCV*, 2010.

[28] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *PAMI*, 2013.

[29] A. Farhadi, I. Endres, D. Hoiem, and David Forsyth. Describing objects by their attributes. In *CVPR*, 2009.

[30] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPR Workshop of Generative Model Based Vision (WGMBV)*, 2004.

[31] C. Fellbaum. *WordNet: An electronic lexical database*. Blackwell Books, 1998.

[32] Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. Cascade object detection with deformable part models. In *CVPR*, 2010.

[33] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *TPAMI*, 2010.

[34] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD: Deconvolutional single shot detector. *arXiv:1701.06659*, 2016.

[35] Golnaz Ghiasi and Charless C Fowlkes. Laplacian pyramid reconstruction and refinement for semantic segmentation. In *ECCV*, 2016.

[36] Spyros Gidaris and Nikos Komodakis. Object detection via a multi-region & semantic segmentation-aware CNN model. In *ICCV*, 2015.

[37] Spyros Gidaris and Nikos Komodakis. Attend refine repeat: Active box proposal generation via in-out localization. In *BMVC*, 2016.

[38] R. Girshick. Fast R-CNN. In *ICCV*, 2015.

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

[40] R. Girshick, P. Felzenszwalb, and D. McAllester. Discriminatively trained deformable part models, release 5. *PAMI*, 2012.

[41] Ross Girshick. Fast R-CNN. In *ICCV*, 2015.

[42] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

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

[44] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *CVPR*, 2015.

[45] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer series in statistics Springer, Berlin, 2008.

[46] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv:1703.06870*, 2017.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*. 2014.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[49] G. Heitz and D. Koller. Learning spatial context: Using stuff to find things. In *ECCV*, 2008.

[50] E. Hjelms and B.K. Low. Face detection: A survey. *CVIU*, 83(3):236–274, 2001.

[51] . Hoiem, D, Y. Chodpathumwan, and Q. Dai. Diagnosing error in object detectors. In *ECCV*, 2012.

[52] Sina Honari, Jason Yosinski, Pascal Vincent, and Christopher Pal. Recombinator networks: Learning coarse-to-fine feature aggregation. In *CVPR*, 2016.

[53] J. Hosang, R. Benenson, P. Dollár, and B. Schiele. What makes for effective detection proposals? *PAMI*, 2015.

[54] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[55] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv:1611.10012*, 2016.

[56] A. Humayun, F. Li, and J. M. Rehg. RIGOR: Reusing Inference in Graph Cuts for generating Object Regions. In *CVPR*, 2014.

[57] Tao Kong, Anbang Yao, Yurong Chen, and Fuchun Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *CVPR*, 2016.

[58] P. Krähenbühl and V. Koltun. Geodesic object proposals. In *ECCV*, 2014.

[59] Ivan Krasin, Tom Duerig, Neil Alldrin, Andreas Veit, Sami Abu-El-Haija, Serge Belongie, David Cai, Zheyun Feng, Vittorio Ferrari, Victor Gomes, Abhinav Gupta, Dhyanesh Narayanan, Chen Sun, Gal Chechik, and Kevin Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages*, 2016.

[60] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[61] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[62] C.H. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *CVPR*, 2009.

[63] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[64] Y. Lecun and C. Cortes. The MNIST database of handwritten digits, 1998.

[65] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[66] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common objects in context. *arXiv:1405.0312*, 2015.

[67] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. *arXiv:1612.03144*, 2016.

[68] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for object detection. In *ICCV*, 2017.

[69] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[70] T.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[71] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, and Scott Reed. SSD: Single shot multibox detector. In *ECCV*, 2016.

[72] Wei Liu, Andrew Rabinovich, and Alexander C Berg. ParseNet: Looking wider to see better. In *ICLR workshop*, 2016.

[73] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[74] David G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[75] S. A. Nene, S. K. Nayar, and H. Murase. Columbia object image library (coil-20). Technical report, Columbia Universty, 1996.

[76] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016.

[77] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015.

[78] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42:145–175, 2001.

[79] V. Ordonez, J. Deng, Y. Choi, A. Berg, and T. Berg. From large scale image categorization to entry-level categories. In *ICCV*, 2013.

[80] V. Ordonez, G. Kulkarni, and T. Berg. Im2text: Describing images using 1 million captioned photographs. In *NIPS*, 2011.

[81] S. Palmer, E. Rosch, and P. Chase. Canonical perspective and the perception of objects. *Attention and performance IX*, 1:4, 1981.

[82] G. Patterson and J. Hays. SUN attribute database: Discovering, annotating, and recognizing scene attributes. In *CVPR*, 2012.

[83] P. O. Pinheiro and R. Collobert. Recurrent conv. neural networks for scene labeling. In *ICML*, 2014.

[84] Pedro O Pinheiro, Ronan Collobert, and Piotr Dollar. Learning to segment object candidates. In *NIPS*, 2015.

[85] Pedro O Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to refine object segments. In *ECCV*, 2016.

[86] J. Pont-Tuset, P. Arbeláez, J. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping for image segmentation and object proposal gen. *PAMI*, 2015.

[87] D. Ramanan. Using segmentation to verify object hypotheses. In *CVPR*, 2007.

[88] C. Rashtchian, P. Young, M. Hodosh, and J. Hockenmaier. Collecting image annotations using Amazon's Mechanical Turk. In *NAACL Workshop*, 2010.

[89] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[90] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. *arXiv:1612.08242*, 2017.

[91] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[92] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[93] Shaoqing Ren, Kaiming He, Ross Girshick, Xiangyu Zhang, and Jian Sun. Object detection networks on convolutional feature maps. *PAMI*, 2016.

[94] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[95] H.A. Rowley, S. Baluja, and T. Kanade. Human face detection in visual scenes. Technical Report CMU-CS-95-158R, Carnegie Mellon University, 1995.

[96] O. Russakovsky, J. Deng, Z. Huang, A. Berg, and L. Fei-Fei. Detecting avocados to zucchinis: what have we done, and where are we going? In *ICCV*, 2013.

[97] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

[98] B.C. Russell, A. Torralba, K.P. Murphy, and W.T. Freeman. LabelMe: a database and web-based tool for image annotation. *IJCV*, 77(1-3):157–173, 2008.

[99] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47(1-3):7–42, 2002.

[100] Alexander G Schwing and Raquel Urtasun. Fully connected deep structured networks. *arXiv:1503.02351*, 2015.

[101] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR*, 2006.

[102] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using conv nets. In *ICLR*, 2014.

[103] P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, 2013.

[104] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014.

[105] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *CVPR*, 2008.

[106] J. Shotton, J. Winn, C. Rother, and A. Criminisi. TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *IJCV*, 81(1):2–23, 2009.

[107] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, 2016.

[108] Abhinav Shrivastava, Rahul Sukthankar, Jitendra Malik, and Abhinav Gupta. Beyond skip connections: Top-down modulation for object detection. *arXiv:1612.06851*, 2016.

[109] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from RGBD images. In *ECCV*, 2012.

[110] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[111] Rebecca Sitton. *Spelling Sourcebook*. Egger Publishing, 1996.

[112] Kah-Kay Sung and Tomaso Poggio. Learning and Example Selection for Object and Pattern Detection. In *MIT A.I. Memo No. 1521*, 1994.

[113] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[114] C. Szegedy, S. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. *arXiv:1412.1441*, 2014.

[115] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv:1602.07261*, 2016.

[116] A. Torralba and A. Efros. Unbiased look at dataset bias. In *CVPR*, 2011.

[117] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *PAMI*, 30(11):1958–1970, 2008.

[118] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *IJCV*, 2013.

[119] R. Vaillant, C. Monrocq, and Y. LeCun. Original approach for the localisation of objects in images. *IEE Proc. on Vision, Image, and Signal Processing*, 1994.

[120] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[121] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-201, Caltech, 2010.

[122] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010.

[123] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *ICCV*, 2015.

[124] Y. Yang, S. Hallman, D. Ramanan, and C. Fowlkes. Layered object models for image segmentation. *PAMI*, 34(9):1731–1743, 2012.

[125] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[126] Sergey Zagoruyko, Adam Lerer, Tsung-Yi Lin, Pedro O Pinheiro, Sam Gross, Soumith Chintala, and Piotr Dollár. A multipath network for object detection. In *BMVC*, 2016.

[127] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *CVPR*, 2010.

[128] S. Zheng, S. Jayasumana, B. Romera-Paredes, B. Vineet, Z. Su, D. Du, C. Huang, and P. Torr. Conditional random fields as recurrent neural nets. In *ICCV*, 2015.

[129] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014.

[130] X. Zhu, C. Vondrick, D. Ramanan, and C. Fowlkes. Do we need more training data or better models for object detection? In *BMVC*, 2012.

[131] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*, 2014.