Dissertation

Reliable Company Logo Detection with Deep Convolutional Neural Networks

Addressing the Small Object Detection Problem

Christian Eggert

February 2019

Department of Computer Science
University of Augsburg

Advisor:	Prof. Dr. Rainer Lienhart
Reviewers:	Prof. Dr. Rainer Lienhart
	Prof. Dr. Björn Schuller
Thesis defense:	February 8th, 2019
Examiners:	Prof. Dr. Rainer Lienhart
	Prof. Dr. Björn Schuller
	Prof. Dr. Elisabeth André

Abstract

Many companies conduct market research to gather information about their customers. Among other things, they are often interested in information about customer expectations, customer satisfaction, brand popularity and brand perception. A classical instrument of market research to gather such data are surveys.

While surveys undeniably offer valuable insights into many aspects of customer relations, they are time-consuming and expensive to conduct. Because of these limitations, surveys can often be only conducted on a comparatively small sample of people. Therefore, market research organizations are always interested in obtaining data through different channels. The rise of social media has opened up such a channel. With cell phone cameras being ubiquitous nowadays, many people post pictures of their daily activities on social media sites. In doing so, they capture their interactions with certain brands. This capture of brand interaction sometimes is a conscious act, e.g. by showing off a new car to friends or by displaying affection to a certain brand of beer. However, more commonly these brand interactions are captured inadvertently. For example, a self portrait might capture the brand of clothing the user likes to wear. In any case, images on social media provide a useful source of information for market research. The reliable detection of company logos forms an important building block for any further analysis of these brand interactions.

Object detection is a well-studied problem in computer vision. Great advances have been made especially in recent years with the advent of object detection pipelines like R-CNN and Fast(er) R-CNN, which are based on deep convolutional neural networks. In this work, we have applied these new techniques to the problem of company logo detection. We have created a new object detection dataset as a benchmark, specifically targeted for company logo detection. And while on the face of it, company logo detection is nothing but a special case of object detection, we have noticed some peculiarities when applying these new object detection pipelines to our new dataset. We have noticed for example, that while being slower, the oldest approach (R-CNN) performs considerably better on our dataset than newer approaches like Fast(er) R-CNN which are usually considered to be improvements over R-CNN – both in terms of detection performance and speed.

In this work we investigate the reasons for this discrepancy by analyzing each step of these object detection pipelines. In particular, we look at the generation of both heuristic and trainable object proposals and their classification. For heuristic object proposals we look at two commonly used algorithms: Selective Search and Edge Boxes. We observe some conditions under which these algorithms fail that have particular relevance for company logos. In particular we notice that both algorithms struggle to identify proposals for text-based company logos and introduce an additional heuristic to mitigate this problem and demonstrate its effectiveness. For trainable object proposals we look at Region Proposal Networks (RPNs) which we analyze in detail, both theoretically and in in practice and notice some fundamental

shortcomings for detecting small company logos. We introduce some simple modifications and show that these are able to considerably improve the performance of small object proposals.

We also look at the classification stage and identify the receptive field of the network as an important quantity to improve the classification performance. We finally look at SSD, a more modern single-stage approach for object detection which allows us to incorporate all our observations into a single detection framework. Our improvements allow us to improve our object detection pipeline to a point where where we not only exceed the detection performance of R-CNN but can also perform real-time company logo detection.

Acknowledgements

There are many people who deserve credit because they – directly or indirectly – influenced this work and made it possible. Most notably, I want to thank Prof. Lienhart, whose door was always open, for supervising and reviewing this thesis and for providing ideas and criticism. Similarly, I want to express my gratitude towards Prof. Schuller for the time he invested in reviewing this thesis.

Also, I want to thank my colleagues Dan Zecha, Philipp Harzig, Stephan Brehm and Moritz Einfalt for the countless discussions which often provided valuable insights and alternative views on many problems. This also extends to my former colleagues Anton Winschel, Christoph Lassner, Christian Ries, Fabian Richter and Stefan Romberg from whom I learned a lot.

I also want to express my gratitude towards Carolin Kaiser, Holger Dietrich, René Schallner and Raimund Wildner from GfK-Verein who partially funded much of the work that forms the basis of this thesis. They also had a major part in re-annotating the FlickrLogos-32 dataset for object detection.

Finally, I want to thank my family and especially my mother who always supported me during my studies.

To my mother, who always supported me.

Contents

Chapter 1

Introduction

1.1 Motivation and Applications

For many companies, market research is a powerful tool to remain competitive. By analyzing current trends and customer demands, market research allows companies to tune their products to fit the demand of their target markets better.

Market research companies have developed many metrics to gain information on the target markets. One important metric is to track brand popularity among specific population groups – usually defined by gender, age, and income brackets. Another metric is to track whether certain population groups have a predominantly positive or negative perception of a particular brand. Sometimes, market research companies also try to gauge the emotional investment into brands.

In order to apply these metrics, data has to be gathered. Surveys are one of the most important sources of information. However, surveys are costly and time consuming to conduct, which usually only allows questioning a comparatively small sample of people. Therefore market research companies are looking for different sources of information which are more easily accessible.

The recent rise of social media has created a new source of information for market research companies. Combined with cell phone cameras – which are ubiquitous nowadays – many people use services like Twitter, Facebook and Instagram to share pictures of their daily life with each other. By sharing these images, consumers provide market research companies a window into their habits and preferences. Sometimes these preferences are expressed as a conscious act by the user, e.g., by showing off a newly purchased product to their friends. However, most of the time, these preferences are expressed unintentionally, e.g., the brand of clothing a user usually wears. By analyzing the image content, these images can not only convey information about brand preferences and popularity but also about the emotional connection a user has with a particular brand.

The analysis of customer brand relations is not the only field of activity for market researchers. Sometimes, researchers also survey supermarkets to gather information on what brands are carried by which supermarket chains. Many companies are

interested in this information because it allows them to gain knowledge on which competing products are on offer and how they are priced in comparison to their product. This surveying of supermarkets is usually conducted by taking pictures of shelves which are later analyzed manually.

Manually collecting and analyzing these images is a time consuming and expensive process and therefore suffers from many of the same drawbacks as customer surveys. Automating image analysis is, therefore, an essential focus of market research companies. Computer vision can assist market researchers in performing these analyses. However, in this work, we will not look at the automation of the actual analyses but instead focus exclusively on the reliable detection of company logos which forms an important building block for the analyses mentioned above.

1.2 Challenges

Object detection refers to the ability to recognize and localize object instances in a scene. What we mean by *localization* is the identification of an image area containing the object by enclosing it with a tight bounding box. By *recognition* we mean the assignment of this image area to an element in a set of pre-defined classes. An image may contain an arbitrary number of object instances of arbitrarily many classes.

The human brain is able to recognize and localize objects in an instant so that we are tempted to think that this is an easy task. However, when we are pressed to give a precise mechanism by which we recognize objects, we usually are unable to do so. This inability to define a precise function for mapping image content to object classes makes it hard to tackle this problem in a computer.

Even if we disregard the problem of localization, objects can vary strongly in their visual appearance, and the recognition function needs to be able to account for these variations. We can roughly group the variation in appearance into the following four categories:

Variance in Scale A classic problem in object detection that any object detection algorithm needs to overcome is the problem of scale variance (i.e., object instances can vary in size). In the case of company logo detection we have found that these variations in scale can be quite severe: It is not uncommon that the side length of the largest object has 50 times the size of the smallest object. These extreme variations in scale are often due to the fact that some images are taken with the intent of depicting product logos while others are not. In the former case, the logos tend to appear quite large while in the latter case, they can be tiny.

Variation in Pose Another classic problem of object detection is variation in pose. The pose of an object refers to its orientation in 3D space relative to the camera. An example of differences in object pose would be the existence of front and profile views of the same object. Although these views might visually be quite

different, an object detection algorithm should be able to recognize both views of the object as belonging to the same object class. Fortunately, this problem is only of minor significance to company logo detection since most company logos are 2D objects. As such, they are usually found on planar surfaces or surfaces that can be approximated as planar. Differences in pose can therefore often be modeled as a simple perspective transformation.

Truncation and Occlusion Sometimes, object instances are not fully visible. In such a case we refer to the object as *truncated*. Truncation can have multiple reasons: One of the most common reasons is that the object has only been partially caught in the picture frame. *Occlusion* is a special case of truncated where another object is partially blocking the view and therefore truncating the object in question. While the aforementioned situations are commonplace in object detection, we found that another case of truncation is relevant for company logo detection: Since most company logos are 2D objects, they may exist on a curved surface. A typical example of such a situation would be the logo of a brewery on a bottle. In such a case the bottle could be rotated in such a way that the logo is only partially visible.

Intra-class Variance Even objects of the same class can vary in appearance. One example regarding company logos might be varying backgrounds. For example, Coca-Cola logos might be located as advertisements on billboards, which tend to have a white background, or on bottles, which tend to have a dark background. Another example that falls into this category are multiple versions of the same company logo: Companies sometimes re-brand their products by changing the logo associated with the product. Ideally, the recognition function should be able to classify each logo version equally well.

Another source of visual variation is company logos which consist of both a symbol and a text component. Symbol and text often appear in different geometric configurations and sometimes only one of these components is actually present. While it is indeed possible to train a classifier which is able to take all these modes into account, we do not encounter this problem in this work, because we model text-based and symbol-based company logos as separate classes (see Chapter 2.1).

Computational Complexity Since the ultimate goal of company logo detection is to evaluate images from social media, the algorithms need to be able to scale to a potentially large number of images. Therefore, the computational effort to analyze these images becomes an important factor. As we shall see in Chapters 4.2.3 and 4.4 this is especially true when considering small object instances because detecting small object instances is usually associated with more computation.

(a) (b) (c) (d)

Figure 1.1: Challenges of company logo detection. (a) Strong differences in scale. (b) Differences in pose. (c) Truncated logo instances. (d) Intra-class variance.

1.3 Related Work

Object detection is a traditional discipline of computer vision. Long before deep convolutional neural networks made their impact on the computer vision community, much work has been done in this field. The primary challenge of object detection is to take account of the large visual variety of objects, as described in the previous chapter.

Before the advent of deep convolutional neural networks, this was usually difficult to achieve on raw pixel data. Therefore, these early approaches typically relied on hand-crafted image features which helped a classifier to focus on relevant image content.

Features These features could take very different forms, depending on the desired application: For some time, features based on Haar wavelets [32] were popular. Improved features inspired by Haar wavelets were used by [79] and [53] for detecting human faces. In some rather rare instances, texture-based features extracted by Gabor filters [29] were used for detecting humans and parsing body poses [69]. Object shape was also previously used for detection [84] using shape context [6] features. However, the by far most popular features for object detection were gradient histograms, either computed on a dense grid across the whole image [14] or at sparse locations [57]. While the approaches mentioned above can achieve reasonable detection performance in specialized applications, the underlying hand-crafted features limit their general applicability since different objects may be mapped to a similar feature representation. This lack of distinctiveness has been demonstrated to be a problem for gradient histogram based features in [80] who conclude that better learning algorithms or bigger datasets are unlikely to improve the quality of object detectors significantly.

Object models Largely independent from the underlying feature representation is the choice of object model. One of the most simple models consists of a single fixed feature template which is matched to image regions which potentially contain object instances [57, 14]. This approach usually works well for rigid objects whose pose does not vary very strongly. For objects which are not rigid, this approach is unlikely to yield good results, and part-based models have been introduced to address this problem.

Part-based models can roughly be classified as probabilistic models or *pictorial structure*-based models. Probabilistic part-based models – often called *constellation models* [81, 27] – model an object as collections of features which have an expected geometric relation with each other. For these models to be computationally feasible, they usually need to make strong assumptions about the probability distributions which express the relations between features. Models built on the idea of pictorial structures [28] regard objects as being composed of multiple parts (e.g., a face being composed of eyes, nose, mouth, etc.). Each part exists in a preferred geometrical configuration with respect to the other parts but is allowed to deviate from this preferred position. By deviating from the expected position, the model incurs deformation costs. Pictorial structure-based model have been applied to object detection [25] with great success. Deformable part models [24] – which are one of the most successful approaches to object detection prior to the triumph of deep learning – are based on pictorial structures. They restrict the object model to a star-shaped arrangement of parts which can be computed efficiently by employing a distance transform [26].

DCNN-based approaches After the initial success of deep convolutional neural networks for image classification by Krizhevsky et al. [49], new network architectures like VGG16 [72], Inception [74] and Residual Networks [36] were developed. They were able to significantly improve on the results of Krizhevsky et al. Deep neural networks also had a major impact on object recognition because they blur the distinction between feature extraction, object model, and classification. All these previously separate steps are moved into the layers of a single neural network that is jointly optimized.

The high computational cost of evaluating a deep neural network made it infeasible to apply it in a traditional sliding window fashion applied over multiple image scales. Similar to the idea of sparsely distributed interest points, the idea of *object proposals* – a sparse set of bounding boxes describing the outline of potential object instances – was introduced. There exists a large variety of object proposal algorithms based on different principles: Some algorithms are based on rapidly generating multiple object segmentations [11, 21, 63]. Other algorithms start with a single (over-)segmentation of the image and iteratively merge object proposals [59, 77]. Some object proposal algorithms are based on a ranking mechanism which measures the *objectness* [1, 12] of a proposal. In this work, we will only cover two object proposal algorithms in detail (see chapter 3), namely Selective

Search [77] – which is based on merging superpixels – and Edge Boxes [85] – which ranks the *objectness* of a dense grid of proposals.

R-CNN [31] was the first object detection framework which was based on convolutional neural networks and used Selective Search [77] as a source for object proposals. While the use of object proposals made R-CNN computationally feasible, it is comparatively slow in processing images. Subsequent improvements like Fast R-CNN [30] and Faster R-CNN [66] made significant improvements regarding detection speed. Because all of these detection frameworks use object proposals in one way or another, we refer to these class of algorithms as *two-stage object detection* approaches. We will cover these approaches in detail in Chapter 4. Another class of neural network based object detection pipelines do away with object proposals entirely. We refer to these algorithms as *single-stage object detection* approaches which we cover in chapter III in greater detail. The two most notable algorithms in this class are the *Single Shot MultiBox Detector (SSD)* [56] and *YOLO* [65]. For the scope of this work, we disregard YOLO since even the authors point out that by its very design it has difficulties to detect small objects and groups of object instances which are located close to each other – two attributes that make YOLO unattractive for company logo detection.

Recent improvements to general object detection While the above-mentioned object detection pipelines have been very successful, they rely only on deep image features (i.e., features produced from layers at a late stage of the network) – with the exception of SSD which combines predictions from multiple feature maps. In classical network architectures like the ones described above, deep features have two properties: They describe high-level semantic aspects of an image but have a lower spatial resolution than the original input image. By contrast, shallow features (i.e., features produced from early layers of the network), usually only describe low-level image structures like edges or corners, but still maintain much of the spatial resolution of the original image. It has been observed that this reliance on low-resolution deep features is not ideal, especially for tasks which require precise location or fine-grained classification (e.g., segmentation). More recent approaches, therefore, attempt to incorporate multiple feature maps into the prediction: SSD [56] uses multiple network layers to make independent predictions for smaller and larger objects. Network architectures using Hypercolumns [33] make predictions by concatenating multiple features from different feature maps into a new feature representation. Other architectures that rely on concatenation of features are HyperNet [46] and Inside-Outside Networks [5]. A different approach to combine multiple feature maps are encoder-decoder networks: U-Net [70] follows the architecture of traditional classification networks by performing downsampling of feature maps between convolutions producing a low-resolution feature map. This low-resolution feature map is upsampled using deconvolutions while allowing earlier high-resolution information to refine the feature representation through skip-connections. Other approaches which employ skip-connections are Recombinator

Networks [40] and Stacked Hourglass Networks [60]. Feature Pyramid Networks [54] are conceptually similar to U-Net but in addition to U-Net use multiple feature maps to make predictions. Many of these approaches make use of multiple feature maps for prediction. All of them acknowledge the need for high-resolution information to improve predictions. However, there is another concept that none of them consider as an important factor: The *receptive field* of a feature map. In this work we will show the influence of the receptive field on the detection quality.

In addition to improvements in network architectures, some work has been done on the pipelines itself: [8] describe an improved non-maximum suppression which not only takes the score of the detection into account but also the degree of overlap between detections. However, this mechanism only offers potential improvements for datasets in which the bounding boxes of object instances tend to overlap strongly, which is not the case for company logo detection. There are other new approaches to object detection, which have been published recently that we do not cover in this work: Mask R-CNN [34] simultaneously predicts bounding boxes and segmentation masks. The inclusion of the segmentation task is believed to be beneficial to object localization. Cascade R-CNN [10] is a multi-stage extension to R-CNN, which implements a cascade classifier. Although designed for use with R-CNN it can in principle be applied to Fast(er) R-CNN as well. This approach can help to reduce false positives and improve localization. [44] uses a network to predict the overlap of detections with a groundtruth object which can be used to guide the non-maximum suppression. CornerNet [51] models object detection as a regression problem to predict the corner points of bounding boxes.

Company logo detection Finally, we like to examine some related work which is specific to company logo detection. There exist quite a few datasets for company logos: The METU dataset [75] is a very large dataset containing 923,343 instances from 687,842 different trademarks. However, this dataset is built primarily for trademark retrieval and thus is only of limited use for the task of company logo detection: It only provides image-level annotations and does not contain real-world images: Each image usually consists of a tight cut-out of the logo in front of a uniform white background. A large dataset geared towards company logo detection is the *Logos in the Wild* [76] dataset. This dataset consists of 32,850 annotated logo instances of 871 different brands. While both the quantity of logo instances and the quality of annotations is impressive, it suffers from two major drawbacks: It lacks a standardized evaluation script, and the images are not directly available for download. The dataset provides a list of URLs from which the images can be retrieved. Unfortunately, many of the URLs are no longer accessible which means that even with a commonly accepted evaluation protocol, the results may not be comparable. Aside from the FlickrLogos-47 dataset which we will introduce in chapter 2.1 there are a few other datasets which are usable for company logo detection. The most important examples are the BelgaLogos dataset [45, 52] – which consists of 2,697

annotated logo instances (which are not classified as *junk*) – and the LOGO-Net [38] dataset.

Previous work modeled company logo detection as an image retrieval problem [3, 45, 2] using SIFT [57] features. To improve the scalability of these approaches, feature triplets [68] have been introduced and have later been generalized to feature bundles [67].

Su et al. [73] investigate the viability of data augmentation to train a logo detection system in the absence of large amounts of training data. The approach is conceptually very similar to previous work [18] done by us. Hoi et al. [38] introduce LOGO-Net and apply several deep learning based approaches to object detection such as R-CNN [31] and Fast R-CNN [30] as a base line. In doing so, they notice similar problems for company logo detection as we do, which we will discuss in chapter 4.2.2. Bombonato et al. [9] apply the SSD framework [9] to the problem of company logo recognition.

1.4 Contributions

The contributions covered in this work can be summarized as follows:

Heuristic Region Proposals We analyze two algorithms for heuristic object proposals, namely Selective Search [77] and Edge Boxes [85], for their suitability for company logo detection. We notice that object proposals generated by Selective Search and Edge Boxes complement each other well and that by diversifying the heuristics by which object proposals are generated we are able to improve the performance while still using the same number of proposals. During our analysis of Selective Search, we notice a failure mode which frequently occurs in combination with text-based company logos. Since this failure mode is at the very core of Selective Search, we introduce VH-connect [82] – an additional heuristic which is targeted towards text-based company logos. We can show that we are able to improve the quality of proposals and that these improved proposals also translate into improved detections.

Improved Region Proposal Networks Aside from heuristic object proposals we also examine trainable object proposals. Specifically, we look at Region Proposal Networks (RPNs). When we apply RPNs to our dataset, we notice that they do not perform as well as heuristic object proposals. We analyze the reasons for this deficiency both theoretically and through experiment and identify two crucial factors which contribute to this behavior. We validate our results and show that by addressing these factors, we can considerably improve the performance of the proposal generation [17, 20].

Receptive field vs. object size We apply the R-CNN and Fast R-CNN detection pipeline to the problem of company logo detection. Although Fast R-CNN is significantly faster than R-CNN, we notice a steep difference in performance. This is surprising because Fast R-CNN can in principle be made equivalent to R-CNN. We analyze the differences between both approaches in detail and identify the relation between the size of the object and the receptive field of the network as an important factor for increasing classification performance. We show that simple magnification of the images can considerably improve the detection performance at the cost of considerably increased runtime. We propose a selective magnification strategy [19] which is able to exploit this fact while reducing the impact on the overall runtime.

Improved SSD Pipeline All the previously discussed techniques are only applicable in the context of two-staged object detection which consists of identification of object proposals and subsequent classification of these proposals. We turn to the SSD detector – a single stage approach to object detection which eliminates the need for object proposals. However, we find that we can apply our findings from both our analysis of Region Proposal Networks and our analysis of receptive fields to the SSD detector. In doing so, we are able to improve the quality of the detections considerably and can detect company logos orders of magnitude faster than R-CNN while simultaneously exceeding its detection performance.

Many parts of this work have previously been published in the academic literature. Our findings regarding improved object proposals with Region Proposal Networks have been published in [17] and [20]. The selective magnification strategy has been published in [19]. Our findings on heuristic object proposals were published in [82] which was done in cooperation with Anton Winschel. Another publication which is only marginally relevant to this work is our work on data augmentation for company logo detection [18] in which we show that we are able to train an R-CNN pipeline using only a few training examples.

1.5 List of Publications

Improving VLAD: Hierarchical Coding and a refined Local Coordinate System, Christian Eggert, Stefan Romberg, Rainer Lienhart, *IEEE International Conference on Image Processing 2014*, Paris, October 2014

On the Benefit of Synthetic Data for Company Logo Detection, Christian Eggert, Anton Winschel, Rainer Lienhart, *ACM Multimedia 2015*, Brisbane, October 2015

Saliency-guided Selective Magnification for Company Logo Detection, Christian Eggert, Anton Winschel, Dan Zecha, Rainer Lienhart, *IAPR*

International Conference on Pattern Recognition 2016, Cancun, December 2016

Improving Small Object Proposals for Company Logo Detection, Christian Eggert, Stefan Brehm, Dan Zecha, Rainer Lienhart, *ACM International Conference on Multimedia Retrieval 2017*, Bucharest, June 2017

A Closer Look: Small Object Detection in Faster R-CNN, Christian Eggert, Stefan Brehm, Anton Winschel, Dan Zecha, Rainer Lienhart, *IEEE International Conference on Multimedia and Expo*, Hong Kong, July 2017

1.6 Thesis outline

This work consists of four parts:

Part I introduces the foundations for this work. We will introduce the FlickrLogos-47 dataset which is a dataset for company logo detection that we have created with the help of GfK-Nürnberg e.V. We will discuss the properties of this dataset and its relation with other datasets for general object detection as well as some unique challenges associated with it.

Part II discusses two-stage object detection. Two-stage object detection consists of a *proposal stage* and a *classification stage*. We will first take a look at object proposals, which we group into heuristic object proposals and trainable object proposals. For heuristic proposals, we analyze the applicability of two algorithms to company logo detection: Selective Search [77] and Edge Boxes [85]. Here we also introduce the aforementioned VH-connect algorithm for improved proposals for text-based company logos. For trainable object proposals, we analyze Region Proposal Networks [66] and suggest improvements based on our observations.

After analyzing the proposal stage, this part also contains a look at the classification stage. We analyze R-CNN, Fast R-CNN, and Faster R-CNN and make our observations regarding receptive field and where we suggest our aforementioned selective magnification strategy.

Part III contains our evaluation of SSD and a description of how we can integrate our observations on the receptive field and on improving Region Proposals Networks into the SSD framework. We present our best pipeline for object detection and perform some evaluations on other datasets. Additionally, we compare our approach to other approaches to company logo detection.

Part IV concludes this thesis and contains an outlook on potentially rewarding directions for future research on company logo detection.

Part I

Foundations

Chapter 2

Datasets and Network Structures

2.1 The FlickrLogos-47 Dataset

2.1.1 Motivation

For evaluating our object proposal and detection pipelines, we have created a new dataset called *FlickrLogos-47*. FlickrLogos-47 is based on the FlickrLogos-32 dataset. FlickrLogos-32 is a dataset which was primarily conceived for the task of image retrieval, although it can also be used for object detection since it features object-level annotations in the form of segmentation masks and bounding boxes as well as image-level annotations.

As the name suggests, FlickrLogos-32 consists of 32 different object classes of company logos. The dataset contains 8240 images in total with 4280 images in the training/validation set and 3960 images in the test set. Because FlickrLogos-32 was conceived for image retrieval, both the trainval set and the test set contain 3000 logo-free images as distractor images.

However, it soon became clear that the object-level annotations of FlickrLogos-32 were insufficiently detailed for the task of company logo detection: The most prominent problem is that not all object instances were annotated but only the most prominent object instances. While this is not necessarily a problem from the standpoint of image retrieval, it is problematic for object detection because the set of negative examples is not clean. These incomplete annotations can have particularly strong effects when strategies as hard-negative mining are used.

Another problem of the FlickrLogos-32 dataset is that every image is associated with only a single class. While in many cases, multiple object instances per image have been annotated, all object instances have to be of the same class. Again, this makes sense in an image retrieval scenario but is too restrictive for the task of object detection because we could only use images that exclusively contain instances of the same logo class.

While it is certainly conceivable to use such a dataset for evaluating object detection pipelines we found that even in the original FlickrLogos-32 dataset multiple

Figure 2.1: Annotations of the FlickrLogos-32 dataset (top) and the FlickrLogos-47 dataset (bottom). Shown are only bounding boxes. The underlying annotations are pixel-level binary masks. The problems with the original annotation of the FlickrLogos-32 are apparent: Not all object instances were annotated and only one class per image was originally annotated. For many classes, FlickrLogos-47 also distinguishes between text- and symbol-based logos.

logo classes are sometimes found on the same image. In such a case usually the most prominent object instance is annotated while the other object instance is being ignored. Again, this poses a potential problem with regards to evaluation and hard-negative mining. Therefore, each object instance should be assigned its own class label for maximum flexibility.

Finally, company logos sometimes consist of multiple parts – often a symbol and a text-based logo. Not all parts are present for every given instance. Even if all parts are present, the geometrical arrangement between them may vary from instance to instance.

As a result of these shortcomings and observations, we have re-annotated and re-structured the FlickrLogos-32 dataset[1]. Aside from updating missing annotations, we have introduced separate class labels for symbol- and text-based logos wherever it made sense and the number of training examples allowed to do so.

Figure 2.1 shows some of the shortcomings of the original FlickrLogos-32 dataset and provides an impression of the differences in annotation between both datasets.

Additionally to providing every logo instance with its own class label, we have also introduced flags for truncated or otherwise difficult instances. This is useful for extremely small logo instances which only span a few pixels. Objectively, the logo is not recognizable from the pixel data itself, and its presence can only be inferred indirectly through the scene context.

Such annotations allow taking this into account during training: We can choose to exclude these object instances and avoid using them as either positive or negative examples in order not to confuse the classifier. During evaluation, we want to ignore

[1]We want to thank Dr. Karolin Kaiser and GfK-Nürnberg e.V. who provided most of the updated annotations for the FlickrLogos-47 dataset.

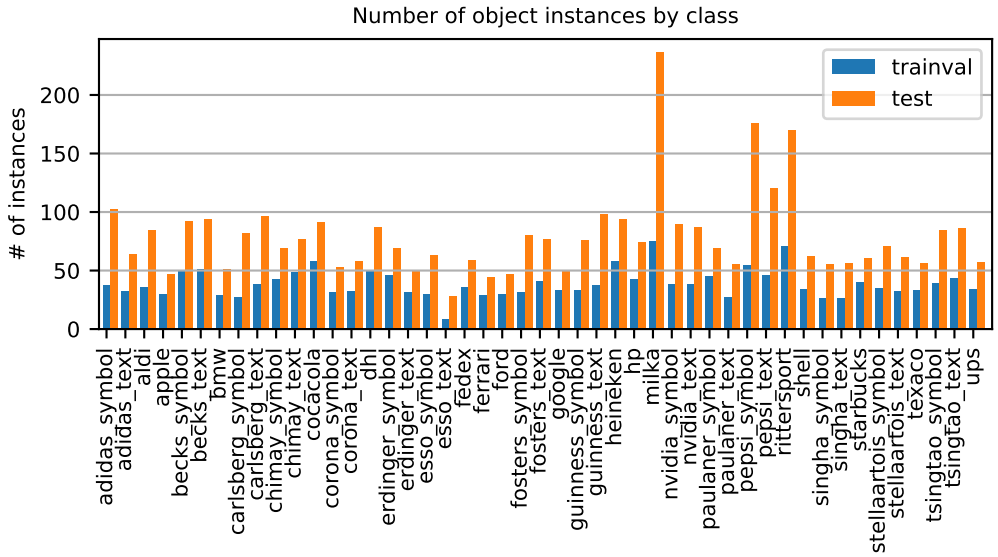


Figure 2.2: Number of (non-difficult) object instances per class. The number of training examples per class are largely balanced across all classes. A noteable exception is the class *esso_text* which only consists of 8 training examples.

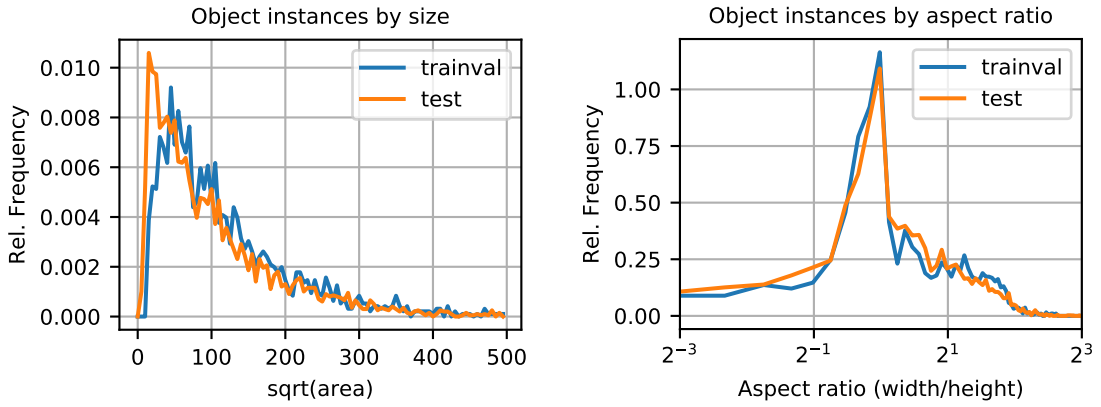
object instances that we cannot reasonably expect to be detected. At the same time, we do not want to penalize detectors that are able to detect such difficult instances. These annotations allow us to achieve exactly that.

2.1.2 Dataset Statistics

The re-annotation of logo instances made it necessary to re-structure the dataset as well. Since the FlickrLogos-32 dataset could only support image-level class information, it is easy to divide the images into training/validation and test so that every set contains enough examples.

With the re-annotation, each image of FlickrLogos-47 can contain multiple object instances of different classes. Since we still have to assign set membership at the image level, we cannot just retain the original image sets from the FlickrLogos-32 dataset because we cannot guarantee that a sufficient number of logo instances from each class will end up in each set.

FlickrLogos-47 is compiled from the same image corpus as FlickrLogos-32, but the assignment of images to trainval and test set needs to be changed to account for the new annotation: The new trainval image set is comprised of 833 images, and the test set contains 1402 images. Figure 2.2 shows the distribution of object instances between trainval and test set for all classes. For a detailed itemization of the dataset complete with information on the number of difficult and truncated examples, please refer to Table A.1 in Appendix A. It can be seen that for every



(a) (b)

Figure 2.3: (a) Size distribution of the FlickrLogos-47 dataset. Plot was generated from a histogram with a bin width of $5px$. A unique challenge of company logo detection are large variations in scale and the large fraction of small logo instances. (b) Distribution of aspect ratios (histogram bin width 0.1). Especially text-based company logos can have quite extreme aspect ratios.

class this new partition ensures that around 33% of all instances are found in the trainval set.

In addition to the *trainval* set, the dataset also contains a set of 3000 logo-free images, which we call the *nologo* set. This *nologo* set is intended to be used for hard-negative mining. Object detection datasets typically have no use for distractor images. For this reason, the distractor images of FlickrLogos-32 have been removed in the FlickrLogos-47 dataset.

A typical image in the dataset has a size of $1024px \times 768px$. The maximum size is $1024px \times 1024px$. A unique challenge of company logos detection is that company logos vary strongly in scale. This challenge can also be found in the FlickrLogos-47 dataset. The smallest non-difficult and non-truncated logo instance in the trainval set has a side length of $15px$ while the largest instance has a side length of $834px$. The median side length is $99px$. Figure 2.3 (a) shows the distribution of instance sizes for the trainval and test set. It is apparent that a significant fraction of company logos is comparatively small. Since small object instances are typically harder to detect than large object instances, this represents another challenge for company logo detection.

Figure 2.3 (b) shows the distribution of aspect ratios for the trainval and test set. Unsurprisingly, most of the object instances have quadratic bounding boxes. However, a significant number of instances have bounding boxes with rather extreme aspect ratios. These instances tend to be associated with text-based company logos. Since many algorithms for object detection make a priori assumptions about the shape of the objects, extreme aspect ratios can also cause problems.

2.1.3 Evaluation Protocol

Evaluation Metrics for Detection

The FlickrLogos-47 dataset comes with a new evaluation script which is targeted at evaluating object detection tasks. In the following, we will describe the metric used for evaluating object detection and how it is computed. We will first look at a single class detection problem – which means one class versus background – and then see that we can easily generalize to multi-class detection problems.

An object detector returns a list of detections. Each detection consists of a location in the form of a bounding box $\mathcal{B}_{det}$, a class label c and a confidence score s. For a single class detection problem, each detection can be classified into one of two categories: *True positives* (TPs) and *false positives* (FPs).

A TP is a detection which predicts the presence of an object at the correct location with sufficiently high confidence. A FP can occur in two situations: (1) A detection with sufficiently high confidence that is predicted at the wrong location. (2) A detection with sufficiently high confidence that is predicted at the correct location, but there exists another detection with higher confidence which already detects the corresponding groundtruth object.

Another important quantity is *false negatives* (FNs) which are groundtruth instances which have not been detected with sufficiently high confidence. Since the absolute numbers are often not easy to interpret, TPs, FPs, and FNs are commonly expressed in terms of *precision* and *recall*. *Precision* is the fraction of all detections which are indeed correct. *Recall* is the fraction of all object instances that have been sucessfully detected.

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

It remains to be resolved what qualifies as *correct location* and *sufficiently high confidence*. We measure the degree of overlap between two bounding boxes $\mathcal{B}_1$ and $\mathcal{B}_2$ using the *intersection over union* (IoU) criterion

$$IoU(\mathcal{B}_1, \mathcal{B}_2) = \frac{|\mathcal{B}_1 \cap \mathcal{B}_2|}{|\mathcal{B}_1 \cup \mathcal{B}_2|} \tag{2.3}$$

where $|\mathcal{B}_1 \cap \mathcal{B}_2|$ represents the size of the area enclosed by the intersection of the bounding boxes $\mathcal{B}_1$ and $\mathcal{B}_2$ and $|\mathcal{B}_1 \cup \mathcal{B}_2|$ the size of the area enclosed by the union of both bounding boxes. A detection is defined to have *correct location* if it has an $IoU \geq 0.5$ with a groundtruth instance.

Sufficiently high confidence is hard to define since the desired level of confidence in a detection will vary depending on the application. For some applications, it

might be acceptable to have a large number of false positives as long as all relevant objects are detected while for other applications it might be important that every returned detection is indeed correct. Since it is hard to specify a single confidence threshold which covers all application scenarios we evaluate precision and recall over all possible confidence thresholds.

We can plot the precision and recall values for all confidence thresholds. This is known as a precision-recall (PR) curve. To compute a single number which measures the overall quality of the detector we use the measure of *Average Precision* (AP) which is the area under the PR-curve. We compute AP from the precision and recall values using the trapezoidal rule.

So far, we have only considered single-class detection problems. In a multi-class problem, TPs and FPs need to be defined slightly differently: A TP is defined as detection of the correct class with sufficiently high confidence at the correct location. FPs come in three types instead of two: (1) A detection with sufficiently high confidence at a wrong location. (2) A detection with sufficiently high confidence of the correct class and at the correct location, but there exists another detection with higher confidence which already detects the corresponding groundtruth object. (3) A detection with sufficiently high confidence at the correct location but with the wrong class.

The evaluation framework can be extended to multi-class problems in a fairly straightforward way: We simply treat a multi-class detection problem as a series of single class detection problems. This means we filter the set of groundtruth instances and the set of detections for a particular class label. We can apply the single class evaluation framework to this set and obtain an (AP) for each class. The final score of the detector over all classes is measured by *mean Average Precision* (mAP) which is simply the mean over all APs.

Note, that this evaluation protocol automatically takes care of type (3) FPs. By filtering the detections by class, an FP at the correct location but with incorrect class will not be counted as a TP for the correct class. It will, however, be counted as an FP when the incorrectly predicted class is being evaluated.

Efficient computation

Since *precision* and *recall* must be evaluated over a large number of thresholds, we briefly discuss how the PR curve is efficiently computed in the FlickrLogos-47 evaluation script. We start by assigning detections to groundtruth items. For each detection, we determine the groundtruth instance with the largest intersection over union (IoU). If the largest IoU is greater or equal 0.5, we assign the detection to this groundtruth instance. Although this requires an exhaustive search over all groundtruth instances in the same image, this does not represent a problem since both the number of detections and the number of groundtruth instances per images are small.

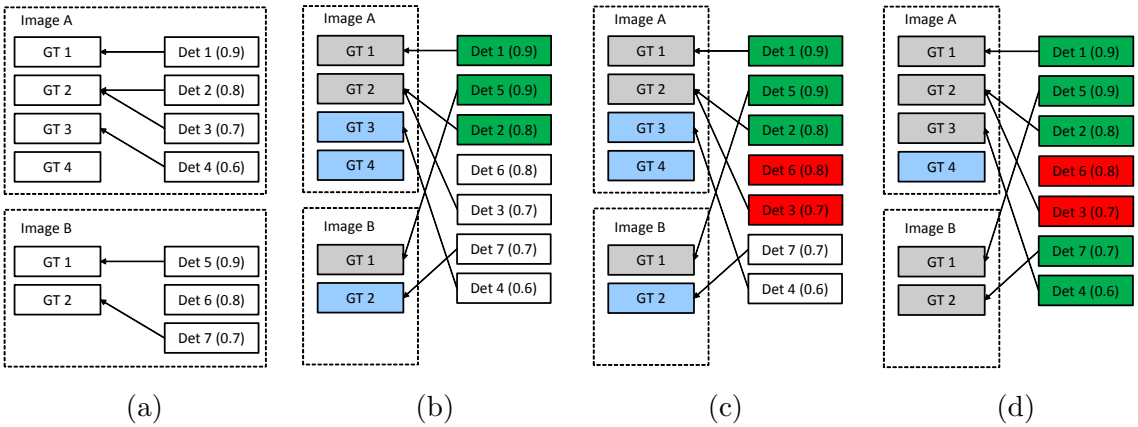


Figure 2.4: Computing the PR curve. (a) Build a graph assigning detections to GT instances based on IoU. (b) Situation after processing the three strongest detections. TPs are marked in green, detected instances in grey and undetected instances (FNs) in blue. (c) State after processing the five strongest detections. FPs are marked in red. (d) All detections have been evaluated.

The result of this assignment can be interpreted as a bipartite graph. Detections and groundtruth instances are represented by nodes and assignments are represented by edges. In such a graph, a groundtruth (GT) node with at least one incoming edge represents an object instance that has been successfully detected while a GT node with no incoming edges represents a *false negative*. If a GT node has more than one incoming edge, only one of the associated detections (the one with the highest confidence score) can be counted as *true positive*, all others must be counted as *false positives* (FPs). Detection nodes with no outgoing edges also represent FPs. Nodes of GT instances that carry the *difficult* flag are removed from the graph along with all detection nodes that have been assigned to them.

After the graph has been built, we sort the list of detections by descending confidence score. By traversing the sorted list, we can incrementally compute *precision* and *recall* for a situation in which we set the confidence threshold to include the n most robust detections. In order to correctly count FPs, we need to keep track which GT instances have already been assigned to another detection. This can be done efficiently through data structures which are commonly used to implement set operations such as hash tables or binary trees.

This process is illustrated in Figure 2.4. It allows us to compute the complete PR curve by traversing the list of detections exactly once. In each step, we update the number of TPs, FPs, and FNs accordingly and therefore compute a point in the PR curve.

2.2 The VGG16 Network

2.2.1 Architecture and Nomenclature

After the success of AlexNet [49] on the ImageNet [15] classification challenge, much research went into alternative network architectures to improve results even further. This has lead to a large zoo of different network architectures with OverFeat [71], Inception [74] and deep residual networks (ResNets) [36] being just a few prominent examples.

In this work, we will refer almost exclusively to the VGG16 [72] architecture. Like the examples mentioned above, the VGG16 and VGG19 architectures [72] were born in the wake of AlexNet. With 16 and 19 trainable layers, these networks were considered to be very deep architectures at the time which pushed the limit of what was considered to be trainable. New ideas allow more modern architectures like ResNets to be much deeper than VGG19. The reason we are choosing VGG16 over more modern network architectures is that we found it to produce the best results on our FlickrLogos-47 dataset (an observation we will discuss in more detail in chapters 4.1 and 4.2).

The VGG16 architecture is composed of only three different building blocks: Convolutional layers, max-pooling layers, and fully-connected layers. In the following, we will refer to the output of fully-connected layers as *features* since they contain a representation of the image content that is in some way useful for the task the network is trying to solve. By contrast, we will refer to the output of a convolutional layer as a *feature map* because unlike fully-connected layers they explicitly retain spatial information.

A feature map $\mathcal{F}_{b,c,y,x}$ can be seen as a 4-dimensional array with the spatial indices x and y and the channel index c. Training usually happens on small subsets of the dataset (so-called *mini-batches*). Therefore, we also index the items of a mini-batch in b. Because most operations in a network are applied independently for each item of a mini-batch, we will sometimes omit the batch index for clarity.

Often we will also use the term *pixel* with regard to a feature map. In this context, a *pixel* refers to a specific spatial location in the feature map. Analogous to an image where every pixel is associated with values for three color channels, the term *pixel values* refers to the vector of values in the channel dimension when used in the context of feature maps.

VGG16 consistently uses convolutional layers with 3×3 kernels. Inputs to convolutional layers are zero-padded at the borders by $1px$ so that the convolutions do not modify the size of the input. Downsampling is performed exclusively by max-pooling layers with 2×2 kernels with a stride of 2. The output size after a max-pooling operation is therefore $\left\lceil \frac{n}{2} \right\rceil$ for an input side length of n. The final layers of the network are composed of fully-connected layers which act as the classifier for the features extracted by the convolutional layers.

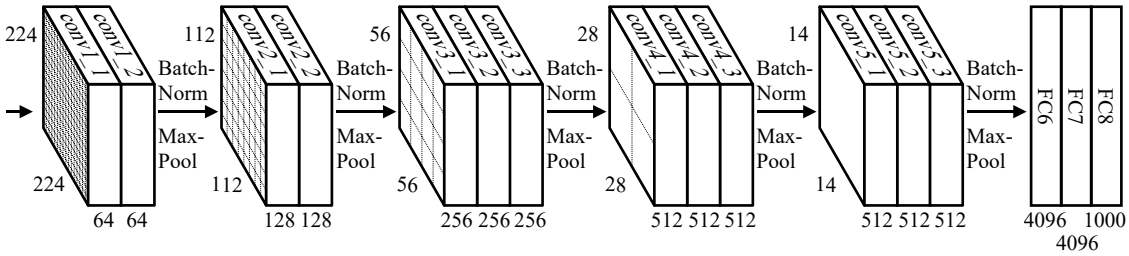


Figure 2.5: Architecture of our modified VGG16 network for ImageNet classification. The difference to the original architecture is the inclusion of batch normalization layers after each convolution group which simplifies the future addition of additional network branches. Batch normalization layers follow the post-activation placement and are located directly before the ReLU of the last convolutional layer in each group.

The network structure of VGG16 is also responsible for one of its most appealing characteristics: Because of the consistent padding and downsampling, there exists a simple mapping between image coordinates $\boldsymbol{x}_\mathcal{I}$ and feature map coordinates $\boldsymbol{x}_\mathcal{F}$ which is given by $\boldsymbol{x}_\mathcal{F} = \alpha_\mathcal{F}^{-1}\boldsymbol{x}_\mathcal{I}$. Here, $\alpha_\mathcal{F}$ represents the downsampling factor of the feature map $\mathcal{F}$, which expresses how much smaller the height/width of $\mathcal{F}$ is compared to the input image. The downsampling factor is determined by the number of max-pooling operations $n_\mathcal{F}$ that preceded $\mathcal{F}$. In the VGG16 architecture, the downsampling factor is given by $\alpha_\mathcal{F} = 2^{n_\mathcal{F}}$. For other network structures like AlexNet, the lack of padding makes this relationship much more complicated.

Figure 2.5 shows the network structure of VGG16. The convolutional layers are organized in groups of two or three before downsampling occurs. In this work, we will often not refer to the convolutional layers individually, but to the group into which they belong: When we mention to the *conv2* layer, we actually refer to the group of layers composed of the *conv2_1* and *conv2_2* layer. Sometimes – depending on the context – the mention of *conv2* will also refer to the output of the last layer in the *conv2* group. With the exception of the last convolution group, the number of channels doubles from each convolution group to the next.

As can be seen from Figure 2.5, our network architecture differs slightly from the original VGG16 network architecture [72]: We introduce batch normalization [43] layers between each convolution group. More specifically, we add batch normalization layers directly after the last convolution in each convolution group, but before the ReLU and max-Pooling layers. This design corresponds to the original idea from Ioffe et. al. [43] which is known as *post-activation* placement. Based on the experiments of He et. al. [37], newer ResNet architectures often place the batch normalization layer in front of the convolutional layer. This design in which the batch normalization is followed by a ReLU which is followed by the convolutional layer is known as *pre-activation* placement. Pre-activation placement has been re-

ported [37] to produce marginally better results for Image classification with deep residual networks.

During training, batch normalization normalizes the activations of a feature map $\mathcal{F}^{in}$ in the following way

$$\mathcal{F}^{bn}_{b,c,y,x} = \frac{\mathcal{F}^{in}_{b,c,y,x} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} \tag{2.4}$$

where ϵ represents a small positive constant designed to prevent division by zero and

$$\begin{aligned} \mu_c &= \gamma \sum_{b,y,x} \mathcal{F}^{in}_{b,c,y,x} \\ \sigma_c^2 &= \gamma \sum_{b,y,x} (\mathcal{F}_{b,c,y,x} - \mu_c)^2 \end{aligned} \tag{2.5}$$

with γ being a normalization factor based on the number of summed items. Informally speaking, batch normalization performs a channel-wise standardization with respect to the current mini-batch. In combination with variance scaling initialization [35] (also known as MSRA initialization), this allows us to directly pre-train the network on ImageNet from scratch. Batch normalization also helps us in training multi-arm networks (as we will discuss in chapter 3.5.4).

Batch normalization can be seen as a way to regularize activations. As a regularization mechanism, it constrains the expressive power of the network. To counter this problem, batch normalization is usually followed by an additional *scale and shift* layer which allows the network to learn a scale factor and an additive offset. In principle, this arrangement allows the network to undo the effects of the batch normalization and restore the expressive power of the original network without batch normalization. Since our primary purpose for employing batch normalization is to normalize feature maps for multi-arm training, we omit this *scale and shift* layer in our network architecture.

The model defined by the network is very large and consists of a total of 138×10^6 trainable parameters (excluding parameters introduced by batch normalization which adds 8×10^3 parameters to the network). It is important to note that most parameters are concentrated in the fully-connected layers of the network which account for 123×10^6 of all parameters. Almost all parameters of the fully-connected layers can be found in the *fc6* layer which contains 103×10^6 parameters.

2.2.2 Receptive Field

In the context of introducing the VGG16 architecture, we also want to introduce the concept of the *receptive field* because it will be used frequently in this work. The *receptive field* of a feature map $\mathcal{F}$ is defined as the set of pixels in the input image that can potentially influence a single output neuron in $\mathcal{F}$.

For example, the size of the receptive field of a hypothetical network consisting of a single 3×3 convolution will be 3×3. If we look the receptive field of a network consisting of two subsequent 3×3 convolutions, the receptive field is 5×5. Since in all cases in this work the receptive field is quadratic we will refer to a $s \times s$ receptive field as a receptive field of size s.

To calculate the receptive field of a network, it makes sense to look at the individual operations or layers in terms of *kernel size* and *stride*. If we enumerate the layers in a network $L_1 \dots L_n$ from the last layer L_1 to the to first L_n, we can compute the receptive field $r = r_n$ as follows:

$$\begin{aligned} r_0 &= 1 \\ r_i &= r_{i-1} s(L_i) + k(L_i) - s(L_i) \end{aligned} \tag{2.6}$$

where $s(L_i)$ represents the stride and $k(L_i)$ the kernel size of the layer L_i. While it is possible to extend the definition of a receptive field to fully-connected layers (see Chapter 2.2.3) it is common to limit the application of this concept to the part of the network responsible for feature extraction. Used in such a way, the receptive field of the VGG16 network is synonymous with the receptive field of the *pool5* feature map which has a receptive field of $212px$.

2.2.3 Usage as fully-convolutional Network

In its original form for ImageNet classification, VGG16 is a network which takes an image with fixed dimensions as input and outputs a vector of fixed length containing the predictions. While this arrangement can be used for some of the approaches discussed in this work (e.g., in R-CNN discussed in chapter 4.1), some object detection approaches like SSD (Chapter 5) require the ability to process images of arbitrary size.

The reason why the VGG16 network has to operate on a fixed-size input are the fully-connected layers. Convolution operations are agnostic to size: If the input size increases, the output size increases as well. Fully-connected layers, on the other hand, pose a problem: Their input and output size are directly tied to the number of weights. Once trained, this number cannot change. Therefore, the fully-connected layers limit the input and output size of the network. A fully-convolutional architecture is required to handle inputs of arbitrary size.

There are two ways in which VGG16 can be converted to a fully-convolutional network: The easiest way is simply to remove the fully-connected layers. If a fixed-sized output is still required (as is the case during ImageNet pretraining), global average-pooling can be used to reduce a feature map of arbitrary size to a feature map of size 1×1.

If removing the fully-connected layers is undesirable, they can be converted to convolutional layers in the following way: Let us assume $H \times W$ to be the spatial dimensions of an input to a fully-connected layer that has N output neurons. The

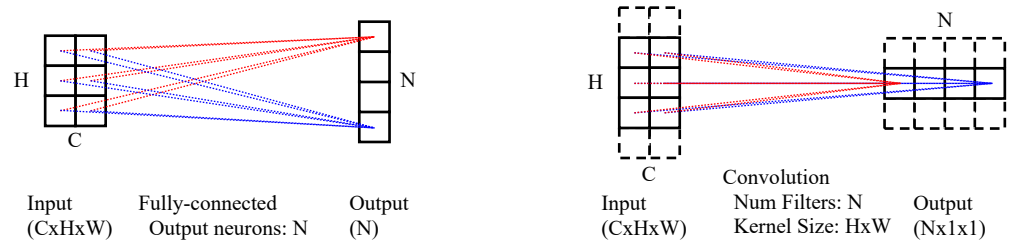


Figure 2.6: Conversion of a fully-connected layer to a convolutional layer. In both cases, the input is a feature map of dimensions $C \times H \times W$ (only one spatial dimension shown). (a) Each line represents a weight in a fully-connected layer producing N output neurons. (b) The weights are re-interpreted as coefficients of N different convolution kernels. This allows pre-trained networks to process inputs of arbitrary size.

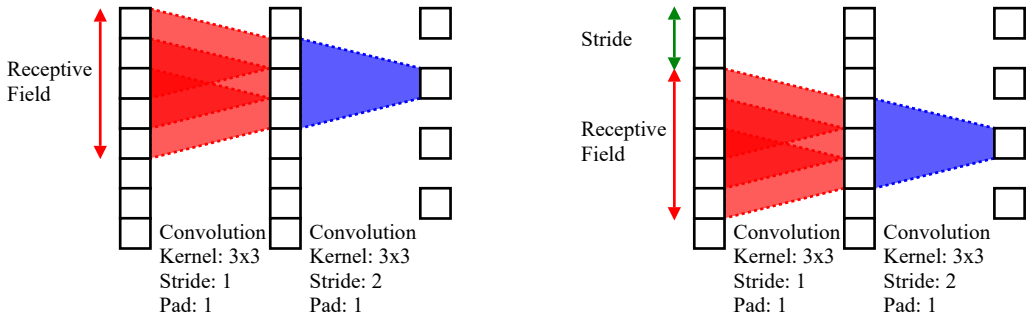


Figure 2.7: Convolutional neural network as filter. The filter size is equal to the receptive field of the network while the stride between filter applications is determined by the downsampling factor of the network. This examples shows a fully-convolutional network with a downsampling factor of $\alpha = 2$. Two neighboring output neurons are computed as two filter applications.

output can be interpreted as the result of N convolutions (without padding) with a kernel size of $H \times W$ as figure 2.6 shows.

By this method, any network that is terminated in fully-connected layers can be converted into a fully-convolutional network. A fully-convolutional network can be viewed as a (highly non-linear) filter that is applied to the image in a strided fashion: The kernel size of the filter is given by the size of the network's receptive field. The stride of the filter application is given by the downsampling factor of the network. This is shown in figure 2.7.

This conversion of VGG16 to a fully-convolutional network is a powerful mechanism which we will use multiple times in this work.

Part II

Two-Stage Object Detection

Overview

The first large-scale application of deep convolutional neural networks [49] to the problem of image classification on the ImageNet [15] dataset did prove to be very successful. Inspired by this success, many efforts were undertaken to make this – at that point previously unseen – classification performance useful for object detection.

In addition to classification, object detection also aims to determine the location of an object in an image. However, both tasks are closely related. Given the existence of a classifier which can classify images, it is – in principle – also possible to solve the problem of object detection by applying this classifier to every location and aspect ratio in an image. However, it is computationally expensive to evaluate a classifier at every position and aspect ratio in the image.

A possible solution is to limit the evaluation of the classifier to pre-selected image regions which are likely to contain objects. We categorize approaches which follow this idea as *two-staged object detection* approaches. These approaches are characterized by two separate stages called the *object proposal stage* and the *classification stage*. The task of the *object proposal stage* is to identify image regions that are likely to contain objects quickly. We often refer to these selected image regions as *object proposals*. The task of the *classification stage* is to classify these *object proposals*.

Two-staged object detection approaches inherently limit the maximum possible recall and thereby also the maximum achievable mAP: Object instances that do not have a strong intersection over union with at least a single object proposal are lost and cannot be detected by any classifier – no matter how well the classifier performs.

On the other hand, two-staged object detection can also offer a potential benefit. They can improve the overall detection performance if a classifier produces a lot of false positives by limiting the number of image regions where the classifier needs to be evaluated.

In this chapter, we will first take a look at the criteria for good objects proposals and discuss evaluation metrics. We will then discuss several algorithms for generating object proposals together with their strengths and weaknesses as well as possible strategies to improve their performance.

Chapter 3

Proposal Stage

3.1 Criteria for good object proposals

The purpose of object proposals is to select image regions which are likely to contain objects which are to be classified later. As a result, it should be true that every object instance is covered by at least a single proposal out of the set of object proposals. A failure to do so limits the maximum achievable recall of the detection pipeline.

On the other hand, this selection needs not be perfect: Not every object proposal is required to contain an object. However, a good object proposal algorithm is characterized by its ability to cover all object instances in an image with a small number of object proposals since it limits the number of regions that need to be considered by the classifier.

Another useful property of an object proposal algorithms is tight localization: It is likely that the classifier will perform better if the proposal stage can provide it with well-localized bounding boxes. This means providing a bounding box with a large intersection over union with the object instance.

For these reasons, we use two different metrics to measure the quality of object proposals: Recall and mean average best overlap (MABO). In order to define recall, we need to define what we regard as a true positive: We define a true positive to be an object proposal whose maximum intersection over union (IoU) with any groundtruth instance is greater or equal to a certain threshold t.

$$recall(t) = \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} \left[\max_{p \in \mathcal{P}} IoU(g, p) \geq t \right] \tag{3.1}$$

where G is the set of all groundtruth instances, $\mathcal{P}$ is the set of all object proposals, [.] denotes the Iverson bracket, and $IoU(g, p)$ represents the intersection over union between a groundtruth instance g and the proposal p. Both groundtruth instances and object proposals are described by their bounding boxes.

For a given IoU threshold t and a given number of object proposals, this metric is able to capture the algorithms ability to find objects. However, it does not capture the ability of the algorithm to localize objects precisely.

For this reason, we plot IoU threshold against recall and measure the area under curve which we call *average recall* (AR).

$$AR = \frac{1}{1 - t_l} \int_{t_l}^{1} recall(t)dt \tag{3.2}$$

where t_l represents the minimum acceptable IoU a proposal is allowed to have with an object to count as a true positive example. This value is usually set to $t_l = 0.5$, and we also follow this commonly accepted threshold. Since the maximum achievable value for this integral depends on the integration boundaries, we introduce the normalization constant $\frac{1}{1-t_l}$. This results in a metric where every perfect (recall and localization) object proposal generator has an $AR = 1.0$, regardless of the choice of t_l.

In the literature [77, 83, 50] we often also find another metric called *mean average best overlap* (MABO) which was introduced by [77] and which we also provide for easier comparison. The computation of MABO starts with the *average best overlap* (ABO):

$$ABO_c = \frac{1}{|\mathcal{G}_c|} \sum_{g \in \mathcal{G}_c} max_{p \in \mathcal{P}}(IoU(g, p)) \tag{3.3}$$

where $\mathcal{G}_c$ represents the set of all object instances for a particular class c.

While the ABO metric is only defined for a single object class c, the MABO metric is defined on the complete dataset as the mean over all ABO measurements for each class.

$$MABO = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} ABO_c \tag{3.4}$$

where $\mathcal{C}$ represents the set of all object classes.

3.2 Selective Search

Selective Search [77] is a heuristic approach for generating object proposals. It starts with an initial over-segmentation of the image. Each of these segments can be represented by a bounding box and represents an object proposal. Selective Search iteratively merges the most similar adjacent regions in the image. Similarity is defined as a sum of multiple similarity metrics which we discuss in the following chapter. This process continues until only a single region remains, which covers the complete image. After each merger of two regions, a new object proposal is

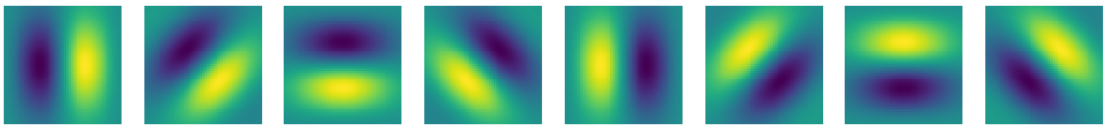


Figure 3.1: Gabor filterbank used by Selective Search for measuring texture similarity between image regions.

generated which is the bounding box of the newly merged regions. This process results in a hierarchical grouping of image regions.

3.2.1 Similarity Metrics

Since even objects of the same class can vary strongly in appearance, the similarity metrics used by Selective Search to determine similar image regions need to take this variability into account. For a heuristic approach like Selective Search, which only relies on hand-crafted features, a single similarity metric is unlikely to produce the desired result. In order to diversify the similarity search, Selective Search uses four different similarity metrics to cover the visual variability of object appearances:

Color similarity For each pair of image regions (r_k, r_l), Selective Search uses color histograms to compute the color similarity. For each region r_k, three histograms $H^{i,k}_{col} = (h^{i,k}_{col,0}, \ldots, h^{i,k}_{col,B-1})$ with $B = 25$ bins each are used, one for each color channel i. The color histograms $H^{i,k}_{col}$ are normalized by concatenating the histograms for all color channels into a single histogram which is L_1 normalized. We will refer to the normalized histograms as $\tilde{H}^{i,k}_{col} = (\tilde{h}^{i,k}_{col,0}, \ldots, \tilde{h}^{i,k}_{col,B-1})$. This concatenated and normalized histogram forms the color descriptor for this region. Color similarity $s_{col}(r_k, r_l)$ between two image regions (r_k, r_l) is computed using the histogram intersection:

$$s_{col}(r_k, r_l) = \sum_{i \in C} \sum_{b=0}^{B-1} min(\tilde{h}^{i,k}_{col,b}, \tilde{h}^{i,l}_{col,b}) \tag{3.5}$$

Texture similarity In order to measure texture similarity, Selective Search uses a similar approach as for color similarity. Each color channel of the image is separately convolved with a set of Gabor filters [29]. We limit ourselves to gaussian directional derivatives in $D = 8$ directions as shown in Figure 3.1. For each color channel i and directional derivative d we construct a histogram $H^{i,d,k}_{tex} = (h^{i,d,k}_{tex,0}, \ldots, h^{i,d,k}_{tex,B-1})$ with $B = 10$ bins. The texture histograms are concatenated into a single histogram which is L_1 normalized. This ensures equal contribution of color and texture similarity. We denote the normalized histograms as $\tilde{H}^{i,d,k}_{tex} = (\tilde{h}^{i,d,k}_{tex,0}, \ldots, \tilde{h}^{i,d,k}_{tex,B-1})$

Again, we use the histogram intersection to measure texture similarity $s_{tex}(r_k, r_l)$ between two image regions (r_k, r_l):

$$s_{tex}(r_k, r_l) = \sum_{i \in C} \sum_{d=0}^{D-1} \sum_{b=0}^{B-1} min(\tilde{h}_{tex,b}^{i,d,k}, \tilde{h}_{tex,b}^{i,d,l}) \tag{3.6}$$

Region size One way to visualize the grouping strategy of Selective Search is to view it as a tree. Each node in this tree represents an image region which has two child nodes: The two image regions which have been merged in a previous iteration of Selective Search. The leaf nodes represent the regions that have been identified through the initial segmentation of the image. In order to prevent degeneration of this binary tree, Selective Search encourages that small regions be merged first. This is done by introducing region size s_{size} into the similarity metric:

$$s_{size}(r_k, r_l) = 1 - \frac{size(r_k) + size(r_j)}{size(im)} \tag{3.7}$$

where $size(r_k)$ refers to the size of the image region r_k in pixels and im represents a region which encompasses the whole image.

Region fill Since the only criteria considered so far are similarity and region size, it is possible that the merging process will create gaps. If for example, three regions encompass each other it is possible that the outer regions are being merged first, leaving a gap in the center. In order to introduce a preference for filling space Selective Search introduces a term s_{fill} into the similarity metric:

$$s_{fill}(r_k, r_l) = 1 - \frac{size(bb(r_k \cup r_l)) - size(r_k) - size(r_l)}{size(im)} \tag{3.8}$$

where $bb(r_k)$ represents the bounding box around the image region r_k. For consistency with s_{size} the term is normalized by $size(im)$.

3.2.2 Diversifying object proposals

The previously discussed similarity metrics look at different aspects of image regions. To capture a diverse range of objects, these similarity metrics are used in different combinations during multiple runs of Selective Search. The final similarity s_{total} between a pair of image regions (r_k, r_l) is given as:

$$s_{total}(r_k, r_l) = \begin{pmatrix} \alpha_{col} \\ \alpha_{tex} \\ \alpha_{size} \\ \alpha_{fill} \end{pmatrix} \cdot \begin{pmatrix} s_{col}(r_j, r_k) \\ s_{tex}(r_j, r_k) \\ s_{size}(r_j, r_k) \\ s_{fill}(r_j, r_k) \end{pmatrix} \tag{3.9}$$

Figure 3.2: Illustration of Selective Search. From left to right: Input image. Initial oversegmentation. Iterative hierarchical grouping of image regions. With each merger of adjacent regions, a new proposal is generated whose bounding box covers the merged region.

$\alpha_{col}, \alpha_{tex}, \alpha_{size}, \alpha_{fill} \in \{0, 1\}$ are binary weights that determine which of the individual similarity metrics are active in the current run of Selective Search.

We implement the 'fast' Selective Search mode proposed by [77] which executes Selective Search eight times on each image: The algorithm is executed on both the RGB and HSV representation of the image, using two different initial segmentations for each and using two combinations of similarity metrics for each segmentation. Following [77], we use: $(\alpha_{col}, \alpha_{tex}, \alpha_{size}, \alpha_{fill}) = (1, 1, 1, 1)$ and $(\alpha_{col}, \alpha_{tex}, \alpha_{size}, \alpha_{fill}) = (0, 1, 1, 1)$ as configurations for the similarity metrics.

3.2.3 Ranking object proposals

Since Selective Search can generate a large number of object proposals it is desirable to rank these proposals by relevance. Selective Search uses a simple heuristic to rank proposals: Image regions that are merged at a later stage in the algorithm are more likely to contain objects.

If after the initial segmentation there are K image regions, Selective Search will need $K-1$ iterations to merge all of them. For each merger, Selective Search records the iteration i at which the proposal p_i was generated. The relevance score v_i for this proposal is given by $v_i = RND \cdot (K - i)$, where RND is a random number from the interval $[0, 1]$. Aside from the random element, this will result in a score of $v_{K-1} = 1$ for the final merger (generating a proposal that covers the complete image) and a score $v_i > 1$ for all the other proposals. Proposals are sorted by ascending relevance score to obtain the final ranking.

A problem with this strategy is that Selective Search is usually run multiple times to diversify object proposals. However, since Selective Search is biased towards large objects, every run tends to displace small objects from the top-scoring positions of the result list. The random element introduced by RND is designed to counteract this problem.

3.3 Edge Boxes

Another popular heuristic for generating object proposals is Edge Boxes [85]. While Selective Search [77] uses the principle of grouping similar image regions, Edge Boxes operates on dissimilarities between image regions – so-called edges. The intuition behind this approach is that object instances are likely to be found at image regions that are enclosed by edges – so-called contours.

Edge Boxes is able to compute an *objectness score* which is based on the number of edges that are enclosed by the bounding box of the proposal minus the number of edges which overlap the box's boundary. This *objectness score* allows Edge Boxes to rank proposals directly based on image content while Selective Search relies on the order of merger as an indirect ranking mechanism.

The algorithm behind Edge Boxes itself is a heuristic approach. However, the underlying edge detection algorithm [16] used by [85] is not, since it makes use of a random forest. Since any algorithm which is able to generate probabilistic edges (i.e., edges associated with a confidence value) can be used, we still feel justified in calling Edge Boxes a heuristic approach.

3.3.1 Finding and grouping edges

In principle, it is possible to use any algorithm that is able to find probabilistic edges in images in conjunction with Edge Boxes. These edge detectors might themselves be based on heuristics. However, the original Edge Boxes implementation by [85] uses a trainable edge detector – called a *structured forest* [16] edge detector – which is based on a random forest classifier which takes an image patch as input. The classifier predicts whether the center of the image patch is located on an edge or not.

The structured forest detector returns an edge map which for every pixel contains a confidence value that this pixel is located on an edge and an orientation estimate of that edge. This edge map is binarized using a threshold, and the resulting hard edges are grouped into clusters to speed up the computation of the scoring function. Edge Boxes [85] uses a simple greedy grouping algorithm which is outlined in Appendix B: For each edge that has not been previously assigned to a group, the algorithm looks at the neighboring 8-group of edges, which is added to a set of *discovered* edges. The algorithm then picks the edge with the smallest difference in orientation from the *discovered* set and adds this edge to the group. The algorithm continues with the 8-neighborhood around the edge that was just selected until a threshold on total angular difference within a group is reached.

The output of the edge grouping algorithm are groups of edges G_i. Each edge group G_i is represented by its mean pixel position p_i and its mean orientation θ_i. Based on this information an affinity matrix $A_{i,j}$ is built between edge groups G_i and G_j as follows:

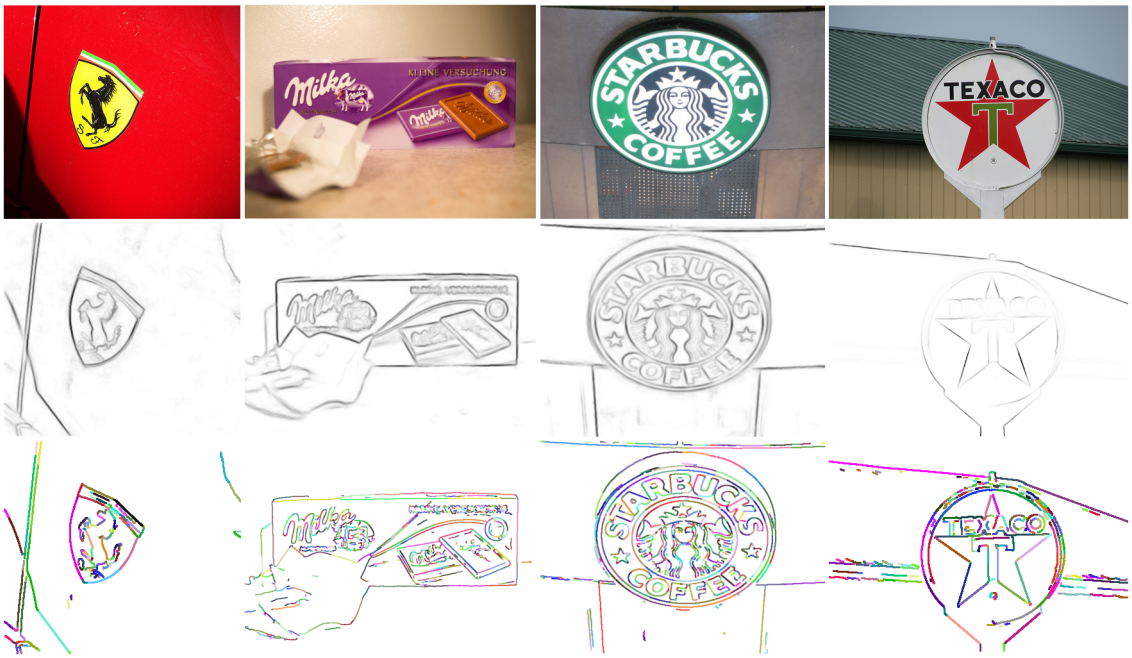


Figure 3.3: Visualization of edge groups. **First row:** Input images. **Second row:** Output of the structured forest edge detector. **Third row:** Visualization of edge groups. Each color represents an edge group. Proposal scores are based on the number of enclosed edge groups by the candidate box.

$$A_{i,j} = \begin{cases} |cos(\theta_i - \theta_{i,j})cos(\theta_j - \theta_{i.j})|^\gamma & \text{if } d(i,j) < t_d \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

$\theta_{i,j}$ represents the enclosed angle between mean edge group positions p_i and p_j. $d(i,j)$ stands for the euclidean distance between edge group positions (as represented by p_i) and t_d represents a distance threshold. Typically, t_d is set to $t_d = 2$ which results in a very sparse affinity matrix. γ controls the sensitivity of the affinity to differences in orientation and is set to $\gamma = 2$.

3.3.2 Scoring function

The grouping of edges allows for a fast computation of the scoring function which serves to rank proposals. For a given proposal (as represented by a bounding box b), the intuition behind the scoring function is for a given object proposal to count the number of enclosed edges.

For this purpose, a value $w_b(G_i) \in [0,1]$ is computed for every edge group $G_i \in G$ which represents the degree to which G_i is contained by the box b. A value of $w_b(G_i) = 0$ represents an edge group G_i that is not enclosed by b. Ideally, every edge group G_i that is fully enclosed by b should get a value of $w_b(G_i) = 1$. However,

since edge detection is not perfect, a conceptual edge that exists in the original image may end up being split into two edge groups that are located closely to each other. Therefore it is advisable to take the affinities between edge groups into account when calculating the value $w_b(G_i)$.

To compute the w_b value for a proposal b, Edge Boxes partitions the set of edge groups G into three disjoint subsets $N_b \cup I_b \cup E_b = G$. I_b represents the subset of edge groups that intersect with the boundary of the proposal b. N_b and E_b represent the subset of edge groups which are wholly located outside and inside of the box b, respectively. The w_b values are set as follows:

$$w_b(G_i) = \begin{cases} 0 & \text{if } G_i \in N_b \cup I_b \\ 1 - \max_T \prod_{k=1}^{|T|-1} A_{k,k+1} & \text{if } G_i \in E_b \end{cases} \tag{3.11}$$

where T represents a path of edge groups that starts at $G_i \in E_b$ and ends at some $G_j \in I_b$. T, therefore, represents a sequence of edge groups (indexed by k) that connects G_i most strongly to an edge group that intersects with the boundary of the box b. Note, that $w_b(G_i) = 1$ in the case of an edge group in $G_i \in E_b$ that has no connection to an edge group on the box boundary.

The score s_b for a proposal b is determined by summing over the w_b values weighted by the magnitudes of the edges $m(G_i)$ in the following way:

$$s_b = \frac{\sum_{G_i \in G} w_b(G_i) m(G_i)}{2(b_w + b_h)^\kappa} \tag{3.12}$$

where $m(G_i)$ represents the sum of edge magnitudes in edge group G_i. b_w and b_h represent width and height of box b, respectively. Since edges have a width of one pixel regardless of scale, Edge Boxes normalizes the score by the perimeter of the box rather than its area. However, the number of edge groups contains in a box is proportional to its area. Therefore, the scoring function is biased towards larger boxes. $\kappa = 1.5$ is supposed to counteract this bias.

The authors [85] propose an extension to the scoring function which produces slightly better results. They observe that edges located at the center of a proposal box are not as important as edges near the boundary. In order to award more importance to edges at the boundary they propose to amend s_b by removing edges near the center of the proposal box from the scoring function in the following way:

$$s_b^* = s_b - \frac{\sum_{p \in b_c} m_p}{2(b_w + b_h)^\kappa} \tag{3.13}$$

where b_c is a proposal box of width $\frac{b_w}{2}$ and height $\frac{b_h}{2}$ that is centered within the original proposal box b. m_p refers to the magnitude of the edge at location p.

We also adopt this extended scoring function s_b^* for all our experiments.

3.3.3 Search strategy and proposal refinement

The scoring function provides a heuristic which allows ranking the likelihood of any given proposal containing an object. The boxes are obtained through a sliding window search.

For a given box size, the horizontal stride d_w between two proposal boxes is given by

$$d_w = \frac{b_w(1-\delta)}{1+\delta} \tag{3.14}$$

where b_w is the width of the box. This results in two neighboring boxes with an intersection over union of δ. The vertical stride d_h is similarly given by substituting b_w with the height of the box b_h. We follow the original authors [85] recommendation by settings $\delta = 0.65$.

The sliding window approach is repeated for different scales and aspect ratios. Aspect ratios α are selected from a set $\alpha \in \left\{\frac{1}{k}, k | k = 1 \dots \tau\right\}$. We follow the original authors in using $\tau = 3$. Scales are defined through proposal box area. As minimum object size, a proposal box area of $\sigma_{min} = 1000$ pixels – which corresponds to a side length of approx. $32px$ for an aspect ratio of 1 – is used and the maximum size σ_{max} is set to the area of the image. The scale space is sampled in such a way that two consecutive scales σ_a and σ_b are related by $\sigma_b = \alpha\sigma_a$. α controls density of the scale space sampling and is usually set to $\alpha = 2$ resulting in N_s scales:

$$N_s = \left\lceil \log_2 \frac{\sigma_{max}}{\sigma_{min}} \right\rceil \tag{3.15}$$

For a typical $1024px \times 1024px$ image in the FlickrLogos-47 dataset this corresponds to $N_s = 11$ scales. With a stride between two proposal boxes based on $\delta = 0.65$ this results in approx. $47,000$ proposal boxes. For $\tau = 3$ we need to consider 5 aspect ratios which results in approx. $235,000$ proposal boxes per image for which we need to evaluate the scoring function.

Additionally, proposal boxes which have received a score $s_b^* >= 0.01$ are being subjected to a box refinement step which aims to provide better localized proposals. The refinement is a greedy recursive search over deformations of the original box while in each recursive step the search window is reduced. Initially, the width of the original box is extended by $\frac{d_v}{2}$, once to the left and once to the right. Likewise, the height of the original box is extended by $\frac{d_h}{2}$, once to the top and once to the bottom. This results in four deformed versions of the original box. Each of these deformed boxes is scored using the metric described in chapter 3.3.2. If one of the deformed boxes receives a higher score as the original box, we select the box with the highest score. The process is repeated using this selected box as a reference in combination with reduced search windows of $\frac{d_v}{4}$ and $\frac{d_h}{4}$, respectively.

3.3.4 Discussion of Selective Search and Edge Boxes

On the face of it, both object proposal algorithms operate on similar principles: Without a semantic understanding of the image content, heuristic object proposal algorithms can only operate on low-level features such as region similarity.

However, both algorithms use these features in different ways: While Selective Search explicitly defines similarity metrics between image regions, Edge Boxes operates on edges which are broadly defined as strong dissimilarities between image regions. Edge Boxes never explicitly defines a measure of dissimilarity between image regions. Instead, the dissimilarity is implicitly measured through an edge detector. The original work uses an edge detector based on a structured forest classifier, which operates on small image patches. Therefore the implicit measure of dissimilarity has no semantically palpable interpretation.

Furthermore, in order for the edge grouping algorithm to work, the obtained edges need to be binarized which involves setting a confidence threshold. This threshold may somewhat limit the sensitivity of Edge Boxes for objects that are not clearly delineated from the background through strong edges. By contrast, Selective Search has no such hard limitation. Instead, the similarity between two regions – which is explicitly defined – determines the order of merger. Image regions which are not too dissimilar to each other are typically merged first, while image regions which are most dissimilar will be merged later.

The downside of Selective Search is the lack of a scoring mechanism which is based directly on image content. Instead, the scoring is based on the order of merger. Regions which are merged later tend to receive a higher score. Objects tend to have a high dissimilarity with the background which controls the order of merger and only indirectly influences the score of the proposal.

To summarize, in Edge Boxes the measure of dissimilarity is defined indirectly while the scoring of proposals is defined directly. For Selective Search it is precisely the opposite: The measure of dissimilarity (or rather the measure of similarity) is defined explicitly while proposal ranking is defined indirectly.

3.4 Heuristic object proposals on FlickrLogos

3.4.1 Performance Evaluation

We have evaluated the performance of the two aforementioned object proposal algorithms on the FlickrLogos-47 dataset individually. Figure 3.4 shows the performance of Selective Search and Edge Boxes. On the left side, we show recall as a function of IoU. On the right side, we show the average recall for a fixed maximum number n of proposals.

While the two algorithms exhibit very similar overall performance as measured by average recall (AR), the maximum achievable recall tends to be higher for Selective Search than for Edge Boxes. Edge Boxes, however, tends to offer better localization.

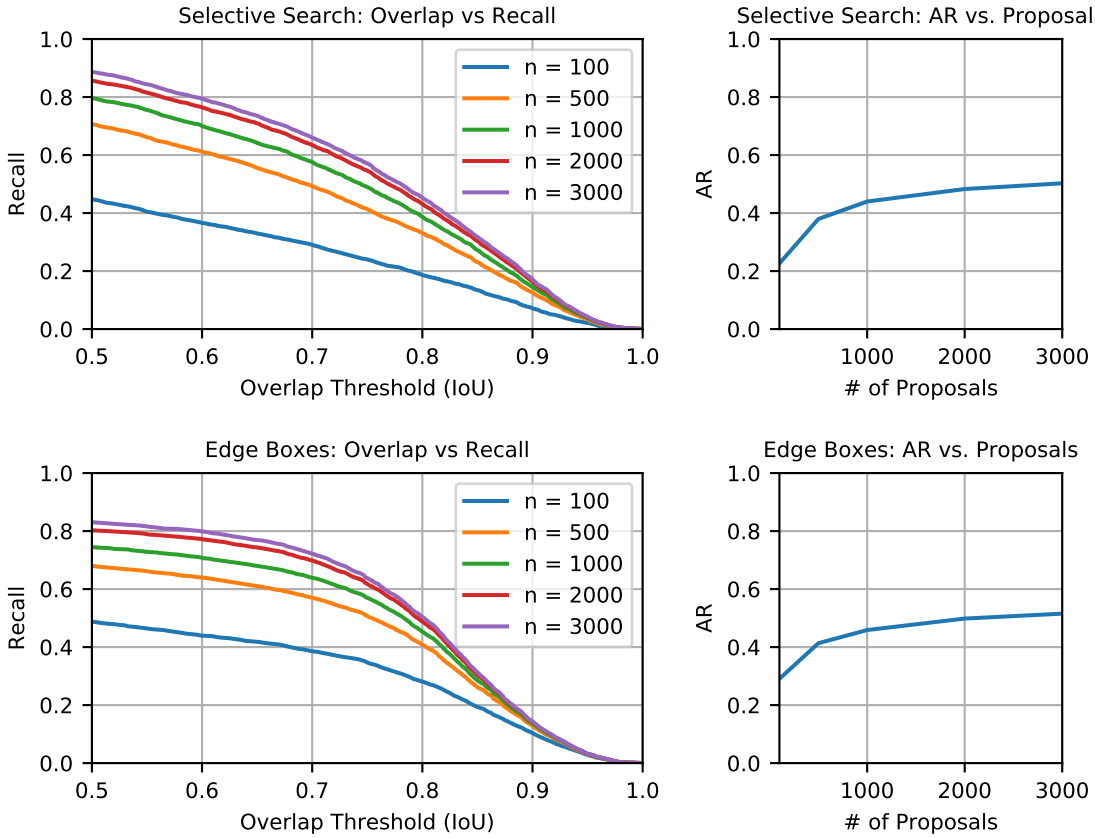


Figure 3.4: Performance of Selective Search and Edge Boxes on the FlickrLogos-47 dataset for different number of proposals.

Runtimes of the two algorithms are also very comparable. Selective Search averages around $0.90s$ per image and is slightly faster than Edge Boxes which averages around $0.98s$ per image[1].

Aside from better localization, another benefit of Edge Boxes is that it tends to work better if we only allow a small number of proposals. This property is not surprising, since contrary to Selective Search, Edge Boxes employs a scoring function that operates directly on the content of the proposals. However, this advantage of Edge Boxes quickly disappears if we allow more than 1000 proposals.

A common property of both algorithms is that increasing the number of proposals beyond $n = 2000$ only yields minor improvements. If not otherwise indicated, all following experiments use $n = 2000$ proposals.

[1]Runtimes were measured as single-core performance on an Intel Xeon E5-2680 @ 2.5 GHz

3.4.2 Selective Search Error Modes

As an object proposal algorithm, Selective Search is designed to replace an exhaustive sliding window search over all possible object positions and scales. In order to do so, it has to employ heuristics which may fail under some circumstances. Although it is based on heuristics, Selective Search is able to achieve an impressive recall on the FlickrLogos-47 dataset. However, there are cases, where Selective Search performs rather poorly or even has conceptual problems. In the following, we are going to analyze the most common error modes of Selective Search.

Selective Search relies on two major assumptions: One of them being that an object can be distinguished from the background by the employed similarity metrics. The other assumption is that the object can in principle be built up by iteratively merging only adjacent regions. As we will see in the following, especially the second assumption will turn out problematic in the context of company logo detection.

Error Mode (A): Spatially divided components on uniform background One of the most common failure modes can be observed when an object consists of multiple spatially divided regions on a uniform background. This situation is not as common in real-world pictures as it is with company logos where it can often be observed for text-based logos. An example of this case can be seen in Figure 3.5. Selective Search will be able to generate a proposal for each individual letter. Once all letters are identified, Selective Search has no other option besides merging one letter with the background, since Selective Search only considers adjacent regions. However, once the region corresponding to a letter is merged with the background, Selective Search is highly unlikely to generate the correct proposal. The only way for Selective Search to recover from this situation is when the background region happens to roughly match the shape of the object. In this particular example, it is impossible to produce the correct proposal because from this level in the hierarchy on, all bounding boxes must be at least as large as the background. This is a conceptual problem with Selective Search that cannot be addressed within the framework of the algorithm (e.g., by adding more sophisticated similarity metrics).

Error Mode (B): Partially occluded objects A less common error mode is objects which are partially occluded by a partially transparent structure (e.g., a metal gaze). An example of this error mode can be seen in Figure 3.5. Although the setting of the image is different, this error mode is closely related to the previous one since the underlying problem is the same: The metal gaze itself is relatively uniform, meaning regions on the gaze are likely to be merged first. However, at some point during the algorithm, a region needs to be merged with the region of the surrounding gaze. Once this has happened, Selective Search is unlikely to find the correct proposal.

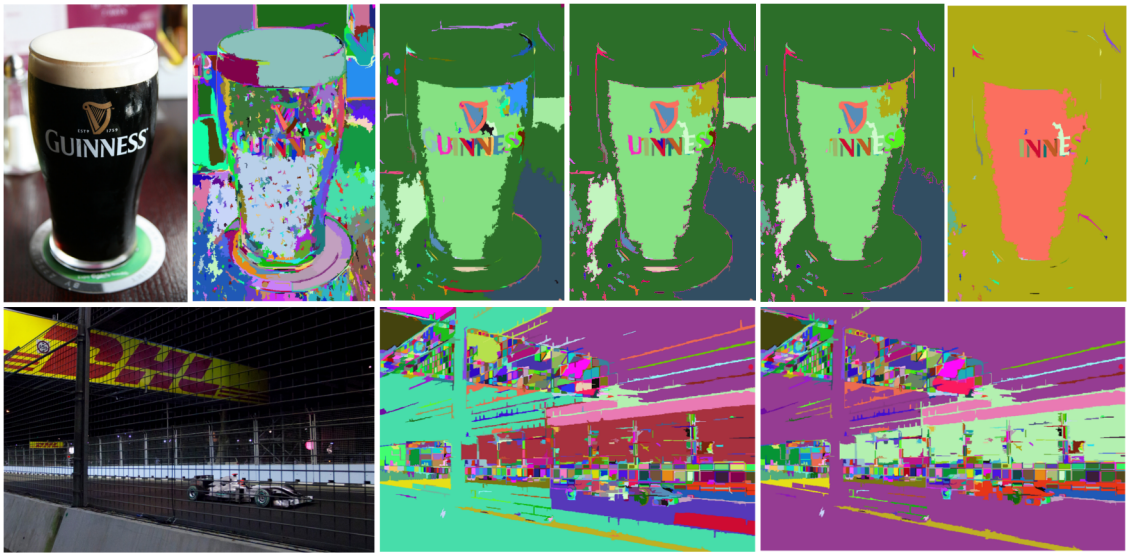


Figure 3.5: Illustration of Selective Search error modes. **Top:** Error mode (A). Once a letter is merged with the background, further iterations are unable to generate the correct object proposal. **Bottom:** Error mode (B). Once a region is merged with the foreground mesh, Selective Search is unlikely to recover.

3.4.3 Object Proposals for Text-based Company Logos

We have shown some instances where Selective Search has difficulties producing the correct object proposal. These error modes are often connected to text-based company logos and arise from the fact that Selective Search only considers adjacent regions for merging.

Edge Boxes does not have this shortcoming since it uses a dense grid of candidate boxes of fixed scales and aspect ratios and scores them by counting the number of enclosed contours in a given candidate box. Therefore, Edge Boxes is in principle able to capture text-based logos, provided a suitable candidate box is defined by the grid. However, text-based logos often are associated with rather extreme aspect ratios. A way to counter this problem would be to modify the grid of candidate boxes to include these aspect ratios. This approach quickly leads to an overwhelmingly large number of candidate boxes. In the following, we will propose a simple algorithm to explicitly model text-based logos. We call this algorithm, designed to address error mode (A) of Selective Search, VH-connect.

The VH-connect Algorithm

VH-connect starts by computing the morphological gradient of the greyscale input image.

$$G_m(x, y) = [f \oplus b](x, y) - [f \ominus b](x, y) \tag{3.16}$$

(a) (b) (c) (d)

Figure 3.6: Illustration of the VH-connect algorithm. (a) Input image. (b) Morphological gradient. (c) Binarized image after applied Otsu threshold. (d) Result after a single closure step in horizontal direction. Each connected component becomes a candidate for an object proposal.

where $\oplus$ represents the dilation and $\ominus$ the erosion operation using an elliptical structuring element b. We use Otsu's method [62] for determining a threshold for binarizing the gradient image $G_m(x, y)$ into $G(x, y)$, maximizing the intra-class variance between edge values and non-edge values.

Text usually produces strong gradients at the edges of individual letters. For a given scale and text direction, these gradients are usually spaced apart reasonably regularly. In order to identify regions containing text, we need to close these characteristic spacings. In order to do so, we apply a morphological closing operation $\bullet$ to the binarized gradient image $G(x, y)$. In order to capture text on multiple scales as well as vertical and horizontal text, we repeatedly use this closing operation with multiple rectangular structuring elements k_l for different scales l. For each scale l the structuring elements have the form $1 \times l$ and $l \times 1$.

$$C_{k_l}(x, y) = [G \bullet k_l](x, y) = [(G \oplus k_l) \ominus k_l](x, y) \tag{3.17}$$

After each closure step, we retrieve connected components c_i as candidates for object proposals. These candidates are filtered by the following heuristic which is similar to the *region fill* metric used in Selective Search (see chapter 3.2.1).

$$\frac{size(c_i)}{bb(c_i)} > t \tag{3.18}$$

where $size(c_i)$ refers to the number of pixels in the connected component c_i and $bb(c_i)$ refers to the number of pixels within the bounding box around c_i. We found $t = 0.3$ to be a suitable threshold.

A visualization of the algorithm can be found in Figure 3.6. On average VH-connect generates between 200 and 400 proposals per image.

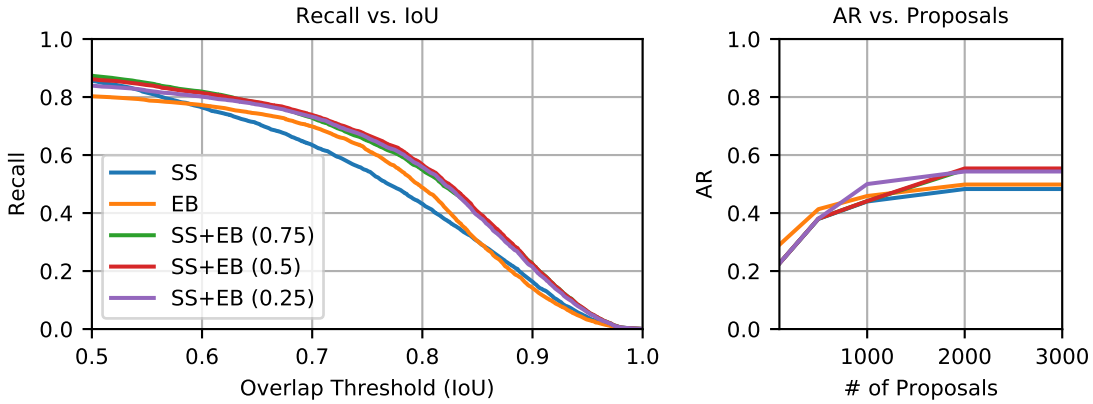


Figure 3.7: Performance gain by combining object proposals generated by Selective Search and Edge Boxes for different mix ratios ($n = 2000$). The ratios refer to the fraction of Selective Search boxes compared to all boxes. Color code is identical for both plots.

3.4.4 Improving heuristic Object Proposals

As we have shown in chapter 3.4.1 both Selective Search and Edge Boxes have similar performance when evaluated individually. We also have demonstrated some of the error modes of Selective Search in chapter 3.4.2, and we have speculated that Edge Boxes might provide better performance in these circumstances.

In the following we want to evaluate two things: If we limit ourselves to $n = 2000$ proposals, can we benefit from diversifying our object proposal strategy? That is: Can we increase performance when we incorporate proposals from both Selective Search and Edge Boxes? Can we further improve proposal performance by explicitly modeling text logos by including proposals from the VH-connect algorithm?

Diversifying object proposals In order to investigate this possibility, we select the top proposals from both approaches with different proportions so that the total number of proposals is $n = 2000$. The results of this experiment are shown in Figure 3.7. We abbreviate Selective Search as SS and Edge Boxes as EB. When mixing proposals from multiple sources we include the fraction of Selective Search proposals relative to all proposals. For example, the label SS+EB (0.75) refers to a mixture of 75% Selective Search proposals and 25% Edge Boxes proposals.

As it can be seen from Figure 3.7 it is almost always beneficial to combine proposals from both Selective Search and Edge Boxes. Both the localization and maximum achievable recall show improvements. There are two noteworthy exceptions to this pattern:

If only few proposals are allowed, Edge Boxes by itself tends to perform better (measured by AR) than a mixture of different proposals. This is not surprising since because of its scoring algorithm, Edge Boxes has an advantage in selecting the most

Proposals	Ratio SS	Ratio EB	Ratio VH	MR	AR	MABO
SS	1.00 (2000)	0.00 (0)	0.00 (0)	0.857	0.483	0.722
EB	0.00 (0)	1.00 (2000)	0.00 (0)	0.802	0.498	0.722
SS+EB	0.75 (1500)	0.25 (500)	0.00 (0)	**0.874**	0.549	**0.753**
SS+EB	0.50 (1000)	0.50 (1000)	0.00 (0)	0.862	**0.554**	0.749
SS+EB	0.25 (500)	0.75 (1500)	0.00 (0)	0.839	0.543	0.734
SS+EB+VH	0.70 (1400)	0.20 (400)	0.10 (200)	**0.888**	**0.574**	**0.767**
SS+EB+VH	0.45 (900)	0.45 (900)	0.10 (200)	0.874	0.570	0.764
SS+EB+VH	0.20 (400)	0.70 (1400)	0.10 (200)	0.856	0.568	0.752

Table 3.1: Performance measures of combined object proposals. MR (max. recall) refers to the recall at $IoU = 0.5$. In all cases the total number of proposals is $n = 2000$. Ratios are always given with respect to n. Diversification of object proposals clearly is beneficial in all performance measures, as is the inclusion of VH-connect. For best performance the fraction of Selective Search proposals should be larger than the fraction of edge box proposals.

promising proposals. If we mix Selective Search boxes into this list, it will displace some of the highly-scored boxes in Edge Boxes. Since in the case of few proposals Selective Search on average performs worse than Edge Boxes, the average recall is bound to suffer.

Another exception is when edge box proposals make up the majority of proposals. Since Selective Search on its own can deliver a very high maximum recall, the maximum achievable recall suffers when many edge box proposals are used.

However, for $n = 2000$ proposals and a Selective Search ratio of 0.75 the combination of Edge Boxes and Selective Search outperforms the performance of each individual proposal algorithm in all metrics. This is shown in Table 3.1.

Evaluating VH-connect In order to demonstrate the benefits of the VH-connect algorithm, we select the top-ranked object proposals from Selective Search, Edge Boxes and VH-connect with different proportions while keeping a maximum of $n = 2000$ proposals. Table 3.1 shows the results of this experiment.

It becomes clear that the inclusion of VH-connect is always beneficial under all metrics despite the fact that the number of proposals from Selective Search and Edge Boxes need to be reduced in order to keep a maximum of $n = 2000$ proposals. Similar to the situation with Selective Search and Edge Boxes only, the diversification is most successful when a majority of object proposals are taken from Selective Search.

3.4.5 Conclusions

Both Selective Search and Edge Boxes are able to deliver acceptable proposal performance for the task of company logo detection. We have analyzed the error modes of Selective Search and have shown how Edge Boxes can complement Selective Search in principle. We have experimentally verified that diversifying proposals by using multiple proposal algorithms is beneficial to the overall proposal performance. Finally, we were able to show that explicitly modeling text-based logos can further increase the performance of object proposals.

3.5 Trainable Object Proposals

All object proposal algorithms we have evaluated so far were heuristic approaches. Edge Boxes, for example, uses the number of enclosed contours as a heuristic for determining likely object locations. Selective Search uses the order of merger based on hand-crafted similarity measurements to identify potential object locations.

In this chapter, we want to extend our examination of object proposal algorithms to include machine learning-based approaches to object detection. In contrast to heuristic approaches, we do not rely on hand-crafted features, scoring functions or similarity metrics. For a given region of interest, we want to directly learn a mapping from the pixel data to the likelihood of it containing an object. Specifically, we want to look at one particular approach for generating object proposals: Region Proposal Networks (RPNs).

3.5.1 Region Proposal Networks

On an abstract level, Region Proposal Networks (RPNs) place a dense grid of bounding boxes of various scales and aspect ratios across the input image. Each of these bounding boxes is a potential object proposal and is called an *anchor box*. Consequently, the grid is often called the *anchor grid*.

The task of an RPN is two-fold: One task is to predict a score for each anchor box which measures the likelihood of the box containing an object. Another task is to refine the boundaries of each anchor box to cover the nearest object better. This process is called *bounding box regression*.

RPNs are implemented as a fully-convolutional neural network. Most often, a traditional classification network such as VGG16 [72] or a Resnet [36] are used as a basis for RPNs. After pre-training on classification tasks such as ImageNet [15], these networks are usually cut off after the last convolutional layer.

The RPN consists of three convolutional layers that are added to the last feature map, which for VGG16 is the *conv5* feature map. One layer is a convolution with a 3×3 kernel which is directly attached to the *conv5* feature map. The number of channels can vary and is usually set to the same number as the feature map to which it is attached. The other two layers are 1×1 convolutions which are attached in

parallel to the newly added layer: One layer is responsible for predicting a score for each anchor box, and the other layer predicts the values used for the bounding box regression. We will describe these layers in greater detail in the following subsections.

The Anchor Grid

In the following, we will introduce the anchor grid more formally. For clarity, we will distinguish coordinates on the feature map $\mathcal{F}$ from coordinates on the input image $\mathcal{I}$. We denote image coordinates as $\boldsymbol{x}_\mathcal{I} = (x_\mathcal{I}, y_\mathcal{I})^T$ and feature map coordinates of the network as $\boldsymbol{x}_\mathcal{F} = (x_\mathcal{F}, y_\mathcal{F})^T$. Both coordinate systems have their origins at the upper-left corner of the image with the positive x-axis extending across the image width and the positive y-axis covering the height of the image. Since deep neural networks usually employ downsampling, the image coordinate system and the feature map coordinate system are related through a downsampling factor $\alpha > 1$.

We need to associate every coordinate on the feature map $\mathcal{F}$ with the center of an anchor box. Since anchor boxes need to be specified in image coordinates, we need to establish a mapping between feature map coordinates and image coordinates. Because only integer values can serve as valid coordinates, the mapping $\mathcal{I} \to \mathcal{F}$ induced by the downsampling factor α is not injective: Therefore, the inverse mapping $\mathcal{F} \to \mathcal{I}$ is ill-defined since every pixel in $\mathcal{F}$ would be mapped to a group of pixels in $\mathcal{I}$. To obtain an inverse mapping $\mathcal{F} \to \mathcal{I}$ we need to pick a single representative from this group of pixels. An obvious choice for this representative is the central pixel of each pixel group in $\mathcal{I}$ that is mapped to the same pixel in $\mathcal{F}$.

Since the downsampling factor is usually a multiple of 2, a central pixel often does not exist. In this case, we settle for the closest pixel on the upper right. Therefore the mapping $f_\mathcal{I} : \mathcal{F} \to \mathcal{I}$ from the feature map $\mathcal{F}$ to a coordinate on the image $\mathcal{I}$ is given by the following relationship:

$$\boldsymbol{x}_\mathcal{I} = f_\mathcal{I}(\boldsymbol{x}_\mathcal{F}) = \alpha \boldsymbol{x}_\mathcal{F} + \left\lfloor \frac{\alpha}{2} \right\rfloor \mathbf{1} \tag{3.19}$$

Anchor boxes are characterized by a set of scales $\mathcal{S}$ and a set of aspect ratios $\mathcal{A}$. At every location $\boldsymbol{x}_\mathcal{I}$ a set of anchor boxes based on $\mathcal{S} \times \mathcal{A}$ is constructed with $\boldsymbol{x}_\mathcal{I}$ as their center. We choose to parametrize the anchor box scales in terms of $\sqrt{area}$ and the aspect ratio in terms of $height/width$. Therefore, the upper-left and lower-right vertices $\boldsymbol{x}_{ul}$ and $\boldsymbol{x}_{lr}$ of the anchor boxes for a given feature map position $\boldsymbol{x}_\mathcal{F}$, scale $s \in \mathcal{S}$ and aspect ratio $a \in \mathcal{A}$ are given by:

$$\begin{aligned} \boldsymbol{x}_{ul} &= f_\mathcal{I}(\boldsymbol{x}_\mathcal{F}) - \frac{s}{2} \begin{pmatrix} \sqrt{a}^{-1} \\ \sqrt{a} \end{pmatrix} \\ \boldsymbol{x}_{lr} &= f_\mathcal{I}(\boldsymbol{x}_\mathcal{F}) + \frac{s}{2} \begin{pmatrix} \sqrt{a}^{-1} \\ \sqrt{a} \end{pmatrix} \end{aligned} \tag{3.20}$$

To see why these vertices results in an anchor box of aspect ratio a and scale $\sqrt{area}$ we look at the diagonal $\boldsymbol{d} = (d_x, d_y)^T$ of the anchox box which is given by:

$$\boldsymbol{d} = \boldsymbol{x}_{lr} - \boldsymbol{x}_{ul} = f_{\mathcal{I}}(\boldsymbol{x}_{\mathcal{F}}) + \frac{s}{2}\begin{pmatrix}\sqrt{a}^{-1}\\ \sqrt{a}\end{pmatrix} - f_{\mathcal{I}}(\boldsymbol{x}_{\mathcal{F}}) + \frac{s}{2}\begin{pmatrix}\sqrt{a}^{-1}\\ \sqrt{a}\end{pmatrix} = s\begin{pmatrix}\sqrt{a}^{-1}\\ \sqrt{a}\end{pmatrix} \tag{3.21}$$

The aspect ratio is given by the height d_y and width d_x of the diagonal:

$$\frac{d_y}{d_x} = \frac{s\sqrt{a}}{s\sqrt{a}^{-1}} = \sqrt{a}^2 = a \tag{3.22}$$

The scale s is defined as $\sqrt{area}$ which can also be obtained from $\boldsymbol{d}$ by

$$\sqrt{d_x d_y} = \sqrt{s\sqrt{a}^{-1} s\sqrt{a}} = \sqrt{s^2} = s \tag{3.23}$$

This mapping allows us to associate every position in the feature map with a set of anchor boxes in image coordinates. For every anchor box defined in this manner, we compute the maximum intersection over union (IoU) with any groundtruth annotation, regardless of its class. If the maximum (IoU) is greater or equal 0.5 we classify this anchor box as a positive example and as a negative example otherwise.

Anchor Score Prediction

The anchor grid defines a set of anchor boxes (in image coordinates) for every position of the feature map. We can divide these anchor boxes into positive and negative examples based on their intersection over union with the groundtruth annotations. In this chapter, we will discuss how RPNs use this information to predict the likelihood that these anchor boxes contain objects.

As mentioned in chapter 3.5.1, RPNs add three additional convolutional layers to the last feature map of an existing network, resulting in two additional network outputs. One of these layers is responsible for predicting anchor box scores. This score prediction layer outputs predictions with the same spatial resolution as the final feature map, and the number of channels is chosen according to the number of anchor boxes to predict at each location. Since we need to predict a probability distribution over the outcomes *background* and *object* for every anchor box, we employ two output neurons per anchor box. If $\mathcal{S}$ represents the set of scales and $\mathcal{A}$ the set of aspect ratios, the score prediction layer, therefore, needs to output $2|\mathcal{S} \times \mathcal{A}|$ channels.

In order to obtain a probability distribution from the predictions, they are fed into a softmax operation. However, since every location of the feature map produces multiple probability distributions – one for each anchor – there exists no axis across which the softmax can be performed. Therefore, the predictions need to re-interpreted before the softmax can be applied:

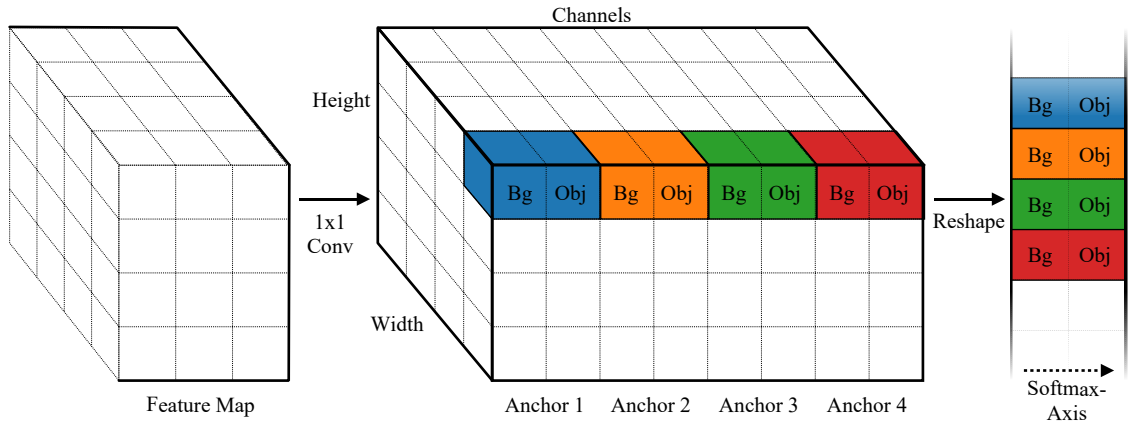


Figure 3.8: Score prediction for Region Proposal Networks (RPNs). For every location on the feature map, the RPN predicts a probability distribution over the outcomes *background* (Bg) and *object* (Obj) for every anchor. Predictions are reinterpreted through a reshape operation to expose the softmax axis.

Let $H_{\mathcal{F}}$ and $W_{\mathcal{F}}$ represent the height and width of the feature map $\mathcal{F}$. If the shape of the network output is $H_{\mathcal{F}} \times W_{\mathcal{F}} \times (2|\mathcal{S} \times \mathcal{A}|)$, a reshape operation changes the output interpretation to $H_{\mathcal{F}} \times W_{\mathcal{F}} \times |\mathcal{S} \times \mathcal{A}| \times 2$. After the reshape operation the softmax can be applied across the last dimension. This is illustrated in Figure 3.8.

The softmax allows the network output for each anchor to be interpreted as a probability distribution $\boldsymbol{p} = (p_{bg}, p_{obj})$ where p_{bg} and p_{obj} represent the probabilities of an anchor belonging to the *background* and *object* class, respectively. During testing, we are only interested in p_{obj} which we use as a confidence score for the proposal. During training, we need to define a loss function on this output which compares our predicted probability distribution with a target distribution. Since we know the correct class labels for each anchor box with certainty, the values of the target probability distribution $\hat{\boldsymbol{p}} = (\hat{p}_{bg}, \hat{p}_{obj})$ are binary: The correct class has probability 1 while the other class has probability 0. A common way to define a loss function on pairs of probability distributions is the cross-entropy loss $H(\boldsymbol{p}, \hat{\boldsymbol{p}})$:

$$H(\boldsymbol{p}, \hat{\boldsymbol{p}}) = -\sum_{i \in \{bg, obj\}} \hat{p}_i \log p_i \tag{3.24}$$

Since anchor score prediction is a binary classification problem there exists an alternative problem formulation with sigmoid functions: In this formulation, it suffices to use a single output neuron per anchor instead of two. The shape of the network output would therefore be $H_{\mathcal{F}} \times W_{\mathcal{F}} \times (|\mathcal{S} \times \mathcal{A}|)$. In such a case, the network output does not need to be reshaped to apply the softmax. Instead the output for an anchor y is squashed into the interval $]0, 1[$ by a sigmoid function such as the logistic function $L(y)$:

$$L(y) = \frac{1}{1 + e^{-y}} \tag{3.25}$$

$L(y)$ can be interpreted as the probability p_{obj} in our previous example and since we have a binary classification problem $p_{bg} = 1 - p_{obj}$ is implicitly defined. The cross-entropy loss can now be applied in the same way as above. While this approach might seem more natural, the original approach by [66] uses a softmax cross-entropy loss. We examine these two options in the context of a simple regression problem in Appendix C and conclude that a two output neuron model for a binary classification problem is more desirable since it tends to converge faster.

Bounding Box Regression

The score prediction layer of the RPN is able to score anchor boxes from the anchor grid by their likelihood of containing an object. Object proposals generated by this method are limited in size and shape by the specification of the anchor grid. While often being able to identify regions of interest, the object box proposals are often not well localized and have a poor fit. Bounding box regression is designed to overcome this problem by allowing individual anchor boxes to be modified in position and shape.

For this purpose, we predict four values for every anchor box at every location. Two of these values encode an offset in x- and y-direction and two values encode scale factors by which to scale the width and height of the bounding box. We follow the parametrization by [66] in which the offsets are predicted relative to the width and height of the bounding box and the scale factors for width and height are predicted on a logarithmic scale:

$$t_x = \frac{x_{gt} - x_a}{w_a} \qquad t_y = \frac{y_{gt} - y_a}{h_a} \tag{3.26}$$

$$t_{s_w} = \log \frac{w_{gt}}{w_a} \qquad t_{s_h} = \log \frac{h_{gt}}{h_a} \tag{3.27}$$

Here, t_x and t_y are the regression targets for the offsets and t_{s_w} and t_{s_h} are the regression targets for the scale factors. (x_{gt}, y_{gt}) and (x_a, y_a) refer to the position of the groundtruth annotation and the associated anchor box, respectively. (w_{gt}, h_{gt}) and (w_a, h_a) represent width and height of the groundtruth and the anchor. In this context, bounding box positions are measured by their centers.

This parametrization ensures that the mean value for each of these regression targets is approximately zero: The offsets are expected to be distributed according to a Gaussian distribution around zero while the scale factors are expected to follow a Gaussian distribution around one. Therefore, the logarithmic scaling maps the expected mean value of the scale factors to zero. All values are predicted relative to the width or height of the anchor box which naturally limits the value range of

the regression targets since the deformation are usually small compared to the size of the anchor box.

In terms of network architecture, the bounding box regression is realized as an additional convolutional layer which is attached to the last feature map of the network. This convolutional layer outputs predictions of the same spatial resolution as the feature map and $4|\mathcal{S} \times \mathcal{A}|$ channels. During training, the *Huber loss* [41]

$$L^{\delta}_{hub}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta(|y - \hat{y}|) - \frac{1}{2}\delta & \text{else} \end{cases} \tag{3.28}$$

is used. In contrast to the more traditional L_2-loss whose derivative can produce arbitrarily strong magnitudes, the Huber loss has an upper limit on the value of its derivative. The *Huber loss* (often also called *smooth L_1-loss*) provides a constant slope of $\pm\delta$ if the predicted value y differs strongly from the actual value $\hat{y}$. This is a desirable property for fine-tuning neural networks since strong gradient magnitudes could destroy the pre-trained weights, especially during the first few training iterations.

Training protocol

We use our batch-normalized VGG16 network which has been pre-trained on ImageNet as a basis for our RPN. Training of the RPN consists of fine-tuning the VGG16 network with attached RPN module for 40000 iterations. We start with a learning rate of $\lambda = 0.001$ and perform a learning rate reduction by a factor of 10 after 25000 iterations. In each iteration, we process a single image.

When training a network for image classification (e.g., on ImageNet), images are usually grouped into *mini-batches*. Gradients are computed with respect to such a mini-batch. The number of images in this mini-batch is usually called the *batch size*. In a classification scenario, every image corresponds to a training example. Therefore, this definition of batch size seems intuitive. However, fully-convolutional networks such as RPNs can produce many predictions for a given image: Every location on the feature map and every anchor constitutes a potential training example. As we will discuss below, we only use a subset of potential examples during training. We call the examples which are used for training *active examples*. Examples which are not used for training are called *inactive examples*. Therefore, we define *batch size* as the number of active examples per image.

During training we need to address the large class imbalance between anchor boxes of class *background* and class *object*: If we were to draw an anchor box from the anchor grid randomly, the probability to draw a box from the *background* class is overwhelming. This imbalance can pose a problem for the classifier: By merely classifying all anchor boxes – regardless of content – as *background*, the classifier can easily minimize its loss function without learning to solve the problem.

In order to counter this effect we sample from the negative examples using the following strategy: We choose the total number of training examples $n_{total} = 128$

(positive and negative) to consider for each iteration. Our *batch size* is therefore equal to n_{total}. If we have identified $\hat{n}_{pos}$ positive examples in an image we sample $n_{pos} = \min\{\hat{n}_{pos}, n_{total}/2\}$. We choose $n_{neg} = n_{total} - n_{pos}$ negative examples.

For the object proposal stage, we use a simple random sampling strategy instead of a more sophisticated hard negative mining or even hard example mining strategy. Hard negative mining specifically selects negative examples that have been confidently classified as positive for training. Hard example mining is an extension of hard negative mining which additionally selects positive examples which have confidently been classified as negative. The reason for this decision is that hard negative mining tends to reduce the *false positive rate* (FPR) of a detector.

$$FPR = \frac{FP}{FP + TN} \tag{3.29}$$

While a low false positive rate is typically desirable for an object detection pipeline, it is not necessarily beneficial for object proposals. False positives (FPs) are only a problem for an object proposal algorithm if a FP prevents a correct proposal from appearing on the list of proposals. As we will see in Chapter 3.5.2, RPNs do not suffer from this problem as they generally require very few proposals to reach their maximum performance. However, by focusing on correctly classifying negative examples, hard negative mining can potentially increase the *miss rate* or *false negative rate* (FNR) of the classifier.

$$FNR = \frac{FN}{FN + TP} \tag{3.30}$$

Therefore, the potential performance benefits of hard negative mining are almost non-existent while the potential drawbacks are a very real possibility. Mining for positive examples is also unlikely to yield any benefits because for most images the number of positive anchor boxes is very small. We rarely exceed $n_{total}/2$ positive examples which means that while processing a mini-batch, we effectively already use all positive examples for training anyways. We therefore apply hard example mining only to the classification stage.

Pruning proposals

The raw output of an RPN during testing consists of a set of bounding boxes $\mathcal{B} = \{b_1, \ldots, b_N\}$. Each bounding box is associated with a confidence score $s : \mathcal{B} \to [0, 1]$. Every bounding box originates from an anchor box which has undergone bounding box regression. Therefore N equals the number of anchor boxes on the anchor grid. The confidence score indicates the likelihood of the bounding box containing an object instance with sufficient IoU.

Depending on the size of the image and the choice of scales $\mathcal{S}$ and aspect ratios $\mathcal{A}$, N can be quite large. In order for RPNs to provide usable object proposals, this

Algorithm 1 Non-maximum suppression

procedure NONMAX_SUPPRESSION($(\mathcal{B}, s), t_{nms}$)
  $\mathcal{B}_{nms} \leftarrow \emptyset$ ▷ Result set
  while $\mathcal{B} \neq \emptyset$ **do**
    $\hat{b} \leftarrow \arg\max_{b \in \mathcal{B}} s(b)$ ▷ Select most highly scored proposal
    $\mathcal{B}_{nms} \leftarrow \mathcal{B}_{nms} \cup \hat{b}$ ▷ Add proposal to result set
    $\mathcal{R} \leftarrow \emptyset$ ▷ Set of proposals to remove
    for $b_i \in \mathcal{B}$ **do** ▷ Suppress all proposals which overlap strongly
      if $IoU(\hat{b}, b_i) \geq t_{nms}$ **then**
        $\mathcal{R} \leftarrow \mathcal{R} \cup b_i$
    $\mathcal{B} \leftarrow \mathcal{B} - \mathcal{R}$ ▷ Note that $\mathcal{R}$ always contains $\hat{b}$ for $t_{nms} \leq 1.0$
  return $\mathcal{B}_{nms}$

set of bounding boxes needs to be pruned. This proposal pruning happens in three steps.

To speed up subsequent post-processing steps as much as possible, the first pruning step consists of removing all but the most highly-ranked proposals. We limit ourselves to the top 8000 proposals.

These filtered proposals undergo a non-maximum suppression (see Algorithm 1). The non-maximum suppression removes bounding boxes which have a high intersection over union (IoU) with a higher-scored proposal. The maximum amount of allowed overlap between two bounding boxes is controlled by the non-maximum threshold parameter t_{nms}. While a low value for t_{nms} can be used to limit the number of proposals, it is important not to choose this parameter too low because it also limits the number of positive examples for the classification stage. If t_{nms} is chosen too low, it can also make the detection of objects which are partially occluded by other objects impossible.

A hard lower bound for the choice of t_{nms} is the non-maximum suppression of the classification stage: Typically, every classification pipeline performs another non-maximum suppression before outputting the final detections. The non-maximum threshold of the proposal stage must not be lower than the non-maximum threshold of the classification stage. For all our experiments with RPNs we set $t_{nms} = 0.7$ which seems to work quite well.

The final pruning step is a hard limit on the number of proposals. If not explicitly stated otherwise we always use the $n = 2000$ highest ranked non-maximum suppressed proposals.

3.5.2 Evaluating RPNs on FlickrLogos

In the original implementation [66] Region Proposal Networks use an anchor grid with aspect ratios $\mathcal{A}_{orig} = \{0.5, 1.0, 2.0\}$ and scales $\mathcal{S}_{orig}\,\{128px, 256px, 512px\}$. This is not a suitable anchor grid to use for the FlickrLogos-47 dataset since a

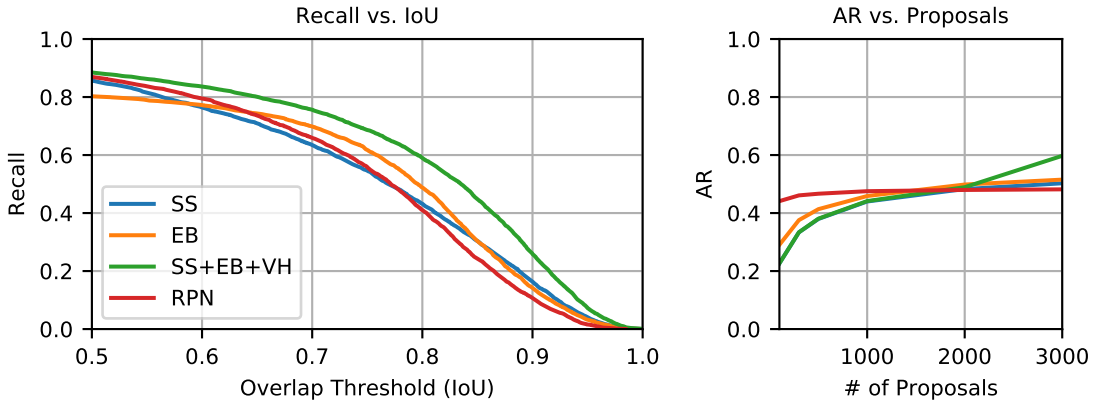


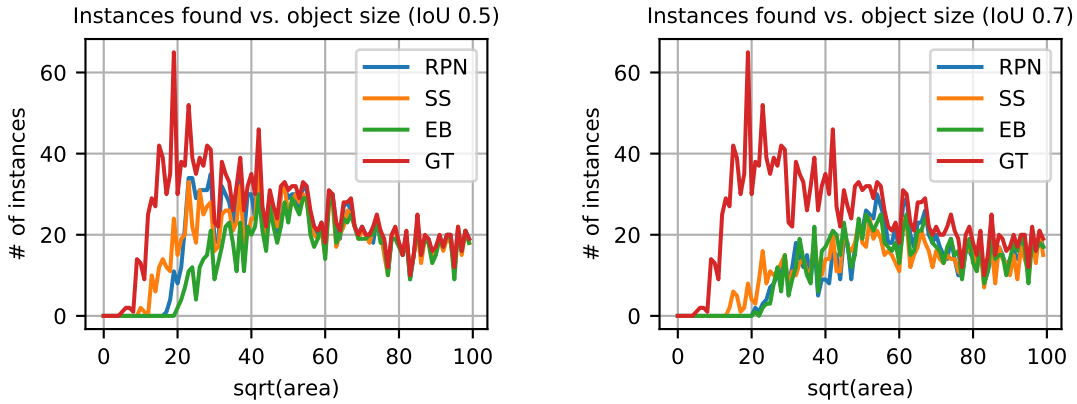
Figure 3.9: Performance of proposals generated by an RPN using anchor grid scales $\mathcal{S}_{ext}$. For $n = 2000$ proposals RPNs cannot offer better localization or maximum recall compared to our improved heuristic. However, RPN performance is a lot less sensitive to the number of proposals than heuristic approaches. RPNs are able to retrieve almost all relevant objects using very few proposals.

large fraction of logo instances are quite a bit smaller than $128px$ (see Chapter 2.1.2). We therefore adjust the anchor grid to include more suitable scales $\mathcal{S}_{ext} = \{32px, 64px, 128px, 256px\}$ while still following the same powers-of-two scheme as [66].

This adjustment yields vastly better results. The RPN performance for this set of scales $\mathcal{S}_{ext}$ for various number of object proposals is shown in Figure 3.9. It is clear from the graph that RPN performance is not very sensitive to the number of proposals: In fact, the performance is almost identical for any setting in which $n \geq 500$ proposals. The measurements make it clear that RPNs offer the remarkable ability to retrieve almost all relevant object instances with only a few proposals.

Figure 3.9 also shows a direct comparison of object proposals from RPNs with object proposals from heuristic approaches like Edge Boxes and Selective Search. For a fair comparison, we allow enough proposals ($n = 2000$), so that the heuristic approaches operate near their maximum performance. Surprisingly, RPNs do not offer superior performance in term of AR or max. recall. They also do not outperform Edge Boxes when it comes to object localization. Nevertheless, RPNs are the preferred method for generating object proposals for multiple reasons: (1) RPNs can archive the same performance than heuristic methods with fewer proposals. (2) RPNs can generate object proposals two orders of magnitude faster than Selective Search or Edge Boxes.

This advantage in speed is not due to lower computational complexity: Since RPNs are deep convolutional neural networks, they tend to be very compute intensive. The reason for the speedup is two-fold: (1) Deep neural networks are usually executed on GPU which speeds up the computation dramatically. (2) In most application scenarios, the computational effort for feature extraction through



(a) (b)

Figure 3.10: Number of retrieved object instances by size using RPN proposals. (a) IoU required for a detection is 0.5. (b) IoU required for a detection is 0.7. Number of proposals is $n = 2000$ in both cases. RPNs have difficulties in detecting small objects. Plots were generated from histograms with a bin width of $2px$. Small object are not localized as accurately as large objects.

a convolutional neural network has to be spent anyways. When viewed from this perspective, the computational effort for a few additional convolutional layers on top of the existing layers is negligible. In contrast, traditional heuristic proposal algorithms usually need to be executed in addition to the existing processing pipeline.

While RPNs do offer competitive performance, a closer analysis reveals some weaknesses: Figure 3.10 shows the successfully retrieved object instances as a function of object size. It is clear that RPNs do have difficulties retrieving small object instances. In the following chapter, we want to analyze RPNs more closely to identify the reason for this bad performance.

3.5.3 Analyzing the Anchor Grid

RPNs are fully-convolutional neural networks. As such, they can be thought of as a big non-linear filter that is applied to the input image in a strided fashion. The stride d between two filter applications is determined by the downsampling factor d^{-1} of the network while the size of the filter is equal to the receptive field of the network.

This means that the center coordinates of two adjacent anchor boxes centers are separated by d pixels. Since we use intersection over union (IoU) with groundtruth boxes as a criterion to divide anchor boxes into positive and negative samples the granularity of the anchor grid becomes important for small object instances. An anchor grid that is too coarse might fail to produce an anchor box with sufficiently high IoU. During training, such a case would result in a failure to use all available

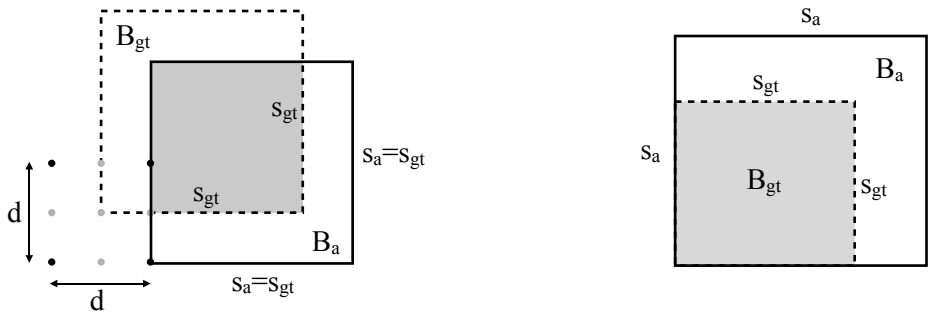


Figure 3.11: The situations shown above are used in our analysis of the RPN anchor grid. **(a)** A groundtruth instance B_{gt} and an ideal anchor box B_a of the same scale and aspect ratio. We look at the maximum displacement between these boxes by the anchor grid. **(b)** We examine the maximum difference in scale two boxes are allowed to have to produce a given intersection over union.

training examples. If we assume that the RPN can perfectly learn the task of predicting anchor boxes of sufficiently high IoU, the RPN will miss such an instance at evaluation time. Therefore, it makes sense to analyze the theoretical limits of the anchor grid with respect to its resolution.

For this purpose, we consider the situation depicted in Figure 3.11. We assume a quadratic groundtruth instance $\mathcal{B}_{gt}$ of side length s_{gt}. Since we only want to look at the effect of feature map resolution, we assume the best possible anchor grid: Namely, that the anchor grid defines an anchor box $\mathcal{B}_a$ of identical scale and aspect ratio. The downsampling factor d^{-1} induces a stride of d between two neighboring anchor boxes on the grid. This means that in the worst case, $\mathcal{B}_{gt}$ and $\mathcal{B}_a$ are displaced against each other by $\frac{d}{2}$ in each direction.

As can be seen in Figure 3.11a, the intersection over union between two such boxes $\mathcal{B}_{gt}$ and $\mathcal{B}_a$ can be calculated by

$$IoU(\mathcal{B}_{gt}, \mathcal{B}_a) = \frac{|\mathcal{B}_{gt} \cap \mathcal{B}_a|}{|\mathcal{B}_{gt} \cup \mathcal{B}_a|} = \frac{(s_{gt} - \frac{d}{2})^2}{2s_{gt}^2 - (s_{gt} - \frac{d}{2})^2} \tag{3.31}$$

In order to get a relationship between stride d and the minimum groundtruth instance size we solve the inequality $t_{ovl} \leq IoU(\mathcal{B}_{gt}, \mathcal{B}_a)$ for s_{gt}. t_{ovl} represents the minimum IoU an anchor has to have with an object in order to classify the object as detected and is commonly set to $t = 0.5$. Solving the quadratic expression above for s_{gt} while assuming $d > 0$ and $0 < t_{ovl} < 1$ and ignoring negative solutions we obtain

$$\frac{d(t_{ovl} + 1) + d\sqrt{2t_{ovl}(t_{ovl} + 1)}}{2 - 2t_{ovl}} \leq s_{gt} \tag{3.32}$$

The VGG16 [72] network has a downsampling factor of $d^{-1} = 16$ for the *conv5* feature map. Using $t_{ovl} = 0.5$ we obtain a minimum detectable object size of $s_{gt} \approx 44px$. For the *conv4* and *conv3* feature map we obtain a minimum detectable object size of $s_{gt} \approx 22px$ and $s_{gt} \approx 11px$, repectively. It should be noted, that these estimates assume an anchor grid that contains anchor boxes of precisely the right size and aspect ratio. These values are therefore a lower bound on what object sizes can reasonably be expected to be detected. In practice, the minimum detectable object size can be much larger and depends on the configuration of the anchor grid.

We will now look at the impact of the anchor grid configuration. More specifically, we will look at the choice of scales. Since we define an anchor box as a positive example for an object if the IoU with any groundtruth instance is greater or equal 0.5, we will assume that we can train a classifier which can learn this concept with perfect accuracy. We ask the question, how far can we space scales apart without running the risk of missing any objects.

To answer this question, we consider the situation in Figure 3.11b. We assume a groundtruth box $\mathcal{B}_{gt}$ of height h_{gt} and width w_{gt} and an anchor box $\mathcal{B}_a$ of height and width h_a and w_a, respectively. Furthermore, we assume w.l.o.g. that $h_{gt} \leq h_a$ and $w_{gt} \leq w_a$. Finally we assume the the side length of the boxes are related through a scale factor $a \geq 1$ by $h_a = \alpha h_{gt}$ and $w_a = \alpha w_{gt}$.

Under these conditions, we can move $\mathcal{B}_{gt}$ anywhere inside of $\mathcal{B}_a$ without changing $IoU(\mathcal{B}_{gt}, \mathcal{B}_a)$. In this case, we can express the IoU as the ratio between the areas enclosed by these boxes.

$$IoU(\mathcal{B}_{gt}, \mathcal{B}_a) = \frac{|\mathcal{B}_{gt} \cap \mathcal{B}_a|}{|\mathcal{B}_{gt} \cup \mathcal{B}_a|} = \frac{w_{gt} h_{gt}}{w_a h_a} = \frac{w_{gt} h_{gt}}{\alpha^2 w_{gt} h_{gt}} = \frac{1}{\alpha^2} \tag{3.33}$$

If we require $t_{ovl} \geq IoU(\mathcal{B}_{gt}, \mathcal{B}_a)$ for a positive classification we find that $\alpha \leq \sqrt{t_{ovl}^{-1}}$. Therefore, in order to not miss any objects, the maximum factor by which the scale of the anchor box and the groundtruth box are allowed to deviate is $\alpha = \sqrt{2}$ for $t_{ovl} = 0.5$. If the scale of a groundtruth item deviates more than a factor of α from the anchor box, it should be picked up by another anchor box on a smaller scale. Therefore, the maximum theoretical amount by which two neighboring anchor scales are allowed to differ is by a factor of α^2. This would support the anchor grid as proposed by [66].

However, this scheme of scales assumes that a classifier will be able to learn the visual difference between an IoU of 0.49 and 0.51 fairly reliably. It seems quite unlikely that in practice a classifier will be able to learn such a hard distinction for two reasons: (1) Training examples with an IoU near 0.5 will look visually similar but may carry conflicting class labels. This tends to result in low-confidence classifications. (2) Object annotations are only given in the form of a bounding box which is only a rough approximation of the object shape.

In the following, we will empirically examine how well an RPN performs in practice as the object size differs from the anchor size. To perform this examination,

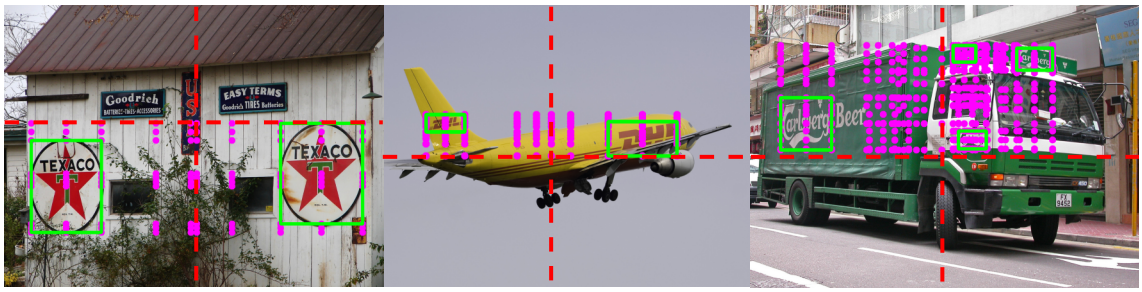


Figure 3.12: Creating the $F_{test,x}$ datasets. The image is recursively split into 2×2 image patches until each patch only contains a single object which is then scaled to the desired size. Image patches need to be small to fit into GPU memory, yet they also need to contain background for meaningful RPN measurements. Split candidates (displayed in pink) are based on the vertices of groundtruth boxes (green). Splits near the image center are preferred.

we use the FlickrLogos dataset to create new datasets with tightly controlled object sizes. In order to be able to scale the objects to a specific size, we first need to make sure that every image in the dataset contains precisely one object instance. In principle, we could easily archive this by directly cutting out the individual objects and scaling them to the desired size. However, since we are interested in evaluating object proposals, we need to include as much background as possible in each image.

At the same time, we need to limit the size of the images. Often, objects instances can be very small compared to the image. Scaling the object to a particular size might increase the size of the image to a point where we are unable to fit it into GPU memory. For this reason, we aim to cut up the individual images into smaller images, each of which roughly covers a fourth of the area. We achieve this by finding points near the image center (which we call *split candidates*) which split the image into four sub-images without intersecting any groundtruth instances. Figure 3.12 illustrates this.

We construct the dataset in the following way: In the simplest case, the image only contains a single object instance. In this case, we can just scale the image so that the $\sqrt{area}$ of the object has the desired size. After the scaling operation, the image might need to be cropped not to exceed the desired dimensions.

In case that an image contains multiple object instances, we find a set $\mathcal{N} = \{n_1, n_2, \ldots, n_m\}$ of pairwise non-overlapping groundtruth boxes. If $\mathcal{N} = \emptyset$ we discard the image. Each groundtruth box n_i is represented by its four vertices $n_i = (\boldsymbol{v}_{i,1}, \boldsymbol{v}_{i,2}, \boldsymbol{v}_{i,3}, \boldsymbol{v}_{i,4})$. We then construct the set $\mathcal{V}_\mathcal{N} = \{\boldsymbol{v}_{1,1}, \ldots, \boldsymbol{v}_{m,4}\}$ which contains the vertices of all the groundtruth boxes in $\mathcal{N}$. From $\mathcal{V}_\mathcal{N}$ we build a set of split candidates $\mathcal{C} = \left\{\frac{\boldsymbol{p}_k + \boldsymbol{p}_l}{2} | (\boldsymbol{p}_k, \boldsymbol{p}_l) \in \mathcal{V}_\mathcal{N} \times \mathcal{V}_\mathcal{N}\right\}$. Each element in $\mathcal{C}$ defines a potential axis-parallel split which splits the image into four sub-images. We try to find a split which is as close as possible to the image center but does not intersect any groundtruth boxes. In case no such split can be found, we discard the image.

This process is repeated recursively on each of the four sub-images until each image contains at most one object instance. Images containing no object instances are discarded. This process is illustrated in Figure 3.12. The result is a dataset in which every image contains exactly one object whose $\sqrt{area}$ has been scaled to a specific size.

We create 11 differently scaled versions of the FlickrLogos-47 test set using this algorithm. In the following, we will refer to these datasets as $F_{test,x}$ where $x \in \{10i + 20 | i = 0 \dots 10\}$ represents the target object size measured in $\sqrt{area}$. Additionally, we use the same algorithm to create a single dataset F_{train} which contains objects scaled in such a way that the object scales are evenly distributed in the interval $[20px, 120px]$.

We choose the anchor scales $\mathcal{S} = \{32px, 45px, 64px, 90px\}$ and train a RPN specifically for each of these scales. For a given scale, we then evaluate the trained RPN on all datasets $F_{test,x}$ while always limiting the max. number of proposals to $n = 2000$. Furthermore, we repeat this process using features from different feature maps $conv_3$, $conv_4$ and $conv_5$.

The results of these experiments are shown in Figure 3.13. Focusing on the results obtained from the $conv_5$ feature map – which is the feature map that is commonly used for RPNs – we notice a general trend of decreasing RPN performance when we go to smaller objects. This effect can be explained through the lack of resolution of the $conv_5$ feature map. Although we do observe a noticeable drop in performance as we move away from the anchor scale, the powers-of-two scheme for selecting anchor scales seems to be sufficiently dense for the $conv_5$ feature map.

We want to examine, whether we can improve RPN performance for smaller objects when using higher-resolution feature maps. As Figure 3.13 shows, this is clearly the case. The gains in performance are quite noticeable for small anchors up to a scale of $64px$. However, we also notice that RPN performance becomes more sensitive to scale when using $conv_3$ or $conv_4$ features. It becomes clear that in this case, the powers-of-two scale selection scheme does not provide an adequate sampling of the scale space. We, therefore, propose to double the number of anchor scales so that two neighboring scales are separated by a factor of $\sqrt{2}$ instead of 2.

An interesting observation is that earlier layers are able to predict small objects just as well as deeper layers can predict large objects. This is remarkable because deeper networks are generally regarded as superior to shallow networks since deeper networks represent more complicated model functions. At least for our task and our network structure, it seems that deeper networks are not necessarily associated with better results.

From our analysis, we, therefore, take note of two design principles for RPN-based small object proposals: (1) We should not rely on $conv_5$ features alone to reliably capture small object instances since the resolution of the associated anchor grid is too coarse. (2) If we use features from earlier feature maps such as $conv_3$ and $conv_4$, we need to increase the density of scale space sampling for anchor boxes.

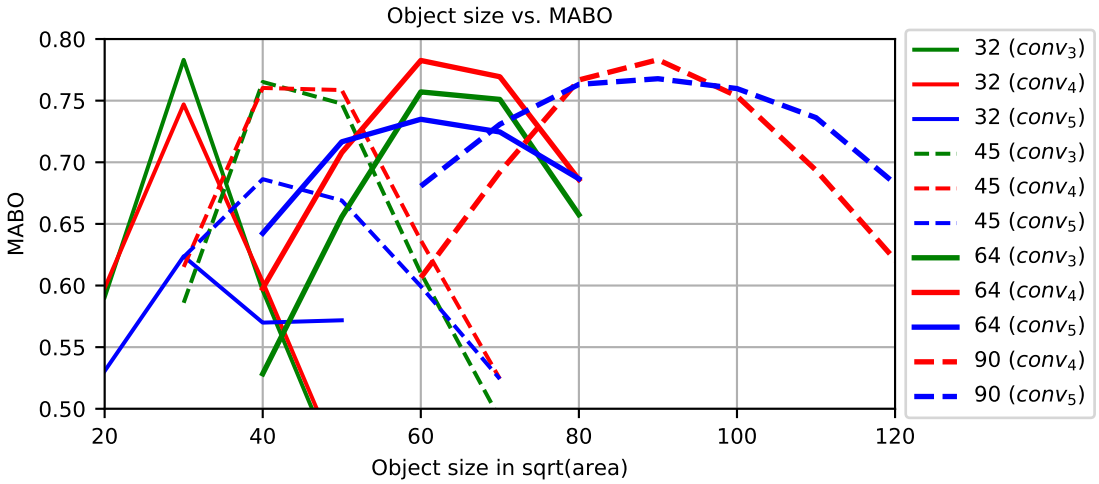


Figure 3.13: Analysis of the scale sensitivity of the RPN anchor grid. Each plot shows the RPN performance as a function of object scale for given anchor box scale and feature map. RPN performance of $conv_5$ features decreases for small objects. Higher-resolution feature maps alleviate the problem, but require a denser scale space sampling.

3.5.4 Improving RPN object proposals

From our previous theoretical and empirical analysis of the anchor grid we have identified two promising approaches to improve RPN performance: (1) Higher-resolution feature maps offer better predictions for small objects than deeper feature maps while deep feature maps are better for predicting large objects. (2) A denser sampling of the scale space is likely to improve performance. However, the performance gains are likely to be more noticeable for higher-resolution feature maps than for deep feature maps.

Denser scale-space sampling

For our initial evaluation of RPNs we used the extended anchor set $\mathcal{S}_{ext} \times \mathcal{A}$ with the aspect ratios $\mathcal{A} = \{0.5, 1.0, 2.0\}$ and $\mathcal{S}_{ext} = \{32px, 64px, 128px, 256px\}$. This extended anchor set better reflected the large variations in scale in the FlickrLogos-47 dataset compared to the anchor set of the original implementation.

Our analysis from Chapter 3.5.3 (see Figure 3.11) revealed that – especially for small object instances – the scale space sampling of the anchor grid is insufficiently dense. We therefore propose a new anchor set $\mathcal{S}_{prop} \times \mathcal{A}$ with $\mathcal{S}_{prop} = \{32px, 45px, 64px, 90px, 128px, 181px, 256px\}$. This new anchor set covers the same range of scales as $\mathcal{S}_{ext}$ but doubles the density of scale space sampling. Each scale is separated by a factor of $\sqrt{2}$ from the next scale.

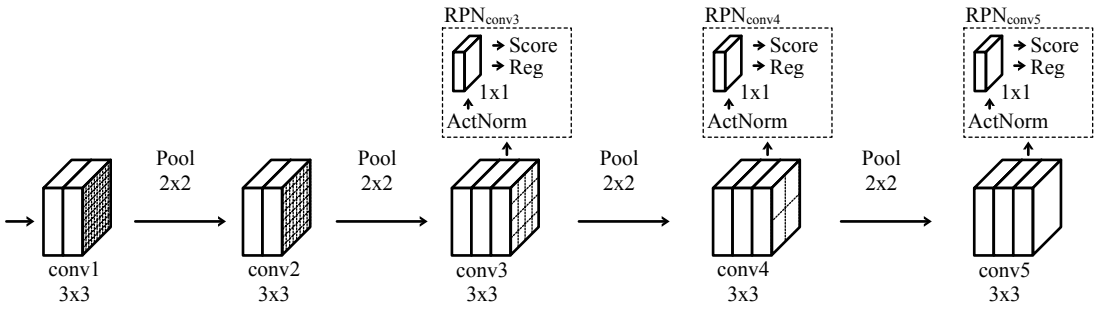


Figure 3.14: Network architecture for improved region proposals. Proposals are generated by multiple RPNs attached to feature maps of different resolutions – each using their own loss functions. Activations need to be normalized between the different feature maps for training.

High-resolution feature maps for small objects

In order to exploit higher resolution feature maps, we propose the following modified RPN architecture which is shown in Figure 3.14: Instead of a single RPN module attached to the last feature map, we attach additional RPN modules to higher-resolution feature maps. Each RPN module is responsible for predicting a different range of anchor scales. In the following, we will refer to these modules by subscripting them with the feature map to which they are attached. E.g., RPN_{conv3} refers to the RPN modules attached to the *conv3* convolution block of the VGG16 network.

Weighting and Normalization Every RPN module requires its own loss functions – one for classification and bounding box regression. This introduces the problem of weighting the influence of each individual loss functions. Since each loss function is responsible for its own range of object scales and all object scales are equally important to predict correctly, it is tempting to assign the same weight to each loss function.

However, assigning equal weights to loss functions does not necessarily mean that each loss function contributes equally to the training progress. In order for loss functions to have the same influence, several conditions must be fulfilled: (1) The loss functions must be of the same type (e.g., Softmax cross-entropy loss). (2) They must operate on the same network output. (3) The same number of training examples must be considered by each loss function.

None of the above-mentioned conditions are met in the case of our network architecture. On each arm, we have a softmax cross-entropy loss $\mathcal{L}_{cls}$ for classification and a Huber-loss $\mathcal{L}_{reg}$ for bounding box regression which violates requirement (1). The different branches of the network violate Requirement (2). Finally, condition (3) is violated because it is almost certain that for any given image, not all object scales are present.

Problem (1) is solved in the following way: We balance the softmax cross-entropy loss $\mathcal{L}_{cls}$ and the Huber-loss $\mathcal{L}_{reg}$ by introducing a factor λ. The total loss is given by $\mathcal{L}_{total} = \mathcal{L}_{cls} + \lambda\mathcal{L}_{reg}$. We follow the recommendation by [66] to set $\lambda = 10$ although [66] can show that the performance is largely unaffected by the actual value of λ for a wide range of scales.

Problem (3) is also easily addressed by normalizing the losses $\mathcal{L}_{cls}$ and $\mathcal{L}_{reg}$ by the number of training examples. Since the default behavior of the loss functions is to normalize by the sum of the loss weights and our loss weights are binary vectors marking the active samples, this requires no intervention on our part.

Problem (2) is more difficult to address. Since all object sizes are equally important, we do not want to change the weighting of the loss functions between different branches. Instead, we normalize the features on which the different branches operate. This introduces an implicit weighting of the gradients from every loss function.

As data passes through a deep convolutional neural network, the variance of the activations tends to decrease as the activations grow sparser. Paired with the fact that activations are typically mean-centered (before applying ReLU) this means, that the energy of earlier feature maps is typically higher than the energy of later feature maps. The normalization, therefore, tends to produce a scaling factor which downscales activations from earlier layers and upscales activations from later layers. During the backward pass, this automatically scales the gradients coming from the loss functions.

This normalization is absolutely crucial for training such a multi-branch network. A failure to introduce normalization can even lead to divergence during training.

Anchor Assignment Another question that needs to be addressed is which anchor scales from $\mathcal{S}_{prop} = \{32px, 45px, 64px, 90px, 128px, 181px, 256px\}$ should be predicted by which RPN module. As we will show in Chapter 4.2.3 the classification performance is maximized when the receptive field of a network matches the object size that it is trying to detect. For object proposals, this means that every anchor should be predicted by a feature map whose receptive field is approximately the same size as the anchor scale.

The receptive fields for the RPN modules shown in Figure 3.14 are $40px$ (*conv3*), $92px$ (*conv4*) and $196px$ (*conv5*). It is immediately apparent that the feature map with the largest receptive field of the network is insufficiently large to accommodate the largest anchor scale. Nevertheless, this is exactly the situation in the original RPN implementation by [66] whose anchor scales $\mathcal{S}_{orig}$ $\{128px, 256px, 512px\}$ contain an even more extreme mismatch between receptive field and anchor scale. We choose the *conv3* feature map to predict the anchor of scale $32px$ and $45px$. The *conv4* feature map is responsible for the anchor scales $45px, 64px$ and $90px$. The *conv5* map is responsible for predicting the entire range of anchor scales $\mathcal{S}_{prop}$.

As it can be seen from the assignment, there is some overlap between the layers' responsibilities: In particular, the last layer is responsible for predicting all anchor boxes. This choice makes our multi-featuremap RPN a true extension to the orig-

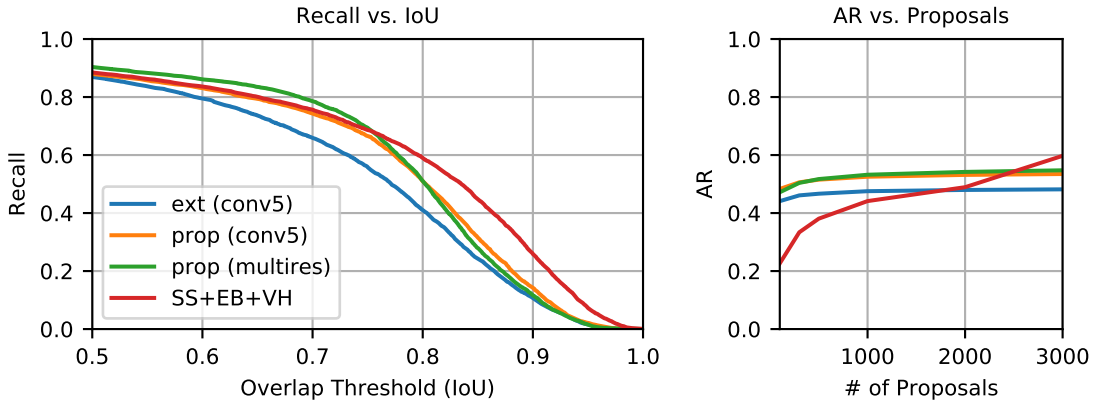


Figure 3.15: Impact of the proposed improvements to the RPN. Our proposed anchor set (*prop*) with denser scale space sampling improves both localization and max. recall compared to the default powers-of-two anchor set (*ext*). Utilizing additional higher-resolution feature maps (*multires*) can improve the max. recall even further.

inal approach. If we were to remove the additional RPN modules, the network architecture is identical to the original approach by [66].

Combining Predictions Since all RPN modules generate independent predictions, the results of all RPN modules need to be combined into a single list of object proposals. This is particularly true because some anchor boxes are predicted by multiple layers.

We apply a bounding box pruning – as described in Chapter 3.5.1 – separately for each RPN module: For each RPN module, we limit ourselves to 8000 pre-NMS proposals. We perform non-maximum suppression with a NMS threshold $t_{nms} \geq 0.7$ and perform a final limit of 2000 proposals per RPN module.

Finally, we concatenate the proposals generated by every RPN module into a single list and run another non-maximum suppression. Our final proposal is the NMS output, limited to 2000 proposals. Because the *conv5* feature map still predicts all anchor boxes using the same pruning process as in the original approach, all performance gains must stem from proposals generated from the other feature maps.

Evaluating the improved RPN

To evaluate our RPN improvements, we first test our proposed anchor set $\mathcal{S}_{prop}$ using the original network architecture with a single RPN module attached to the *conv5* feature map. Figure 3.15 shows the results of this evaluation. Clearly, the proposed anchor set improves both the maximum recall and the localization abilities of the RPN.

To test the effects of incorporating higher-resolution feature maps for predicting small objects we use our proposed anchor set in conjunction with the network

F. map	Scales	$n = 300$ MR	$n = 300$ MABO	$n = 1000$ MR	$n = 1000$ MABO	$n = 2000$ MR	$n = 2000$ MABO	Time / img
conv5	$\mathcal{S}_{ext}$	0.822	0.672	0.856	0.698	0.869	0.710	$0.09s$
conv5	$\mathcal{S}_{prop}$	0.826	0.690	0.864	0.721	0.880	0.733	$0.09s$
multires	$\mathcal{S}_{prop}$	**0.844**	**0.695**	**0.889**	**0.728**	**0.903**	**0.741**	$0.30s$

Table 3.2: Performance of proposed improvements for RPNs for different number of proposals. MR (max. recall) refers to the recall at $IoU = 0.5$. Our proposed anchor set does offer improved performance compared to the *powers-of-two* anchor scales $\mathcal{S}_{ext}$ in all cases. Performance can be enhanced even further by incorporating additional feature maps with higher resolution (*multires*).

architecture discussed in chapter 3.5.4 (*multires*). The benefits are not as obvious as for the $\mathcal{S}_{prop}$ anchor set. While the maximum recall is clearly better than all other methods (including our best heuristic proposals), the same cannot be said for the localization properties. Still, the *multires* approach outperforms the *conv5* approach in terms of average recall (AR) and mean average best overlap (MABO), shown in Table 3.2. In terms of MABO, we are also able to outperform Selective Search and Edge Boxes individually (see Table 3.1 in Chapter 3.4.3). However, combining proposals from Selective Search and Edge Boxes still yields superior MABO performance.

The average runtime per image for generating proposals is also obviously superior to all previously discussed heuristic approaches. However, it is important to keep in mind, that this improved runtime is not due to the lower computational complexity of RPNs but due to the fact that the feature extraction is being run a GPU. In fact, the runtime is mainly dominated by the non-maximum suppression (running on a CPU) which is evident when comparing the runtimes of the *conv5* with the *multires* approach.

3.5.5 Conclusion

We have evaluated two heuristic object proposal algorithms, Selective Search and Edge Boxes, on the FlickrLogos-47 dataset. Without any of our modifications, Selective Search offers high maximum recall while Edge Boxes offers better localization. We have identified common failure modes of Selective Search and suggested that diversifying our source of object proposals might be beneficial. In particular, we have suggested that Edge Boxes might be able to complement Selective Search. We have validated our conjecture through experiment and have shown that by employing both Selective Search and Edge Boxes we can achieve better maximum recall and localization than both algorithms individually.

We have discussed potential deficiencies of both algorithms for generating proposals for text-based company logos. In order to address these problems, we have

introduced the VH-connect algorithm which provides a fast and simple heuristic to improve the recognition of text-based logos. By incorporating VH-connect to our combined Selective Search and Edge Boxes proposals, we were able to achieve even better maximum recall and localization.

Because of the prohibitively long runtimes of heuristic object proposals we have evaluated Region Proposal Networks (RPNs) – an approach to object proposals that can be directly integrated into a DCNN. While being two orders of magnitude faster than heuristic proposals, we found the performance of RPNs to be lacking. We have investigated the reasons for this poor performance and have identified two problems: (1) Insufficiently dense scale space sampling and (2) lacking anchor grid resolution for small objects. We have addressed these problems by proposing an improved set of anchor boxes and by utilizing additional higher-resolution feature maps for the anchor prediction. We were able to show that each of these improvements can boost both the maximum recall and the localization of the generated proposals.

Comparing our improved heuristics with our improved RPN-based proposals, we can make this final evaluation: While we were unable to improve the localization properties of RPN-based proposals to a point where they can beat our best heuristic object proposals, we were at least able to achieve competitive localization performance. Regarding maximum recall we were able to improve RPN-based object proposals so that they actually surpass our best heuristics. Combined with the fact that even our most elaborate RPN-based approach is still an order of magnitude faster than our most elaborate heuristic, we feel justified in recommending RPN-based object proposals over heuristic approaches – even for small object instances as they typically appear in company logo detection.

Chapter 4

Classification Stage

The first deep convolutional networks (DCNNs) like AlexNet [49] and VGG16 [72] were designed for the task of image classification. Image classification seeks to assign a class label to a given image. Usually, this class label refers to the presence of an object in the image.

Object detection goes a step beyond image classification: Not only are we interested in the presence/absence of an object in the image, we also want to know the location of the object instances which may occur multiple times in the same image.

However, the task of image classification and object detection are closely related. Given a classifier that performs image classification, we can apply this classifier to any patch of our image to predict its content. This is the role of the *classification stage* in a two-stage object detection pipeline: It takes image patches proposed by the *object proposal stage* and predicts class labels for each patch. The classification stage provides the class label, and the proposal stage provides the (approximate) location.

In older object detection pipelines [24] the classification was performed over a dense grid of image regions of different scale and aspect ratios. Because the first DCNN-based classifiers were extremely compute-intensive, this dense evaluation was replaced with a sparse evaluation based on object proposals.

Aside from utilizing computational resources more efficiently, the proposal stage can also potentially help the classifier: A classifier which tends to produce many false positives can be assisted by the proposal stage which already excludes many image patches from being processed by the classification stage.

4.1 R-CNN

4.1.1 The R-CNN detection pipeline

R-CNN [31] was amoung the first approaches that applied DCNNs to the task of object detection. In the original implementation [31], R-CNN used AlexNet [49] as classifier.

In order to be able to use a standard classification network such as AlexNet, Girshick et al. had to overcome two problems: (1) Since AlexNet was originally designed for the ImageNet classification challenge [15], it contains 1000 output neurons – one for each class of the ImageNet dataset. The set of ImageNet classes may not completely overlap with the set of classes in the object detection task. One prominent example is that ImageNet does not contain a *person* class while most popular object detection challenges like PASCAL VOC [22] or MSCOCO [55] do. (2) The AlexNet architecture uses fully-connected layers which require fixed-size image patches as input. Since for an object detection task the objects are allowed to have arbitrary size, this mismatch needs to be handled.

R-CNN solves the two above-mentioned problems in the following ways: The last layer of the network which produces the classification is removed from the network. Instead the output of the previous layer is interpreted as a feature vectors which is used as input for a series of Support Vector Machines (SVMs) [78] [13]. Each SVM is responsible for identifying a single class, i.e., solves a binary classification problem of distinguishing one class against the background and every other class. The DCNN is therefore not directly used for classification but for feature extraction.

Additionally, R-CNN performs a bounding box regression to refine the location of each object proposal to better match the object's location. The method used by R-CNN is inspired by the method used in deformable part model-based detectors [24]: For every class c a simple linear regression model $\boldsymbol{L}_c$ is learned through least-squares fitting which predicts four deformation terms $\boldsymbol{t}$ based on the feature vector $\boldsymbol{x}$. The deformation terms predict an offset in x- and y-direction, a scaling factor for width and height and are parametrized in the same way as the regression targets for RPNs (as discussed in section 3.5.1).

$$\boldsymbol{t} = (t_x, t_y, t_{s_w}, t_{s_h})^T = \boldsymbol{L}_c \boldsymbol{x} \tag{4.1}$$

It is important to note that the DCNN used for feature extraction does not necessarily need to be trained to recognize the specific classes of our FlickrLogos-47 dataset. We have observed that a DCNN that has been pre-trained on ImageNet will extract features which generalize well to our task. This observation is also in line with the observation of Razavia et al. [64] who was among the first to notice the power of DCNNs to extract features which generalize well over a wide range of datasets and computer vision tasks. However, detection performance is improved if the pre-trained network is fine-tuned on the specific classification task.

The mismatch between the size of the object proposals and the input size of the network is resolved by simply warping the image patch corresponding to the object proposal to the input size of the network. This may cause significant distortions for object classes with an extreme aspect ratio. However, this is not necessarily a problem because all object instances of the same class will have similar aspect ratios and therefore undergo the same warp.

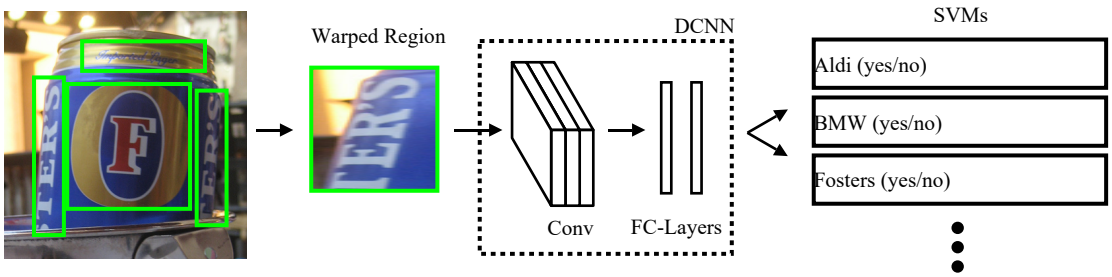


Figure 4.1: Outline of the R-CNN detection pipeline. An object proposal algorithm identifies regions of interest. These image patches are resized to the input dimensions of a DCNN which extracts features. These features are classified by a series of SVMs for the final prediction.

The original implementation of R-CNN [31] used selective search [77] as source for object proposals. An overview of the R-CNN pipeline is depicted in figure 4.1.

4.1.2 Implementation Details

Our implementation of the R-CNN pipeline differs from the original implementation [31] in a few key areas:

Data augmentation Since our dataset contain many small object instances, most object proposals will get upscaled when the image patch is scaled to match the network input dimensions. To create a diverse training set for small objects, we perform data augmentation by creating small resized copies of images containing only large objects. We use this augmented dataset along the original data for fine-tuning the network to our specific dataset. We only focus on images containing large objects because if we downsample images containing small objects we might create object instances that do not contain enough visual information for a reliable classification anymore. This might confuse the classifier instead of enriching the dataset. More specifically, we define a minimum object sidelength of $s_{min} = 20px$ and only resize images for which the following relation holds true:

$$s_{min} < \min_{g \in \mathcal{G}} \left[\min \left(\text{height}(g), \text{width}(g) \right) \right] \tag{4.2}$$

where $\mathcal{G}$ represents the set of all groundtruth instances for the image. If an image fulfills the above-mentioned condition, it is resized by a factor of $\alpha = 0.5$.

Object Proposals Unlike the original R-CNN implementation, we do not solely rely on selective search [77] to generate object proposals. Instead, we make use of our findings from chapter 3.4.4 and use an ensemble of boxes generated by Selective Search, Edge Boxes and our VH-connect extension. We generate up to 2000 object

proposals per image using up to 1400 Selective Search proposals, 400 Edge Boxes proposals, and 200 VH-connect proposals. This mixture archived the best proposal performance as discussed in section 3.4.4. Extracting object proposals in such fashion results in approximately 2.9×10^6 proposals on the augmented trainval set and another 5.9×10^6 proposals on the no-logo training set.

Furthermore, we do not implement the bounding box regression of the original approach to refine detections.

Training Procedure We train out R-CNN implementation in two stages: In the first stage, we fine-tune the network and then train the Support Vector Machines (SVMs) in the second stage.

For fine-tuning, we use all positive examples – defined as proposals with an $IoU \geq 0.7$ – and sample negative (background) examples. The ratio of positive to negative examples is $1 : 5$. In this case, negative examples are defined as proposals with $0.1 \leq IoU \leq 0.3$ resulting in approximately 2.8×10^4 of positive and 5.6×10^3 number of total samples.

The reason for this rather uncommon definition for negative samples is because we do not employ hard-negative mining at this stage, yet we still want the network to focus on classifying hard examples. Since hard negative examples are often proposals which do overlap with a groundtruth object but not sufficiently to be classified as a positive example, this heuristic helps to focus the network on the relevant examples.

We run the fine-tuning for 20000 iterations using a batch size of 256 (which corresponds to approximately 150 epochs). The initial learning rate of $\lambda = 0.001$ is reduced by a factor of 10 after 10000 iterations.

After fine-tuning the network, we train the SVMs on the features extracted from the network. We train one SVM per logo class $c \in \mathcal{C}$ to solve the binary classification problem of distinguishing one specific class from all other logo classes and the background. In this formulation of the classification problem, each proposal requires the evaluation of $|\mathcal{C}|$ SVMs, each of which outputs a confidence score of the proposal belonging to a specific class. The confidence of the classification is given by the SVM that yields the highest score, and the proposal is assigned to the class belonging to the highest-scoring SVM.

Since all SVMs are trained separately from each other, we cannot guarantee that the confidence scores between SVMs are comparable. To mitigate the problem, we L_2-normalize the features before using them in the SVM, which works well in practice.

For our evaluation, we exclusively use SVMs with linear kernels. We have experimented with more complex kernel functions such as the RBF kernel, but we have found the results to be virtually identical with linear kernel functions. It is therefore hard to justify expending the additional computational complexity of such kernels.

SVM training happens in three learning rounds, the initial training and two rounds of hard negative mining. Initially, we use all proposals with an $IoU \geq 0.7$

			VGG16				CaffeNet			
Proposals			Features			Time	Features			Time
SS	EB	VH	pool5	fc6	fc7	/ img	pool5	fc6	fc7	/ img
2000	0	0	**0.718**	0.713	0.712	15.7s	0.715	**0.716**	0.702	5.3s
1400	600	0	**0.737**	0.725	0.721	16.6s	0.723	**0.729**	0.715	6.3s
1400	400	200	**0.747**	0.740	0.738	16.6s	0.737	**0.742**	0.730	6.3s

Table 4.1: Evaluation of our R-CNN implementation on FlickrLogos-47. Detection performance is given as mAP for different feature maps. The performance differences between the different feature maps are mostly marginal. Improved object proposals also translate to better detection performance. However, the source of object proposals can have a significant impact on the total runtime.

with an instance of the current class as positive examples. Negative examples for the initial training are sampled in a ratio of 1 : 1 from all proposals with $0.1 \geq IoU \geq 0.3$.

For hard negative mining, we limit ourselves to the $n = 4$ top detections per image per round. To ensure diverse examples, we additionally require that these detections do not have an $IoU \geq 0.2$ with each other. We first evaluate the SVMs on the *trainval* set of the FlickrLogos-47 dataset. Object proposals are added as hard negatives for class c in two cases: (1) If the predicted class does not agree with the groundtruth class of the proposal. (2) If the predicted class does agree with the groundtruth class of the proposal, but the $IoU \leq 0.3$.

For hard negative mining, we do not distinguish between *symbol* and *text* subclasses: If, for example, the predicted class is *Adidas (Symbol)* and the groundtruth class is *Adidas (Text)* criterium (1) would not be sufficient to categorize the proposal as hard negative.

After classifying the *trainval* set, we continue mining for hard negatives on the *logo-free* image set. On this set, we add the $n = 4$ top detections per image that do not overlap by more than 0.2 with each other to the hard negative set of the predicted class.

4.1.3 Evaluation

We perform the evaluation of our R-CNN implementation using features from different feature maps and object proposals. We evaluate features from the *pool5*, *fc6* and *fc7* map for the VGG16 network and for CaffeNet (which is a slightly modified version of AlexNet [49]).

Table 4.1 shows the result of this evaluation. The performance of the different feature maps is almost identical. However, the *pool5* feature map consistently scores slightly better than all other features for VGG16. For CaffeNet, the *fc6* features consistently score best. The dimensionality d of a *pool5* feature vector is much higher than the dimensionality of a *fc6* or *fc7* feature ($d_{pool5} = 25088$ vs. $d_{fc6} =$

$d_{fc7} = 4096$). Although a high-dimensional feature vector translates to a slower SVM evaluation, the added computational cost is negligible since we use a linear SVM which can be evaluated very quickly. Most of the runtime is spent during feature extraction.

It is important to note that the timing results in Table 4.1 are not directly measured but calculated runtimes which are composed of the average runtime for proposal generation, the average runtime for feature extraction and the average runtime for classification. The runtime per image is extremely high when we use the VGG16 network for feature extraction. It is possible to use a much simpler network like AlexNet [49] which achieves almost identical performance as VGG16 but with sharply reduced runtime (approx. $5s$ / img). In any case, the time required for feature extraction dominates the total runtime of this approach. Although the detection performance of VGG16 is of little interest under these circumstances, we still choose to include them because we will use the VGG16 architecture for the detection pipelines in Chapters 4.2 and 4.3. These measurements allow comparing the different approaches directly.

In Appendix D we show example detections.

4.2 Fast R-CNN

4.2.1 Network architecture and Detection pipeline

While R-CNN is able to produce impressive results in terms of detection performance, its long processing time makes it unattractive for most real-world application. The two main factors for R-CNNs long processing time are the number of proposals to evaluate and the network structure itself.

While both factors can certainly be optimized, the architecture of the R-CNN pipeline itself can often limit the efficiency of the approach as the following example illustrates: Let us assume 2000 proposals per image and a VGG16 [72] network as feature extractor. Since the VGG16 network has an input size of $224px \times 224px$, every proposal – no matter how small and how much they overlap with each other – needs to be resized to these dimensions. This configuration requires the network to approximately process the equivalent of a single $10000px \times 10000px$ image. As most images in our dataset only have a size of $1024px \times 1024px$ it becomes evident that much redundant computation must be involved in this approach.

Fast R-CNN [30] improves on R-CNN in two ways: (1) It eliminates most of this redundant computation by extracting features only once for the entire image. (2) It removes the SVMs as classifiers and uses the network itself to generate the predictions. This allows for joint optimization of the classifier and the feature extractor and simultaneously speeds up evaluation.

Fast R-CNN exploits the fact that the convolution operation is not intrinsically tied to its input or output size: As the size of the input increases, so does the size of

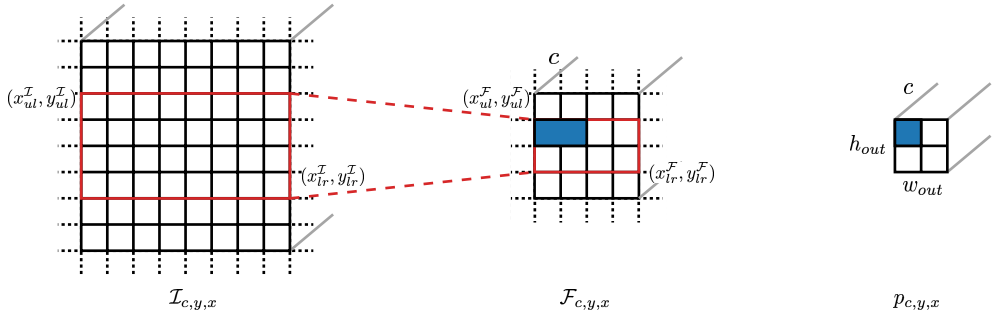


Figure 4.2: Illustration of ROI-Pooling: Input image $\mathcal{I}$ and feature map $\mathcal{F}$ are related through a downsampling factor α. An object proposal (red) is projected onto a corresponding region on the feature map. The feature map entries within this region are pooled into a fixed-size feature. The ROI-pooled features $\boldsymbol{p}$ have user-specified spatial dimensions w_{out} and h_{out} and share the same number of channels as $\mathcal{F}$. Each bin of $\boldsymbol{p}$ originates from a max-pooling operation with a variable-sized kernel (blue).

the output. Therefore, convolutional layers of the network can be resized as desired, according to the size of the input image.

ROI-Pooling Fast R-CNN uses the fully-connected layers of the network as classifiers. Since the networks are typically pre-trained on a classification task such as ImageNet [15] which has 1000 classes, the last fully-connected layer of the network is replaced with a layer that has the desired number of output neurons for the detection task. Aside from the number of output neurons, fully-connected layers are also tied to the number of inputs. Unlike the number of output neurons, the input size is not known in advance, since the input image can vary in size.

In order to solve this problem, Fast R-CNN introduces a so-called *ROI-Pooling Layer* which maps arbitrarily-sized regions from the feature map to a fixed-size representation $\boldsymbol{p}$. Similarly to feature maps, $\boldsymbol{p}$ can be regarded as a 3-dimensional array which has a certain number of channels and a user-specified spatial resolution. We therefore index the individual elements $p_{c,y,x}$ of $\boldsymbol{p}$ in the same way as feature maps: c refers to the channel index, while y and x describes a spatial location in $\boldsymbol{p}$. In the following, we will refer to a single spatial location (x, y) as a *bin*.

On an abstract level, ROI-Pooling works similar to a max-pooling operation: Each output bin of an ROI-Pooling operation is obtained by computing the channel-wise maximum over a given input region. Unlike traditional max-pooling where the size of the input region (the kernel size) is fixed, the output size is fixed in ROI-Pooling. The spatial resolution of the output is determined by user-specified parameters w_{out} and h_{out}, which refer to the number of bins in x and y direction, respectively. The number of output channels is determined by the number of channels in the input feature map. Since the spatial output dimensions are fixed but the

input is allowed to have arbitrary size, ROI-Pooling requires a variable kernel size which depends on the size of the input region and the number of output bins.

More formally, an ROI-Pooling layer takes four inputs: (1) The output of the last convolutional layer $\mathcal{F}$. (2) A list of n object proposals. Each proposal is characterized by its upper-left and lower-right vertex $\boldsymbol{x}_{ul}^{\mathcal{I}} = (x_{ul}^{\mathcal{I}}, y_{ul}^{\mathcal{I}})^T$ and $\boldsymbol{x}_{lr}^{\mathcal{I}} = (x_{lr}^{\mathcal{I}}, y_{lr}^{\mathcal{I}})^T$ The proposal coordinates are specified in image coordinates which we indicate by the superscript $\mathcal{I}$. (3) The downsampling factor α between the image and the feature map $\mathcal{F}$. (4) The desired spatial size of the output feature representation (h_{out}, w_{out}).

First the coordinates of the proposals are mapped onto the corresponding feature map coordinates (indicated by the superscript $\mathcal{F}$) using the downsampling factor α.

$$\boldsymbol{x}_{ul}^{\mathcal{F}} = (x_{ul}^{\mathcal{F}}, y_{ul}^{\mathcal{F}})^T = \alpha^{-1}\boldsymbol{x}_{ul}^{\mathcal{I}} \qquad \boldsymbol{x}_{lr}^{\mathcal{F}} = (x_{lr}^{\mathcal{F}}, y_{lr}^{\mathcal{F}})^T = \alpha^{-1}\boldsymbol{x}_{lr}^{\mathcal{I}} \tag{4.3}$$

The ROI-Pooling layer performs a max-pooling operation for every output bin

$$p_{c,y,x} = max\_pool_{\mathcal{F}_c} \begin{pmatrix} \left\lfloor x_{ul}^{\mathcal{F}} + \frac{x(x_{lr}^{\mathcal{F}} - x_{ul}^{\mathcal{F}})}{w_{out}} \right\rceil \\ \left\lfloor y_{ul}^{\mathcal{F}} + \frac{y(y_{lr}^{\mathcal{F}} - y_{ul}^{\mathcal{F}})}{h_{out}} \right\rceil \\ \left\lfloor x_{ul}^{\mathcal{F}} + \frac{(x+1)(x_{lr}^{\mathcal{F}} - x_{ul}^{\mathcal{F}})}{w_{out}} \right\rceil \\ \left\lfloor y_{ul}^{\mathcal{F}} + \frac{(y+1)(y_{lr}^{\mathcal{F}} - y_{ul}^{\mathcal{F}})}{h_{out}} \right\rceil \end{pmatrix} \tag{4.4}$$

where $max\_pool_{\mathcal{F}_c}(\boldsymbol{f})$ computes the maximum value in an area $\boldsymbol{f}$ on channel c of the feature map $\mathcal{F}$. The area $\boldsymbol{f}$ is specified by a rectangle defined by the upper left and lower right vertices in feature map coordinates. $\lfloor . \rceil$ denotes the operation which rounds to the nearest integer. This process is illustrated in Figure 4.2.

Bounding Box Regression Aside from classifying the image content of an object proposal, Fast R-CNN also performs a bounding box regression to refine the object's location further. In contrast to the R-CNN pipeline, Fast R-CNN integrates bounding box regression directly into the network. This is done by adding a separate fully-connected layer which operates on the same features as the classification layer.

The bounding box regression in Fast R-CNN is very similar to the regression used by RPNs (discussed in Chapter 3.5.1). Most notably, the parametrization of the regression targets is identical and the Huber loss [41] is used during training. A crucial difference to the regression mechanism of RPNs is that Fast R-CNN performs a class-specific bounding box regression: If there are C different (positive) classes, the fully-connected layer for the bounding box regression needs to have $4C$ output neurons.

During training, the regression target for a proposal is only applied to the 4 output neurons associated with the correct class. This is achieved by introducing binary weights for the Huber loss: The loss weights for all outputs are set to zero

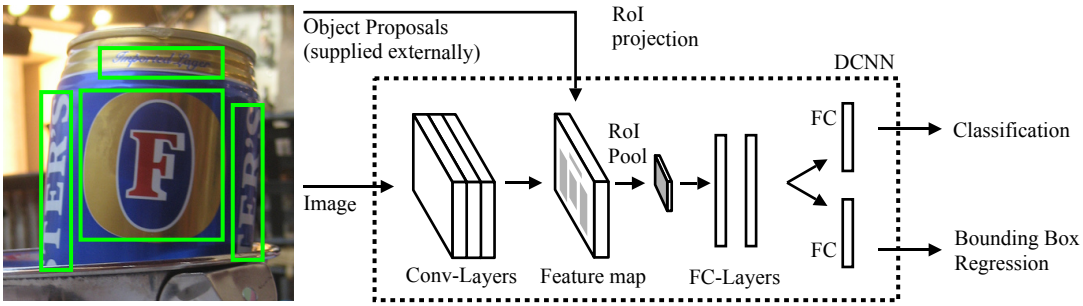


Figure 4.3: Principle of the Fast R-CNN detection pipeline. Feature extraction is performed only once for the whole image through the convolutional layers. Classification and bounding box regression are integrated into the network through fully-connected layers. An ROI-Pooling layer maps variably-sized regions on the feature map to fixed-size features for the fully-connected layers.

except for these 4 outputs. The loss is always normalized by the number of active outputs. Should a batch contain no active outputs for bounding box regression, the loss is set to zero.

Summary Figure 4.3 provides an overview of the Fast R-CNN pipeline. We use the VGG16 network as an example to show, how a DCNN used for ImageNet classification can be turned into a Fast R-CNN compatible object detection network: (1) Remove the last fully-connected layer of the pre-trained network. (2) Add two new layers at the end which are randomly initialized using Gaussian noise: One for classification and one for bounding box regression. If there are C different (positive) classes, the classification layer requires $C + 1$ output neurons (C positive classes and the background class), and the regression layer requires $4C$ output neurons (4 regression targets per positive class). (3) Replace the last max-pooling layer with an ROI-Pooling layer. It is crucial to choose (h_{out}, w_{out}) of the ROI-Pooling layer to match the number of output neurons of the original max-pooling layer. In the case of the VGG16 network, this is equal to a 7×7.

Aside from the improvements in runtime, Fast R-CNN offers another benefit. Since classification, bounding box regression are all integrated into the network, Fast R-CNN can be trained end-to-end.

4.2.2 Fast R-CNN on FlickrLogos-47

We perform our evaluation of Fast R-CNN on FlickrLogos-47 in two parts: First, we evaluate how different numbers of object proposals affect the performance of the classification stage. Second, we want to ascertain, whether our improved object proposals from Chapter 3.4 translate to better detections. For direct comparison,

				VGG16		CaffeNet	
	Proposals			Performance	Runtime	Performance	Runtime
Total	SS	EB	VH	(mAP)	/ img	(mAP)	/ img
300	300	0	0	0.589	0.97s	0.557	0.92s
1000	1000	0	0	0.637	1.00s	0.598	0.94s
2000	2000	0	0	0.643	1.04s	0.606	0.94s
2000	1400	600	0	0.658	2.02s	**0.624**	1.93s
2000	1400	400	200	**0.661**	2.02s	0.623	1.94s

Table 4.2: Evaluation of our Fast R-CNN implementation on FlickrLogos-47. Detection performance is given as mAP for different feature maps. Compared to the R-CNN pipeline, the detection performance is noticably lower. The overall runtime could be reduced significantly and is mostly dominated by the object proposal stage.

we perform this evaluation with the same composition of object proposals which we used to evaluate R-CNN in Chapter 4.1.3.

Training Procedure We use our batch-normalized VGG16 network which has been pre-trained on ImageNet as a basis for our Fast R-CNN implementation. As described in Chapter 4.2.1, the *pool5* layer is replaced by a ROI-Pooling layer and the *fc8* layer is replaced by two fully-connected layers which are both attached to the *fc7* layer: One layer with 48 output neurons (47 logo classes + 1 background class) for classification. The other layer with 188 output neurons for bounding box regression (4 regression targets for each of the 47 logo classes). The training consists of fine-tuning the VGG16 network for 60000 iterations. We use an initial learning rate of $\lambda = 0.001$ with a learning rate reduction by a factor of 10 after 40000 iterations. In every iteration, we draw two images from the dataset at random. For each of these images, we perform data augmentation by randomly flipping the image along the vertical axis. Each of these images is associated with a set of object proposals from which we sample a batch of 256 proposals in total – 64 positive and 192 negative examples. We consider a proposal to be a positive example if it has an $IoU \geq 0.5$ with an object instance.

Training examples are partially drawn at random. However, we also employ both hard positive and hard negative mining (often called *hard example mining*) for the classification task. Hard positives are those positive examples which have been classified with the lowest confidence. From the 64 positive examples in each batch, 16 examples (25%) are hard positives. Hard negatives are those negative examples which have been classified as an object instance with the highest confidence. From the 192 negative examples in each batch, 48 examples (25%) are hard negatives. Since there are no positive or negative examples for a regression task, there is no

hard example mining for the bounding box regression. We always use all proposals which have an $IoU \geq 0.5$ with an object instance for training.

Evaluation We first look at the influence of the number of proposals on the classification performance. In our first experiment, we train Fast R-CNN using 2000 proposals and evaluate its performance using 300, 1000 and 2000 proposals. The results of this evaluation is shown in Table 4.2. It is evident that more object proposals tend to improve detection performance. We have limited ourselves to a maximum number of 2000 proposals because we have established during our evaluation of heuristic object proposals (see Chapter 3.4) that increasing the number of proposals beyond 2000 hardly translates into increased recall anymore. More proposals, therefore, are unlikely to be hugely beneficial to the classifier. This is also evident in the difference in performance between 2000 and 1000 proposals, which is not nearly as large as the difference between 1000 and 300 proposals. Too many proposals can also be detrimental to the classifier's performance: Since the classifier has a certain false positive rate, more proposals can lead to more false positives. Because the recall has not much room to improve, the performance is bound to decline.

It also becomes clear, that Fast R-CNN is able to reduce the runtime compared to R-CNN significantly. For R-CNN most of the time is spent on feature extraction. For Fast R-CNN the runtime is mostly dominated by the object proposal generation. Given the proposals ($n = 2000$), a single image takes only $0.14s$ to process for VGG16 and $0.05s$ for CaffeNet.

For R-CNN, the classification performance of CaffeNet was almost on par with the performance of VGG16. The dramatically lower runtime might convince some users to use R-CNN in conjunction with CaffeNet instead of VGG16. For Fast R-CNN the results are different: The difference in runtime between VGG16 and CaffeNet is mostly insignificant while the difference in detection performance is not negligible. However, both for CaffeNet and for VGG16 the detection performance is noticeably lower with Fast R-CNN as when using R-CNN.

This last observation requires a more in-depth investigation: Conceptually, the differences between R-CNN and Fast R-CNN seem minor, yet the difference in detection performance is quite severe. (1) Both approaches use the same network architecture for feature extraction. (2) R-CNN uses an SVM for classification while Fast R-CNN integrates the classification into the network. While it is reasonable to expect a difference in performance when exchanging the classifier, it seems unlikely that this can fully account for the magnitude of the change. (3) The ROI-Pooling layer produces the same output as a max-pooling layer if we prepare the image patches in the same way as for R-CNN.

Therefore, it seems that the fact that R-CNN resizes the input patches has a beneficial effect on the detection performance. However, this conclusion seems strange for two reasons: (1) The original approach [30] reports an improvement in both runtime and detection performance compared to R-CNN for the task of

general object detection. (2) Since most of the logo instances on our dataset are small, R-CNN on average needs to upsample the proposals to fit the network input dimensions. However, upscaling does not add new information to the image.

It seems strange that resizing the inputs to the network should be the cause for the performance difference. In the following, we will analyze our suspicion in greater detail and will answer the question of why Girshick [30] was able to report an improvement in detection performance.

4.2.3 Receptive field and object size

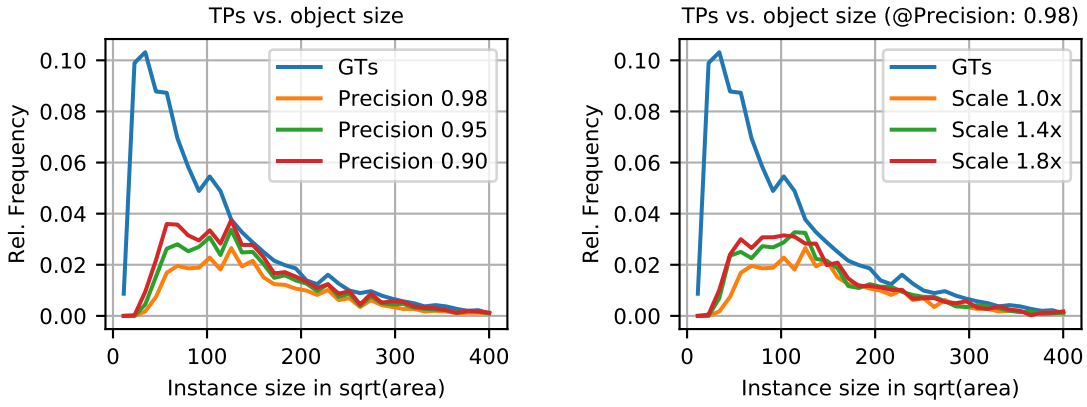
To analyze the performance of Fast R-CNN on FlickrLogos-47 more deeply, we plot the size distribution of correctly detected objects in Figure 4.4. Figure 4.4 makes it obvious that large object instances are detected almost perfectly, while only a relatively small fraction of small objects are detected. This discrepancy can be made smaller at the cost of lower precision which indicates that small object instances tend to be detected with lower confidence than large objects.

Object size vs. Detection Performance In order to test our suspicion that object size plays an important role, we perform the following experiment: We upscale all images by a constant factor and re-run training and evaluation on these upscaled images. We choose a specific level of precision (in this case 0.98) and plot the size distribution of detected objects in Figure 4.4. It becomes clear that by upscaling images by a factor of 1.8 we can detect roughly the same number of objects at a precision of 0.98 as by processing the unscaled images at a precision of 0.90.

For comparison with our results in Table 4.2 in Chapter 4.2.2, we are able to achieve a mAP of 0.692 and 0.710 using a scale factor of 1.4 and 1.8, respectively (using $SS+EB+VH$ proposals). Because of limited GPU memory, we are unable to upscale images beyond a scale factor of 1.8 with our current network architecture. However, the aforementioned results do suggest that by upscaling the images we can build a Fast R-CNN pipeline which at least rivals the detection performance of R-CNN. Since upscaling does not add new information to the images, these experiments suggest, that the size of the object itself indeed has an influence on detection performance.

We can also observe from Figure 4.4 that only small object instances benefit from this upscaling. Object instances larger than approx. $120px$ do not tend to benefit. Upscaled by a factor of 1.8, this means that these objects have a size of approx. $216px$ in the upscaled image. As mentioned in Chapter 2.2.1, the receptive field of the VGG16 network is $212px$ wide (at the *pool5* layer). This observation provides a first hint to understanding this phenomenon: Maybe it is the case that objects are detected best when they have the same size as the receptive field of the network.

In order to confirm this observation and to test whether we can potentially exploit this discovery by reducing the receptive field of the network, we follow up



(a) (b)

Figure 4.4: Analysis of correctly detected objects using Fast R-CNN by size. (a) Especially for small object instances there exists a large discrepancy between detected objects and total objects. This gap can be closed at the cost of lower precision. (b) For a given level of precision, another way of partially closing this gap is by upscaling the images. Graphs were generated from histograms with a bin width of $40px$. For all graphs, the instance size is given with respect to the original image (Scale 1.0).

by performing another experiment: We step back from the complexities of object detection and consider a simple classification problem.

The importance of the receptive field In order to test the influence of the receptive field on image classification, we create multiple versions of the VGG16 classification network (introduced in section 2.2.1) with different receptive fields by removing pooling layers. Since in the VGG16 architecture all downsampling is performed by pooling layers, this is the most effective way to influence the receptive field. Since pooling layers do not contain trainable parameters, this also has the added benefit that in removing them we do not affect the depth of the network. In the context of this experiment, we refer to the original VGG16 architecture as $VGG16_{212}$ which indicates its receptive field of $212px$ (calculated from a neuron on the last feature map). Furthermore we create a $VGG16_{108}$ and a $VGG16_{58}$ network by removing the *pool1* and the *pool2* layer. In order to ensure that the weights of each network are optimally adapted to objects of a specific size, each network is pre-trained on an appropriately[1] scaled version of ImageNet [15].

We tightly cut out logo instances from the FlickrLogos-47 dataset and scale them to a specific size. In this way we create five datasets: $\mathcal{D}_{256}$, $\mathcal{D}_{128}$, $\mathcal{D}_{64}$, $\mathcal{D}_{32}$ and $\mathcal{D}_{16}$

[1]The $VGG16_{212}$ network was pre-trained on images of size $224px \times 224px$. Pre-training on $VGG16_{108}$ was done with $112px \times 112px$ images. $VGG16_{58}$ was pre-trained on images of size $56px \times 56px$.

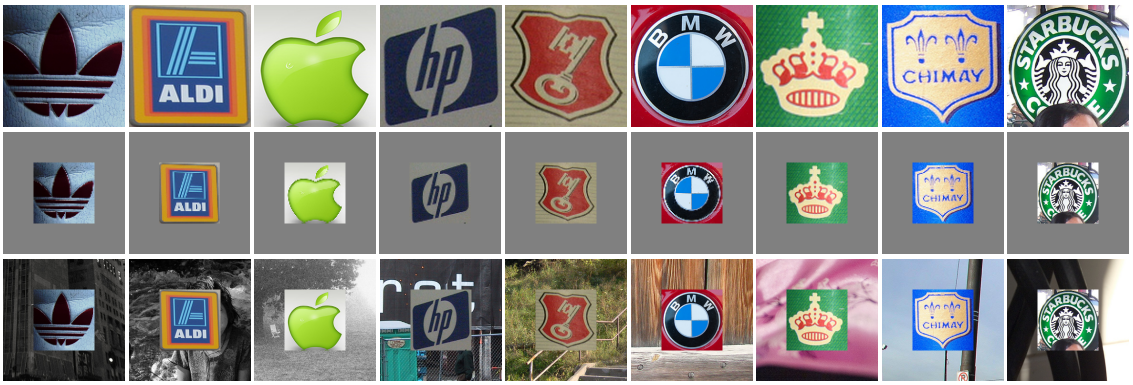


Figure 4.5: Examples from the datasets used to test classification performance as a function of object size and receptive field. First row: Instances from the $\mathcal{D}_{256}$ dataset. Second row: Instances from the $\mathcal{D}_{128}$ dataset (zero padding). Third row: Instances from the $\mathcal{D}_{128}$ dataset (random padding).

with an object size of $256px$, $128px$, $64px$, $32px$ and $16px$, respectively. Each dataset version consists of the same image corpus which is split into a training and test set.

We want to train and evaluate each network with each of the dataset versions. To do this, we need to pad images which are smaller than the receptive field of the network to the correct dimensions. We perform two types of paddings: *zero padding* and *random padding*, where we paste the object instance into a random image containing no logos. Examples of the dataset are shown in Figure 4.5.

We then fine-tune each network once for every dataset version and record the classification performance as top-1 accuracy on the corresponding test set. Table 4.3 shows the results of this evaluation. We can see a consistent pattern where the classification performance is always maximal when the object size fits right into the receptive field of the network, which confirms once more our initial observation. It is also apparent that a network with a small receptive field can achieve comparable performance on small objects as a network with a large receptive field on large objects. This observation is important since it shows a way to improve the performance of Fast R-CNN.

A comparison between the results of zero-padded and random-padded images also gives us insights into why the receptive field of a network is important: When there is an extreme mismatch between receptive field and object size, the networks trained on randomly-padded images often do not converge. A failure to converge manifests itself the following way: The activations of the last feature maps – and therefore the input to the classification layers – consist of constant zeros, for any input image. Neurons which do not activate under any circumstances are unable to update their weights anymore, and the training process is stuck.

Although we have also observed this behavior with zero-padded images, we most often have found it to be associated with randomly-padded images. Because of this

	Zero padding			Random Padding		
	$VGG16_{212}$	$VGG16_{108}$	$VGG16_{58}$	$VGG16_{212}$	$VGG16_{108}$	$VGG16_{58}$
$\mathcal{D}_{256}$	0.867			0.870		
$\mathcal{D}_{128}$	**0.883**	0.879		**0.879**	0.880	
$\mathcal{D}_{64}$	0.879	**0.886**	0.878	0.875	**0.882**	0.877
$\mathcal{D}_{32}$	0.866	0.872	**0.878**	0.063*	0.064*	**0.879**
$\mathcal{D}_{16}$	0.798	0.065*	**0.845**	0.748	0.063*	**0.824**

Table 4.3: Classification performance (top-1 accuracy) of networks with different receptive fields on objects of a particular size. Generally, classification performance is strongest when objects fit right into the receptive field of the network. Networks with large receptive fields have difficulties adapting to extremely small objects (especially when objects are embedded in a noisy context) and may even fail to converge (*).

observation, we speculate that a properly-sized receptive field helps the network to separate the signal (the object) from the noise (the surroundings).

The failure of a network to converge can be fixed by replacing the activation function in the last layers with a leaky ReLU [58] (lReLU) during fine-tuning.

$$lReLU(x) = \begin{cases} \gamma x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{4.5}$$

However, this has to be done carefully: If we replace the activation function of too many layers or if we choose the slope γ for the negative half-space too large, the gradient magnitudes in the first iterations can be very large which results in the destruction of the pre-trained weights. We found that replacing the *conv5* activation functions with lReLU using $\gamma = 0.01$ works well in practice.

Difference to general object detection It remains to be addressed, why Girshick [30] was able to report improvements in both detection performance as well as in runtime for Fast R-CNN compared to R-CNN. As we have seen in Chapter 4.2.2, we were also able to report substantial improvements in runtime, but we found the detection performance of Fast R-CNN to be inferior to R-CNN. Since unlike R-CNN, Fast R-CNN offers an end-to-end trainable pipeline and thus a joint optimization over feature extraction and classification, it intuitively makes sense that Fast R-CNN has the potential to offer superior detection performance compared to R-CNN which needs to optimize classifier and feature extractor separately. However, we have also shown in this chapter, that object size plays a vital role for the performance of the Fast R-CNN pipeline as small object are not detected nearly as well as large objects.

In the original paper, Girshick was working on general object detection using the PASCAL VOC dataset [23] for his experiments. The PASCAL VOC dataset differs from the FlickrLogos-47 dataset in two critical ways: (1) Objects in the PASCAL

VOC dataset tend to be larger than in the FlickrLogos-47 dataset. The mean object size in the FlickrLogos-47 dataset is $103px$ (measured in $\sqrt{area}$) while the mean object size in PASCAL VOC is $140px$. (2) The average image size is significantly smaller in PASCAL VOC ($384px \times 473px$) than in FlickrLogos-47 ($708px \times 1001px$). Therefore, not only the objects itself tend to be larger in PASCAL VOC, but also the object size in relation to the image.

In the original implementation by Girshick [30] the images are resized in such a way that the smaller side of the image is scaled to $600px$. However, this rescaling is limited such that the larger side will not exceed $1000px$. If we examine how this rescaling affects the average object size in the dataset, we find, that after this resize operation the average object size is $231px$. This represents a significant difference to the $103px$ in the FlickrLogos-47 dataset. Scaling our objects in the same way as Girshick would result in images which would not fit into GPU memory[2] anymore and is therefore not a viable strategy. In Chapter 4.4 we will discuss an alternative strategy.

4.3 Faster R-CNN

4.3.1 Network architecture

As we have seen in previous chapters, Fast R-CNN is able to improve the processing time per image compared to R-CNN substantially. However, the runtime per image is still in the range of seconds and is dominated by the object proposal stage. As we have seen in Chapter 3.5.2, RPNs can generate object proposals extremely fast and are therefore a promising way to reduce processing time. Although its localization properties are not on par with the examined heuristic object proposals, they offer a higher maximum achievable recall.

Both properties have the potential to affect the performance of the classification stage: (1) The inferior object localization might negatively affect the classifier. A classifier is more likely to produce a correct classification with high confidence when it is presented with an image patch in which the object in question is not truncated and potentially even centered. (2) On the other hand, objects proposals with high recall give the classification stage the opportunity to detect more objects. Faster R-CNN [66] combines the fast classification of Fast R-CNN with the speed of object proposals by RPNs. It remains to be seen which of the properties of RPN-based proposals has a stronger influence on the classifier.

In order to achieve this goal, Faster R-CNN adopts the network structure of the Fast R-CNN classification stage without any modifications. Faster R-CNN introduces an RPN which operates after the last convolutional layer of the VGG16 network (see Figure 4.6). The RPN outputs a list of proposals which are then used in the same way as for Fast R-CNN: The regions on the feature map which correspond

[2]We are using an nVidia Titan X (Pascal) with 12GB of GPU memory for our experiments.

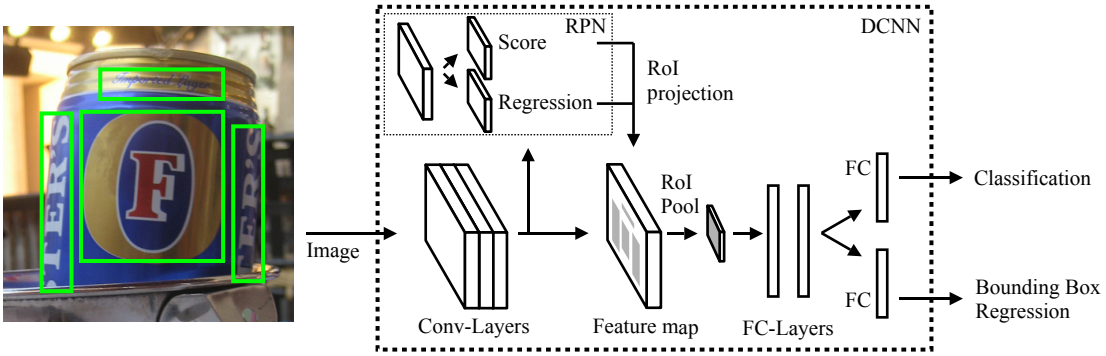


Figure 4.6: Overview of the Faster R-CNN object detection pipeline. The classification stage of Fast R-CNN is adopted without modifications. Instead of externally supplied object proposals, proposals are generated inside the network by an RPN.

to the proposals are subjected to ROI-Pooling. Each ROI is classified separately and undergoes a class-specific bounding box regression.

By virtue of this design, Faster R-CNN inherits the same problems regarding object size – as discussed in Chapter 4.2.3 – from Fast R-CNN. Therefore, the same strategies for improving detection performance also hold true for Faster R-CNN. In particular, our selective magnification strategy, which we will discuss in Chapter 4.4 can in principle also be applied to Faster R-CNN. In this chapter, we are concerned solely with the differences in detection performance between Fast R-CNN and Faster R-CNN on unscaled images.

4.3.2 Results on FlickrLogos-47

Training Procedure For training, we use our batch-normalized VGG16 network as base network. We extend this network by adding layers for FlickrLogos-47 classification and bounding box regression in the same way as for Fast R-CNN (see Chapter 4.1.1). We also attach a Region Proposal Network (RPN) to the *conv5* feature map of the VGG16 network. We adopt the *Approximate Joint Training* approach proposed by Ren et al. [66] which requires the joint optimization of four separate loss function: (1) The Softmax cross entropy loss for the ranking of object proposals. (2) The Huber loss [41] used for the bounding box regression of the object proposal stage. (3) The Softmax cross entropy loss for predicting the class label of the object proposals. (4) The Huber loss used for bounding box regression of the classification stage. This approach avoids the cumbersome alternating optimization in which the RPN stage and classification stage are trained separately while the other stage is being held fixed.

This approach is close to a full joint optimization of the model. However, it is still an approximate optimization because it ignores gradients which might flow from the ROI-Pooling layer back into the RPN. In order to allow a full optimization,

Proposals		$n = 300$		$n = 1000$		$n = 2000$	
F. Map	Scales	mAP	Runtime	mAP	Runtime	mAP	Runtime
conv5	$\mathcal{S}_{ext}$	**0.642**	$0.11s$	0.639	$0.13s$	0.637	$0.18s$
conv5	$\mathcal{S}_{prop}$	**0.668**	$0.10s$	0.667	$0.13s$	0.662	$0.18s$
multires	$\mathcal{S}_{prop}$	**0.687**	$0.31s$	0.680	$0.34s$	0.672	$0.40s$

Table 4.4: Detection performance of Faster R-CNN on FlickrLogos-47 for different anchor sets. Our improved region proposals clearly translate into improved detection results. Large numbers of proposals tend to be detrimental to detection performance.

the ROI-Pooling layer would need to be differentiable with respect to the proposal coordinates. This is a difficult task and does not constitute a promising route for further improvements since Ren et al. [66] compare several modes of optimization with each one yielding comparable results.

We employ the same sampling strategies that we have discussed in the context of RPNs (see Chapter 3.5.1) and Fast R-CNN (see Chapter 4.2.2) and train the network for 60000 iterations. We start with an initial learning rate of $\lambda = 0.001$ and reduce the learning rate after 40000 iterations by a factor of 10.

Results For our evaluation of Faster R-CNN on the FlickrLogos-47 dataset, we make use of our improved object proposals from Chapter 3.5.4 and want to ascertain whether the improved recall can translate into better detections. We compare three different sources for object proposals which we have already used in our evaluation of the RPN: (1) Object proposals which have been generated from a single feature map (*conv5*) using the $\mathcal{S}_{ext}$ set of anchor scales. $\mathcal{S}_{ext} = \{32px, 64px, 128px, 256px\}$ uses the same powers-of-two scale space sampling as in the original Faster R-CNN implementation [66] but the scales have been adopted to fit the object sizes in the FlickrLogos-47 dataset. (2) Object proposals which have been generated from a single feature map (*conv5*) but using our improved $\mathcal{S}_{prop}$ set of anchor scales ($\mathcal{S}_{prop} = \{32px, 45px, 64px, 90px, 128px, 181px, 256px\}$). (3) Object proposals generated from multiple features maps (*multires*) using our improved anchor set $\mathcal{S}_{prop}$.

The results of this evaluation are shown in Table 4.4. In a direct comparison using 2000 proposals during evaluation we can see that the detection performance of Faster R-CNN is approximately on par with the performance of Fast R-CNN (see Table 4.2 in Chapter 4.2.2). When Fast R-CNN is used in conjunction with our improved set of anchor scales $\mathcal{S}_{prop}$ we can achieve a slightly better detection performance as when using Fast R-CNN with our best-performing heuristic proposals. In addition to slightly improved detection performance, Faster R-CNN is also able to improve the runtime significantly.

Another interesting result from Table 4.4 is that reducing the number of proposals during test time is actually beneficial to the performance. This behavior can easily be explained through an observation we have discussed when evaluating RPN-

based proposals in Chapter 3.5.2: RPN-based object proposals are able to retrieve most relevant object with very few proposals. We have observed that the recall does not improve significantly anymore if we increase the number of proposals beyond 300 proposals. Therefore, the classification stage does not have more opportunities to detect objects with more proposals. However, it does have the opportunity to make more mistakes. Increasing the number of proposals, therefore, cannot lead to more true positives, but more false positives which negatively affects the detection performance.

In conclusion, Faster R-CNN is able to improve detection performance slightly and significantly reduces the runtime compared to Fast R-CNN. However, neither Fast R-CNN nor Faster R-CNN can rival the detection performance of R-CNN.

4.4 Selective Magnification

In chapter 4.2.2 we have seen that the detection performance of Fast R-CNN falls strongly behind the performance of R-CNN. We analyze the problem in chapter 4.2.3 and have identified the receptive field as one cause of the inferior performance of Fast R-CNN on FlickrLogos-47. In this chapter, we want to make use of our observations and investigate methods to improve our detections.

This chapter is based on our work *Saliency-guided Selective Magnification for Company Logo Detection* [19]. Most graphs and results are taken directly from this paper. It should be noted that these results have been generated with an older version of the detection pipeline and an older version of the FlickrLogos-47 dataset than in Chapter 4.2.2. In particular, the old dataset version did not have annotations for *difficult* and *truncated* logo instances. The old detection pipeline did not have support for excluding *difficult* object instances from training. Additionally, while we also used VGG16 as base network for detection, this network did not include batch normalization [43]. Therefore, the results are not directly comparable to the results reported in chapter 4.2.2. However, in this chapter, we will provide the relevant baselines generated with the old version of the detection pipeline for comparison.

4.4.1 Overview

Our previous experiments suggest that we could increase the performance for small objects if we reduce the size of the receptive field. However, doing this would come at the cost of reliably detecting objects larger than the receptive field.

One possibility to address this problem would be to create a separate network with a smaller receptive field. Object proposals could be partitioned into two sets, small and large. The classification stage could be executed twice, once for each set of object proposals. The small proposal set could be processed by the network with a small receptive field while the network with a wide receptive field could be responsible for processing the large proposal set.

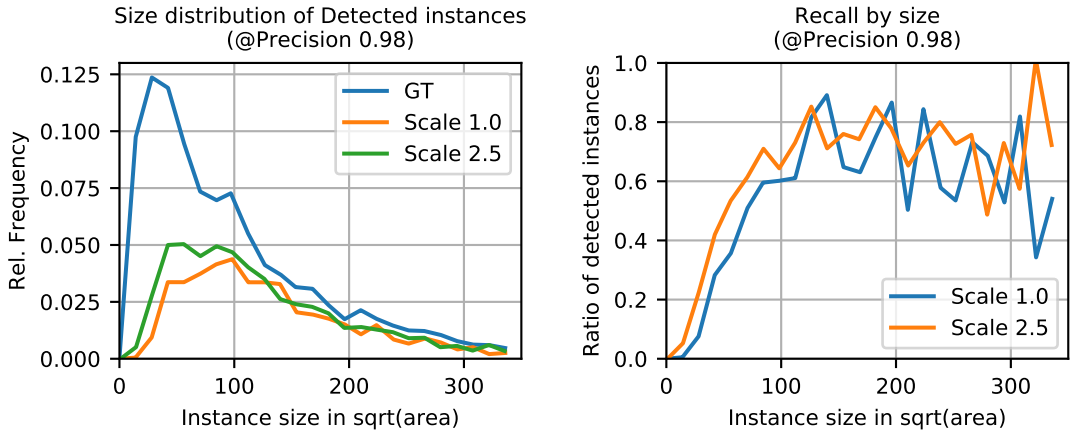


Figure 4.7: Detected objects using Fast R-CNN by size. Plots were generated from histograms with a bin width of $12px$. Small objects tend to be detected with a lower confidence score. Magnifying the image improves detection performance for small objects, even for high-precision detections. For objects with a sidelength up to approx. $100px$ magnification tends to be beneficial while larger objects do not benefit from magnification.

While this approach would work in principle, there are some practical problems with implementing it. One problem is that creating a network with a smaller receptive field implies less downsampling between the feature maps. Because of the increased resolution of the feature maps, this also increases memory usage and computational effort. Another problem with this approach is that it would also require duplicating the fully-connected layers which contain almost all model parameters. Since the standard model already contains 122×10^6 parameters in the *fc6* and the *fc7* layers, such an inflated model is hard to justify. This is especially true when considering that the number of training examples which can be used for each network is essentially reduced by half. Although it would be possible to counter this problem by data augmentation, we propose a different approach.

Instead of duplicating the network, we propose to keep a single network but to run the classification on two scales: One time with the original image and one time with an upsampled version of it.

As we have seen in Chapter 4.2.2, upsampling the images tends to improve detection performance. Figure 4.7 breaks down these performance gains by object size. Small object instances tend to be classified with lower confidence than large objects. This results in many objects not being detected, especially when high precision is required. For objects with a sidelength up to $100px$ a clear trend is visible: Magnification tends to be beneficial for detection. For larger object instances, no such trend is visible.

This is in line with our observations about the receptive field in Chapter 4.2.3 which suggest that a classifier works best when the object to be classified has ap-

proximately the same size as the receptive field of the network: Since the receptive field of the VGG16 network is $212px$ we would expect objects up to a maximum size of $85px$ to benefit from a magnification by a factor of 2.5.

However, upsampling the image is not without its downsides. Because the computational effort of applying the network is dominated by the feature extraction in the convolutional layers, magnifying the image by a factor of α increases the computational effort for running the classification approximately by a factor of α^2. The same problem exists for memory which especially on GPUs is a scarce resource. In order to reduce memory and computational effort, we propose to upsample only parts of the image which are likely to contain objects.

To predict which regions are likely to contain objects, we take the original image and run the classification stage with all proposals, regardless of size. For every proposal, the ROI-Pooling layer generates a fixed-dimensional feature representation which we use in two ways: (1) We run the features through the classification and bounding box regression layers to obtain a list of detections that is identical to the original Fast R-CNN implementation. (2) We use an SVM [13] to predict whether the image patches which correspond to these features are likely to contain an object.

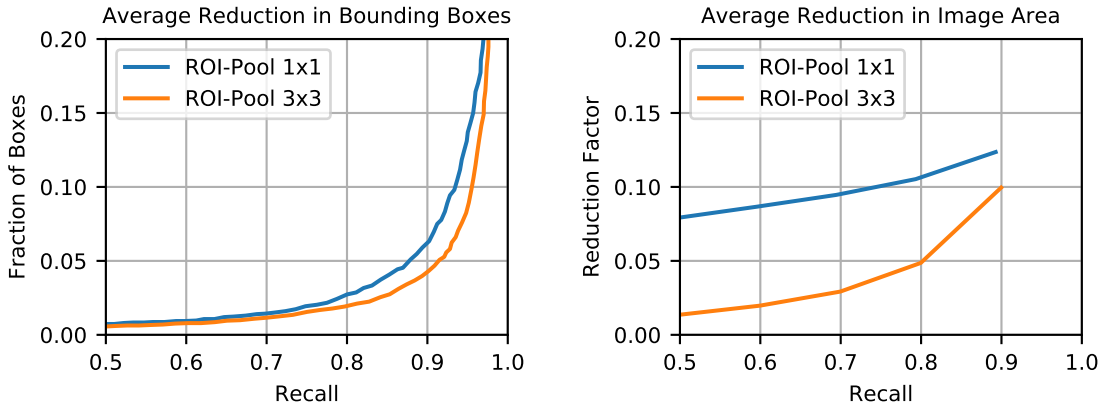
Proposals which have been deemed interesting enough to warrant closer examination are then magnified and fed again into the classification stage. The detection results from the original image and the magnified proposals are merged into a single list of detections. A non-maximum suppression is run on them to produce the final detection.

The intuition behind this idea is that even if there exists a low-confidence detection on the original image, magnifying the input to better fill the receptive field of the network might generate another detection at the same location with higher confidence. If another detection with higher confidence is generated through magnification, the non-maximum suppression will suppress the first detection.

4.4.2 Selecting object proposals for magnification

To select proposals for magnification we first need to select a size threshold s_t for dividing proposals into small and large proposals. Proposals whose size exceed this threshold are not considered for magnification. Since the receptive field is the upper limit beyond which magnification is not beneficial anymore, the threshold itself is dependent on the magnification factor we want to select.

For the purposes of this evaluation we choose a factor of 2.5 for several reasons: Upsampling an $1024px \times 1024px$ image – which is the largest image size in our dataset – by a factor of 2.5 corresponds to the maximum image size we are able to fit into GPU memory. Since we want to compare our selective magnification strategy with brute force upsampling we require such a baseline for comparison. Figure 4.7 shows the benefits of brute force upsampling.



(a) (b)

Figure 4.8: Effectiveness of selecting proposals for magnification for different spatial resolutions of ROI-Pooled features: (a) Average reduction in the number of proposals and the (b) corresponding reduction in image area as a function of recall[3]. It is possible to tune the SVM to prune most proposals while maintaining a large fraction of relevant proposals.

Together with a receptive field of $212px$ at the *conv5* feature map of the VGG16 network this magnification factor would place an upper bound on our size threshold at $85px$. We select a threshold of $s_t = 80px$ to provide us with a little headroom.

Object proposals with an $IoU \geq 0.5$ with at least one groundtruth instance and whose sidelength is smaller or equal than s_t are assigned to the set $\mathcal{P}_s$ which comprises the set of small positive training examples. Object proposals with an $IoU < 0.5$ with all groundtruth instances and whose sidelength is smaller or equal than s_t are assigned to the set of small negative training examples $\mathcal{N}_s$. We only consider examples in these two sets for training. Note, that large objects are not used for either the postive or the negative set. This training mode does not force the classifier to draw a dividing line between small positive proposals and large positive proposals. It is of no consequence if some large positive proposals are selected by the SVM for magnification as long as all small positive proposals can be retrieved.

We use all elements from $\mathcal{P}_s$ for training and sample examples from $\mathcal{N}_s$ in a $1:1$ ratio. The SVM is trained in a single round. We do not employ any hard negative mining because we are not overly concerned with false positives in our classifier.

To extract the features for the SVM, we use an additional ROI-Pooling layer to provide fixed-size feature vectors with a spatial resolution of 3×3. With the 512 channels of the *conv5* feature map this results in a 4608-dimensional feature vector for each proposal.

[3]Unfortunately, the value for a recall of 1.0 is not available in [19] and the raw-data used to generate the plot was not available anymore.

Figure 4.9: Examples for selective magnification using images containing a large number of object instances. Groundtruth instances smaller than $80px$ are marked in green. Image regions selected selected for magnification are marked in red. Image areas which belong to groundtruth instances smaller than $80px$ and have been marked for magnification are displayed in blue.

The reason for including another ROI-Pooling layer is to reduce the dimensionality of the feature vectors compared to the 7×7 ROI-Pooling which is already present in the Fast R-CNN network. We do this out of practical considerations rather than necessity: The 7×7 ROI-Pooling which is used for class membership prediction and bounding box regression produces a very high dimensional feature representation. In order to train the SVM we need to hold all training examples in memory. To reduce the memory footprint of the training we reduce the spatial resolution of the feature vectors.

We have considered features with a spatial resolution of 1×1, 3×3 and 5×5. We have found that the performance of 5×5 features is nearly identical to the performance of 3×3 features which is why we use a 3×3 feature representation.

Only the 3×3 ROI-pooled features of object proposals which have a $\sqrt{area} \leq 80px$ are fed into a linear SVM which selects proposals that should be examined on a larger scale. This SVM does not necessarily need high precision. Our only requirement for the classifier is to maintain high recall while excluding as much of the image area as possible. As Figure 4.8 shows, we can tune the SVM to prune most proposals so that the image patches that are classified as positive are sparsely distributed across the image while still maintaining a large fraction of relevant objects. Figure 4.9 visualizes the result of this selection mechanism for some example images. It is apparent that almost all small positive proposals are selected by the SVM for closer examination (along with some large positive proposals).

4.4.3 Efficient Rectangle Packing

After having identified proposals which warrant closer examination, we want to apply the classification stage to the magnified image patches, hoping for a more confident classification. In theory, we could magnify and classify each proposal separately. However, the considerable degree of overlap between proposals would

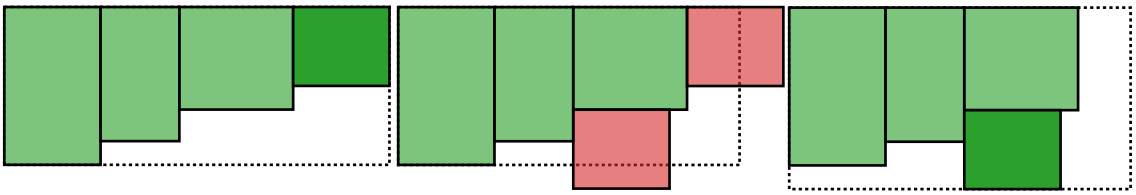


Figure 4.10: Finding a mininum-area enclosing rectangle using the algorithm by [47]: Rectangles are placed into an enclosing rectangle, ordered by height. The rectangle with the largest height is placed first. (a) The algorithm begins with an enclosing rectangle that is guaranteed to accommodate all rectangles. (b) The width of the enclosing rectangle is reduced step by step until placement of rectangles within this enclosing rectangle fails. (c) The height of the enclosing rectangle is increased until placement of rectangles within this enclosing rectangle succeeds. These steps are repeated until the enclosing rectangle's width is equal to the maximum width of the individual rectangles. The enclosing rectangle of minimum area for which the rectangle packing succeeded is the used as result.

not be exploited. The Fast R-CNN approach would degenerate into a slower R-CNN pipeline. In order to avoid redundant computation, we identify connected components of overlapping proposals. We circumscribe a bounding box around each of these connected components.

While selective magnification of the connected components is more efficient than selective magnification of the individual proposals, it still requires to transfer each connected component to GPU memory separately which can be quite time-consuming. To eliminate the need for multiple GPU transfers we paste all connected components into a single image which is subsequently magnified and classified.

The computational cost of applying the VGG16 network is dominated by the feature extraction in the convolutional layers. Since the computational cost of applying a convolution is dependent on the image area, we want to minimize the image area into which we paste the connected components. For this purpose, we employ a rectangle packing algorithm.

The task of a minimum-area rectangle packing algorithm is to position a set of given rectangles $\mathcal{R}$ in such a way that the enclosing rectangle has minimum area. In general, finding an optimal solution to this problem is NP-hard [47, 48]. However, there exist near-optimal heuristics which allow for efficient rectangle packing.

We implement a rectangle packing algorithm based on the algorithm by R. Korf [47] which splits the task of minimum rectangle packing into two sub-tasks: (1) Placing rectangles within an enclosing rectangle of fixed dimensions. (2) Finding an enclosing rectangle with minimum area containing all rectangles.

For a description of the algorithm, we will assume for the moment that we already have a way of placing rectangles into an enclosing rectangle. We will denote the height and width of a rectangle $r \in \mathcal{R}$ as $h(r)$ and $w(r)$, respectively. The

algorithm starts with an enclosing rectangle $\hat{r}$ of height $h(\hat{r}) = \max_{r \in \mathcal{R}} h(r)$ and width $w(\hat{r}) = \sum_{r \in \mathcal{R}} w(r)$.

Assuming the existence of an algorithm which can place rectangles into an enclosing rectangle, this choice of enclosing rectangle guarantees successful placement of $\mathcal{R}$ within it. To search for a minimum area rectangle, the width of $\hat{r}$ is decreased in steps until the algorithm placing $\mathcal{R}$ within $\hat{r}$ fails to find a solution.

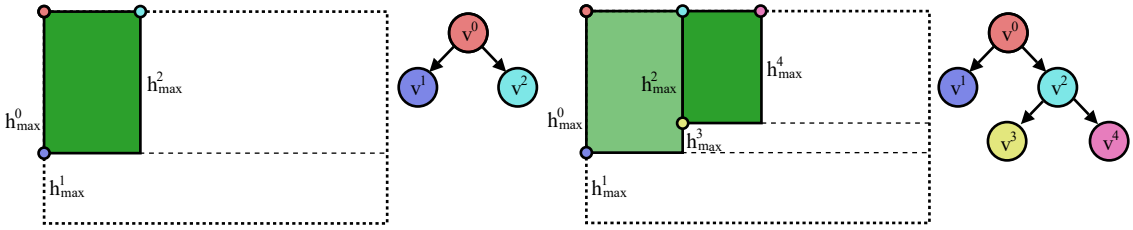
When this happens, the height of $\hat{r}$ is increased in steps until the algorithm which places rectangles in $\hat{r}$ succeeds again. In subsequent steps, the width of $\hat{r}$ is decreased again until rectangle packing fails. These steps are repeated until $w(\hat{r}) \leq \max_{r \in \mathcal{R}} w(r)$. The configuration with the smallest enclosing rectangle $\hat{r}$ for which the rectangle packing has succeeded is returned as solution. This algorithm is visualized in Figure 4.10.

It remains to be resolved, how to pack rectangles into an enclosing rectangle of fixed dimensions. Note, that this framework will return a solution, no matter how this packing algorithm is chosen. However, since we still need to search over many possible enclosing rectangles, the packing algorithm needs to be fast. Also, the choice of algorithm affects the quality of the solution. The better the algorithm manages to concentrate all rectangles in the top-left corner of the enclosing rectangle without a lot of wasted space between the individual rectangles, the smaller the minimum enclosing rectangle will be on average. In [47] multiple algorithms for solving this problem are discussed.

Korf [47] proposes a simple greedy strategy: The rectangles $r \in \mathcal{R}$ are placed inside the enclosing rectangle in order of decreasing height. Each time a rectangle r is placed we choose the leftmost position in the enclosing rectangle where there is room to accommodate r. If multiple such locations exist, we choose the uppermost of these locations. Should no such location exist, the rectangle packing has failed, and we continue our search with an enclosing rectangle of increased height.

While this algorithm is simple to describe, it is not simple to implement efficiently: The packing algorithm needs to solve two problems: (1) Multiple potential placement locations need to be evaluated. (2) For each potential placement location, the algorithm needs to check whether placing a rectangle is actually possible, i.e., the rectangle to be placed does not overlap with any previously placed rectangles. After a few placed rectangles, the area occupied by the placed rectangles can look highly irregular. This problem makes checking whether a placement location is feasible increasingly harder. While in principle these tests could also be accelerated by using an efficient indexing structure (for example through k-d trees [7]) the algorithm remains difficult to implement.

We use our own heuristic for packing rectangles which uses a similar idea but is easier to implement since it almost completely eliminates step (2). Our algorithm is based on a simple search in a binary tree. For each node $v^i \in \mathcal{V}$ in our tree we store a potential placement location $\boldsymbol{p}^i = (p^i_x, p^i_y)$ and the maximum height h^i_{max} of a rectangle that can be placed at this location. At the beginning, our tree consists



(a) (b)

Figure 4.11: Placing rectangles within an enclosing rectangle via tree search. Each node of the tree corresponds to a potential placement location and records the maximum allowed height of a rectangle to be placed at this location. Before placement of the first rectangle the search tree consists of a single node representing the upper left corner. Rectangles are placed in order of decreasing height. (a) Each rectangle placement results in two child nodes being added to the tree. (b) To find a suitable placement, the nodes are traversed in-order to find the first node which can accommodate the height of the rectangle.

of a single node v^0 with $\boldsymbol{p}^0 = (0, 0)$ and h^0_{max} is set to the height of the enclosing rectangle.

Similar to the algorithm proposed by [47] we place rectangles in order of decreasing height. Each time we place a new rectangle r, we consult our tree for potential placement locations (we will discuss how such a suitable location is found in a moment). After we have found a suitable node v^i corresponding to a placement location we add two child nodes $v^{i,left}$ and $v^{i,right}$ representing the left and right child, respectively. For the left child $v^{i,left}$ we set $\boldsymbol{p}^{i,left} = (p^i_x, p^i_y + h(r))$ and $h^{i,left}_{max} = h^i_{max} - h(r)$. For the right child node $v^{i,right}$ we set $\boldsymbol{p}^{i,right} = (p^i_x + w(r), p^i_y)$ and $h^{i,right}_{max} = h(r)$.

The result of adding these child nodes is that two new possible positions for rectangle placements are created: One position corresponding to the left child node is located directly below the currently placed rectangle. The position corresponding to the right child node is located directly to the right of the currently placed rectangle. This is illustrated in Figure 4.11.

To find a suitable location for placing a rectangle, we traverse the tree in-order by recursively checking the left sub-tree first. If we encounter a leaf node, we have found a potential placement location. We check whether the height $h(r)$ of the rectangle to be placed can be accommodated by this placement location. If we can, we place the rectangle at this location while updating the tree as described above. If this location cannot accommodate r, we continue with our search. Should we arrive at the rightmost leaf node without being able to accommodate r, the rectangle packing has failed, and we need to continue with a different enclosing rectangle. Example outputs of this algorithm can be seen in Figure 4.12.

While our simple heuristic is likely to produce inferior results to the algorithm proposed by Korf [47], we can see from the examples in Figure 4.12 that the results

Figure 4.12: Selected examples of the output of our rectangle packing algorithm. Proposals which are marked for magnification by the SVM are displayed in red. These proposals often overlap. Connected components of overlapping proposals are represented as a single image patch which is used as input to the rectangle packing algorithm. Also shown are the bounding boxes of individual proposals within each connected component.

seem to be adequate. By using this simple algorithm, checking potential placement locations is reduced to a simple lookup of the remaining height.

4.4.4 Evaluation

To evaluate the effectiveness of our selective magnification strategy we compare three approaches: (1) An unaltered Fast R-CNN based object detection with no magnification, which we call *Fast R-CNN*. (2) A Fast R-CNN based object detection pipeline in which all input images are upsampled by a factor of 2.5. We will call this approach *Direct Magnification (*$2.5x$*)*. (3) A Fast R-CNN based detection pipeline in which we employ Selective Magnification as described in this chapter. Again, we use a magnification factor of 2.5 for the images which are the output of the rectangle packing algorithm. We call this approach *Selective Magnification (*$2.5x$*)*.

Since we are interested in the performance gains of the classification stage, we are not overly concerned by the source of our object proposals because the proposals we use are the same for every training mode. However, since we expect magnification to be particularly relevant for small object instances, we want to make sure that our proposals adequately cover small objects.

We use our SS+EB+VH approach from chapter 3.5.2 which combines Selective Search [77], Edge Boxes [85] and VH-connect (see chapter 3.4.3) to generate object proposals. We use up to $n = 8000$ proposals per image. The reason for this rather large number of proposals is to increase the probability that we have object proposals covering small objects (especially Selective Search is biased towards large objects).

In each case, we train the Fast R-CNN pipeline for 40000 iterations using an initial learning rate $\lambda = 0.001$ and a learning rate reduction by a factor of $\gamma = 0.1$

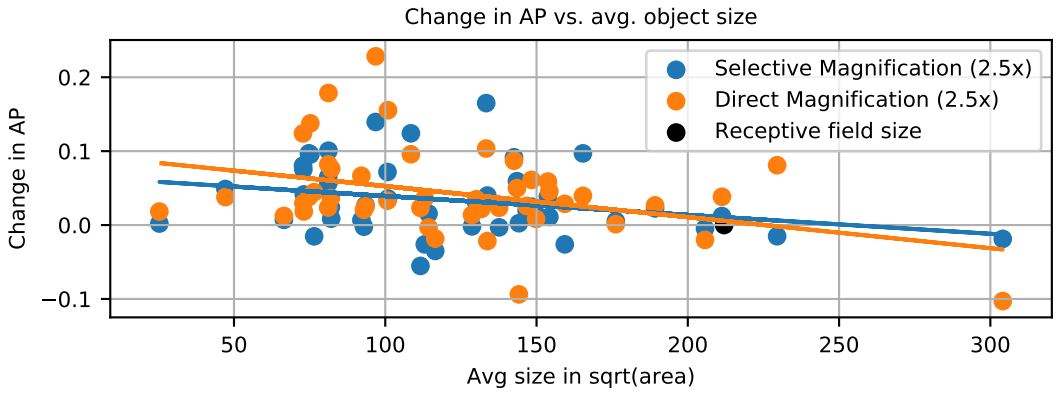


Figure 4.13: Comparing selective magnification to direct magnification. Shown is the performance gain for each object class (measured as the difference in AP relative to the AP at magnification factor 1.0) as a function of the average instance size of this class. Magnification is mostly beneficial to detection performance for small objects and detrimental for large objects. The receptive field seems to limit the object size for which magnification is beneficial.

after 25000 iterations. For our Selective Magnification approach, we tune the SVM so that it retains approx. 5% of the bounding boxes smaller than $80px$.

As a result, we obtain for our classical *Fast R-CNN* approach a mAP of 0.668 with an average processing time per image of $0.82s$ (excluding object proposals). The *Direct Magnification (*$2.5x$*)* approach is able to improve the detection performance to 0.711 (mAP). However, this performance gain comes at the expense of prolonged processing time. Compared to *Fast R-CNN* the processing time per image has more than tripled and is now approx. at $2.83s$ per image. *Selective Magnification (*$2.5x$*)* can retain most of the performance gains (mAP 0.702) but requires an only slightly longer processing time ($1.03s$ per image).

Figure 4.13 plots the differences in AP for every class for both *Direct Magnification (*$2.5x$*)* and *Selective Magnification (*$2.5x$*)* compared to the classical *Fast R-CNN* approach. We plot the difference in AP as a function of the average instance size – measured on the original, non-magnified dataset – of an object in this class. It becomes clear that there is a relationship between the performance gain through magnification and the object size: Small objects tend to benefit most strongly from magnification while for large objects it can be detrimental to detection performance. This correlation is pronounced less strongly for *Selective Magnification (*$2.5x$*)* than for *Direct Magnification (*$2.5x$*)*, but it is still clearly visible with some object classes gaining more than 0.1 in AP (absolute value) which is a substantial improvement.

Another interesting observation can be made when we plot the regression lines for these data points. The point where the regression line intersects the x-axis is the maximum object size for which we can expect any improvements through magnification. As we can see from Figure 4.13, the intersection point is very close to an object

size of $212px$ which is the size of the receptive field of the network. This observation provides another indication for our hypothesis from Chapter 4.2.3 that classification with neural networks performs best when the object size is approximately equal to the size of the receptive field.

4.5 Conclusion

We have evaluated three approaches for two-stage object detection: R-CNN, Fast R-CNN and Faster R-CNN. Both R-CNN and Fast R-CNN make use of heuristic object proposals, and we were able to show that our improvements regarding these proposals are able to translate into better detection performance. Faster R-CNN makes use of RPN-based object proposals. Similar to heuristic proposals, we were able to show that our RPN improvements allow for better detections using Faster R-CNN.

When we compare the different approaches against each other, we find that although R-CNN has a very long runtime, it outperforms both Fast R-CNN and Faster R-CNN by a considerable margin with regard to detection performance. While analyzing the reasons for this difference in performance we have found that Fast R-CNN and Faster R-CNN have problems in detecting small objects. We have identified the receptive field of the network as an important factor to classification performance and have found that a classifier performs best when the object to be classified has approximately the same size as the receptive field of the network.

We have shown that simple upscaling of the images can improve the confidence of the classifier when detecting small objects. Although we are unable to reach quite the same detection performance as R-CNN by upscaling the images in Fast R-CNN, we were able to obtain results which are at least comparable to R-CNN. Although we increase the runtime of Fast R-CNN considerably by upscaling the images, we were able to show that the runtime still remains lower than for R-CNN. To mitigate the impact on the runtime even further, we have introduced a selective magnification strategy. We have shown that our selective magnification strategy is able to retain most of the benefits of upscaling but only has a low impact on the overall runtime.

Therefore we can conclude that – if runtime is of no concern – R-CNN is the best choice when trying to detect small objects like company logos. If runtime does matter we can use Fast(er) R-CNN using a selective magnification scheme which can deliver a performance which is at least comparable to R-CNN.

Part III

Single-Stage Object Detection

Overview

So far, we have only considered two-staged object detection approaches in which the object proposal stage is responsible for identifying potential locations of object instances and the classification stage is responsible for assigning a class label and optionally refining the location through a bounding box regression. Two-stage object detection pipelines have the advantage that by limiting the evaluation of the classifier on potentially interesting image regions, they have the potential to generate fewer false positives. On the other hand, they also have the potential to miss many objects because they inherently limit the maximum achievable recall of the system.

This trade-off between higher recall in the proposal stage and fewer false positives in the classification stage is inescapable in two-staged object detection pipelines. We have already encountered this problem in the case of Faster R-CNN and RPN-based region proposals: For example, we have observed in Chapter 4.3.2 that Faster R-CNN performs better in terms of mAP when we use 300 RPN proposals than when we use 2000 RPN proposals. However, if we were to only look at our measurements for maximum achievable recall for RPNs in Chapter 3.5.4 we would expect the opposite. Therefore, it must be the false positive rate of the classification stage that is responsible for this behavior.

In the case of trainable object proposals, we need to ask ourselves an even more fundamental question. In two-stage object detection pipelines, two classification problems need to be solved. One problem is a binary classification problem of assigning image patches to two classes: *object* and *background*. The other problem is to distinguish image patches between the individual object classes and the *background* class.

It is not immediately obvious which of these problems is easier to solve: On the one hand, the first classifier only needs to separate the feature space into two parts. On the other hand, the generic *object* class of the proposal stage classifier needs to incorporate all the individual object classes and distinguish them from the background. Since the individual object classes can vary strongly in appearance, this means that this classifier needs to deal with lots of variation in the data. It might be an easier problem to separate the feature space into multiple parts, each responsible for a single object class.

A single stage object detection approach does precisely that. It does away with the object proposal stage and directly assigns an image path to the *background* class or an individual object class. In the following, we will examine the suitability of

such an approach for the task of company logo detection. In particular, we will take a closer look at the Single Shot MultiBox Detector [56] (SSD).

Another well-known approach for single-stage object detection is YOLO [65]. However, we exclude YOLO from our considerations for two reasons: (1) The design of YOLO inherently limits the approach to input images of fixed size. (2) The authors of YOLO admit in their paper [65], that YOLO struggles with small object instances. These two properties of YOLO make it unattractive for our evaluation.

We will see that the lessons we have learned by examining the object proposal stage and the classification stage can also be applied in the context of the SSD detection framework. Especially our results regarding the relationship between object size and the receptive field of a convolutional neural network (see Chapter 4.2.3) will prove to be useful. In both two-staged object detection approaches that we have examined, we found it difficult to exploit our results effectively and had to resort to methods like Selective Magnification (Chapter 4.4) which seem contrived. In the context of SSD, we will see that we are able to integrate our results regarding the receptive field of a classifier in a very natural way.

Chapter 5

The Single Shot MultiBox Detector

5.1 Original Implementation

On an abstract level, the *Single Shot MultiBox Detector* (SSD) works very similar to Region Proposal Networks (RPNs): SSD places a grid of bounding box candidates over the image and uses a fully-convolutional network architecture to classify every bounding box simultaneously. The authors of SSD [56] call these candidate boxes *default boxes*. Because this concept of *default boxes* is identical to the concept of *anchor boxes* we have described in the context of RPNs, we refer to these bounding box candidates as *anchor boxes* for consistency.

Similar to our multi-resolution approach to RPNs (see Chapter 3.5.4) SSD does not rely on a single layer for generating predictions. Instead, the network splits into multiple branches with each branch containing feature maps of a different resolution. Figure 5.1 provides an overview of the SSD network architecture using VGG16 as its base network.

Despite all the similarities to RPNs and especially our multi-resolution approach to RPNs, the network output of SSD is quite different: SSD not only performs an *object* versus *background* classification but directly classifies each bounding box into the individual object classes and the *background* class.

Anchor Grid The original SSD implementation by Liu et. al. [56] uses fixed-sized input images. Liu et al. evaluate input sizes of $300px \times 300px$ and $512px \times 512px$. Images whose size or aspect ratio differ from these dimensions are resized appropriately. Since the input size of the network is fixed, all intermediate feature maps have fixed sizes as well. Therefore, the anchor grid can be static for all images. Aside from the anchor grid being static, the definition of the anchor grid in SSD is identical to the one for RPNs: A set of anchor boxes is assigned to every pixel $\boldsymbol{x}_{\mathcal{F}}$ of a feature map $\mathcal{F}$. The center $\boldsymbol{x}_{\mathcal{I}}$ of every anchor (in image coordinates) box for a given feature map position $\boldsymbol{x}_{\mathcal{F}}$ is given by:

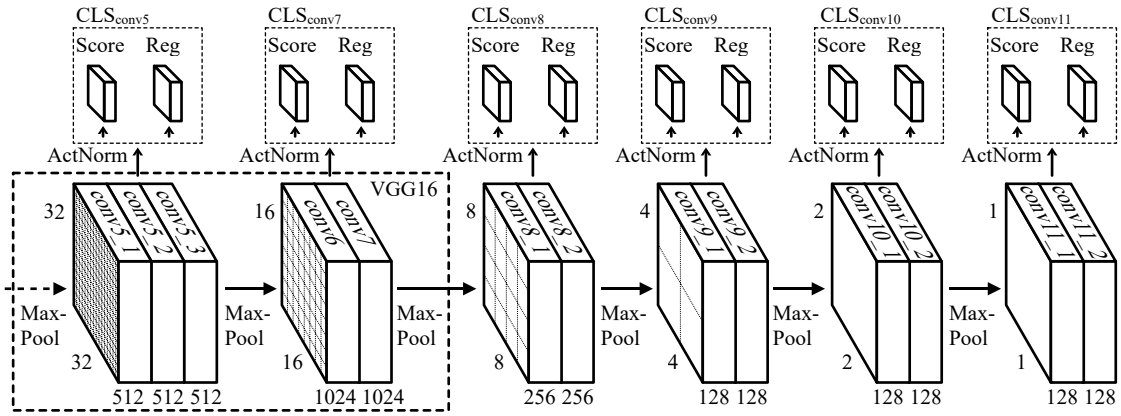


Figure 5.1: Network architecture of the original SSD implementation by [56] for an input size of $512px \times 512px$. The network is based on a VGG16 whose fully-connected layers have been converted into convolutional layers. Each branch of the network is responsible for predicting a single scale.

$$\boldsymbol{x}_{\mathcal{I}} = f_{\mathcal{I}}(\boldsymbol{x}_{\mathcal{F}}) = \alpha \boldsymbol{x}_{\mathcal{F}} + \left\lfloor \frac{\alpha}{2} \right\rfloor \mathbf{1} \tag{5.1}$$

Again, we assume that the side length of the image $l(\mathcal{I})$ and the feature map $l(\mathcal{F})$ are related through a downsampling factor $\alpha = \frac{l(\mathcal{I})}{l(\mathcal{F})}$. In the case of the VGG16 network, all downsampling is performed through max-pooling layers. Each max-pooling layer reduces the side length of a feature map by a factor of 2. Therefore, α is always a power of 2 in the VGG16 network.

From the center of every anchor the upper-left ($\boldsymbol{x}_{ul}$) and lower-right ($\boldsymbol{x}_{lr}$) vertices are then calculated as follows:

$$\begin{aligned} \boldsymbol{x}_{ul} &= f_{\mathcal{I}}(\boldsymbol{x}_{\mathcal{F}}) - \frac{s}{2} \begin{pmatrix} \sqrt{a}^{-1} \\ \sqrt{a} \end{pmatrix} \\ \boldsymbol{x}_{lr} &= f_{\mathcal{I}}(\boldsymbol{x}_{\mathcal{F}}) + \frac{s}{2} \begin{pmatrix} \sqrt{a}^{-1} \\ \sqrt{a} \end{pmatrix} \end{aligned} \tag{5.2}$$

where s represents the anchor scale (measured in $\sqrt{area}$) and $a \in \mathcal{A}$ represents the aspect ratio of the anchor box. The set of aspect ratios $\mathcal{A}$ was originally chosen to be $\mathcal{A} = \left\{\frac{1}{3}, \frac{1}{2}, 1, 2, 3\right\}$.

So far, the construction of the anchor grid has been fairly straightforward and is identical to the RPN anchor grid. The choice of anchor scales, however, is a little more complicated and in some ways inexplicable: SSD ties the density of the scale space sampling to the number of feature maps used for prediction. Specifically, if there are N feature maps used for prediction the scale s_k associated with feature map k is given by:

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{N - 1}(k - 1) \qquad k \in \{1 \dots N\} \tag{5.3}$$

where s_{min} and s_{max} refer to the smallest and largest anchor scale, respectively. Therefore, anchor scales are linearly spaced between s_{min} and s_{max} and have a direct assignment to a specific feature map. However, this hard link between feature maps and anchor scales is softened by introducing another intermediate scale $\tilde{s}_k = \sqrt{s_k s_{k+1}}$. However, this is only done for an aspect ratio of 1 which seems like an ad-hoc design decision.

Network Architecture The original SSD implementation by Liu et. al. [56] uses the VGG16 [72] network architecture as basis for its detection pipeline. As are many network architectures which are used for classification, VGG16 contains fully-connected layers which limit the input size of the network. Since SSD requires dense predictions across the whole image, the fully-connected layers *fc6* and *fc7* are converted to convolutional layers which we call *conv6* and *conv7*, respectively.

We have discussed how to convert a fully-connected layer to a convolutional layer in Chapter 2.2.3. However, the number of parameters and the number of output neurons in the *fc6* layer is prohibitively large. The *fc6* layer of VGG1G operates on a 7×7 feature map and has 4096 output neurons. Directly converting such a layer into a convolutional layer (using the method described in Chapter 2.2.3) would result in 4096 filters, each with a kernel size of 7×7. Since in SSD these filters need to be applied densely across the whole *conv5* feature map, this is computationally expensive.

To speed up computation, both the kernel size of the filters and the number of filters is reduced for the *conv6* layer. The number of filters is reduced to 1024 which is achieved by simply omitting filters from the converted layer. To reduce kernel size, the filter is subsampled: The 7×7 filters which result from the conversion of the *fc6* layer is subsampled to a 3×3 filter. In order to retain the receptive field of the 7×7 filter, an *à trous* [39] convolution (often also called *dilated convolution*) is used with a dilation rate of 3.

It should be noted, that the applicability of SSD is not limited to VGG16. SSD can be used with almost any network architecture that has the notion of a feature hierarchy. SSD uses the feature hierarchy of the network to handle objects of different scales: Since the resolution of the feature maps decreases with deeper layers, large objects are predicted by deep feature maps while shallow feature maps predict small objects. The original SSD implementation uses up to seven feature maps for generating predictions as shown in Figure 5.1. The precise number of feature maps can be adjusted. By default, SSD uses all feature maps between (and including) *conv4* and *conv9*.

A *classification module* is attached to every feature map used for prediction. This classification module consists of a normalization layer and two 1×1 convolutions which are responsible for classification and bounding box regression. The

normalization layer performs a simple lateral L_2-normalization across the channels. The normalized values $\mathcal{N}_{n,c,y,x}$ are obtained from the feature map values $\mathcal{F}_{n,c,y,x}$ as follows:

$$\mathcal{N}_{n,c,y,x} = \mathcal{F}_{n,c,y,x} \frac{\gamma}{\sqrt{\sum_c \mathcal{F}^2_{n,c,y,x}}} \tag{5.4}$$

where γ is a user-specified parameter which determines the L_2-norm to which the vector should be scaled. Liu et. al. set this parameter to $\gamma = 20$.

Outputs for classification scores and bounding box regression are computed from the underlying normalized feature map using a 1×1 convolution for each task. If the feature map $\mathcal{N}$ has height $H_\mathcal{N}$ and width $W_\mathcal{N}$ the output size for the classfication task is $H_\mathcal{N} \times W_\mathcal{N} \times (|\mathcal{S}_\mathcal{N} \times \mathcal{A}_\mathcal{N} \times \mathcal{C}_+|)$. Here, $\mathcal{C}_+$ represents the extended set of object classes. This set includes the object classes $\mathcal{C}$ and the *background* class. $\mathcal{S}_\mathcal{N}$ represents the set of anchor scales, and $\mathcal{A}_\mathcal{N}$ represents the set of aspect ratios which are predicted at the normalized feature map $\mathcal{N}$.

To convert the classification output into a probability distribution over the object classes and the *background* class, a softmax function is used. Similar to RPNs, the output needs to be reshaped before applying the softmax. The classification output is reshaped from $H_\mathcal{N} \times W_\mathcal{N} \times (|\mathcal{S}_\mathcal{N} \times \mathcal{A}_\mathcal{N} \times \mathcal{C}_+|)$ to $H_\mathcal{N} \times W_\mathcal{N} \times |\mathcal{S}_\mathcal{N}| \times |\mathcal{A}_\mathcal{N}| \times |\mathcal{C}_+|$. The softmax is then performed across the last dimension, which contains the classification outputs of the individual classes. Note that this means, that individual anchor boxes do not need to compete with each other. Should two anchor boxes with approximately the same scale, shape and location produce high confidence detections, the non-maximum suppression will eliminate all responses except the highest-scoring one.

The bounding box regression is anchor-specific but not class-specific. Therefore the output size for the regression task is $H_\mathcal{N} \times W_\mathcal{N} \times 4|\mathcal{S}_\mathcal{N} \times \mathcal{A}_\mathcal{N}|$.

Training For training, anchor boxes are divided into positive and negative samples. Each anchor box with an $IoU \geq 0.5$ is considered a positive example. Positive examples are always used for training while negative examples are sampled through hard negative mining. Negative examples are always sampled deterministically based on their classification score without any random component. The ratio of positive to negative examples is $1 : 3$.

The loss function L is designed very similar to RPNs and is composed of two objectives: The classification loss L_{cls} and the localization loss L_{loc}. L_{cls} is responsible for assigning the correct class labels to an anchor box and is realized as a softmax cross-entropy loss. L_{loc} is responsible for the bounding box regression and is realized as a Huber [41] loss.

$$L = L_{cls} + \lambda L_{loc} \tag{5.5}$$

where λ controls the relative importance of L_{cls} and L_{loc}. The original SSD implementation both objectives are weighted equally and therefore sets this parameter to $\lambda = 1$.

To introduce these loss functions more formally, we abbreviate the total number of anchor boxes for a feature map with N. Therefore, the output shape for the classification scores can be written as $N \times (|\mathcal{C}| + 1)$. The output shape for the bounding box regression can be written as $N \times 4$. Out of these N anchor boxes, only a few samples $\mathcal{M} \subset \{1 \dots N\}$ are used for training. Depending on whether an example is active or not, it is assigned a loss weight of $\alpha_i = 1$ if $i \in \mathcal{M}$ and a loss weight of $\alpha_i = 0$ otherwise. The softmax produces a probability distribution $\boldsymbol{p}_i$ over the object classes and the *background* class for each anchor box i. The classification loss can, therefore, be written as

$$L_{cls} = \frac{1}{|\mathcal{M}|} \sum_{i=1}^{N} \alpha_i H(\boldsymbol{p}_i, \hat{\boldsymbol{p}}_i) \tag{5.6}$$

where $H(\boldsymbol{p}_i, \hat{\boldsymbol{p}}_i)$ is the cross-entropy loss between the predicted probability distribution $\boldsymbol{p}_i$ and the target probability distribution $\hat{\boldsymbol{p}}_i$. The target probability for the correct class is 1 while it is 0 for all other classes. It is possible for $\mathcal{M} = \emptyset$. In such a case L_{cls} is set to zero.

For the localization loss, we define $\mathcal{P} \subset \{1 \dots N\}$ as the set of positive examples. β_i is an indicator variable which is set to $\beta_i = 1$ if $i \in \mathcal{P}$ and $\beta_i = 0$ otherwise. For every positive anchor box, the bounding box regression predicts four deformation terms which we write as $\boldsymbol{r} = (r_1, r_2, r_3, r_4)$. These deformation terms describe the offset between anchor box and groundtruth item in x and y direction and scale factors for height and width. The parametrization of these terms is identical to RPNs (see Chapter 3.5.1). Since positive examples are always used for training, the localization loss can be expressed as

$$L_{loc} = \frac{1}{4|\mathcal{P}|} \sum_{i=1}^{N} \sum_{j=1}^{4} \beta_i L_{hub}(r_j, \hat{r}_j) \tag{5.7}$$

where $L_{hub}(r_j, \hat{r}_j)$ is the Huber [41] loss between the predicted regression values r_j and the target values $\hat{r}_j$.

5.2 Improving SSD for Company Logo Detection

5.2.1 Analyzing SSD

Liu et. al. [56] report, that SSD is able to achieve superior detection performance compared to Faster R-CNN [66]. However, SSD takes some design decisions which seem sub-optimal for the task of company logo detection.

Fixed input size

One obvious problem with SSD is its limitation to fixed input images: Images which exceed these input dimensions need to be resized appropriately. From our experiments with Fast R-CNN (see Chapter 4.2.3), we know that Fast(er) R-CNN has trouble to detect small object instances reliably. It is reasonable to assume that SSD suffers from similar problems. Another potential problem are images with extreme aspect ratios. The resize operation might cause strong deformations for object instances.

Anchor grid

In the original SSD implementation, the number of feature maps used for predictions defines the density of the scale space sampling of the anchor grid. Based on a user-defined minimum and maximum scale, the scales are spaced linearly across the different feature maps. There are a few potential problems with this approach:

Linear vs. exponential scale space sampling One potential problem is the linear spacing of the anchor scales itself. This is a problem for two reasons:

(1) In Chapter 3.5.3, we have analyzed the RPN anchor grid. We have shown in order not to miss any objects, neighboring anchor scales cannot be spaced apart by more than a factor of 2. We have also determined through experiment that a factor of 2 is insufficiently dense for small objects and that a factor of $\sqrt{2}$ is a more appropriate choice. However, regardless of the choice of scale factor between neighboring anchor scales, the anchor grid scales follow an exponential pattern. This is fundamentally incompatible with the linear spacing of SSD. If the large anchor scales are sufficiently dense sampled, it inevitably means that the small scales are sampled insufficiently dense and are therefore underrepresented.

(2) Another problem is that the assignment of feature maps to anchor scales is arbitrary. In its default configuration, the user-specified mininum and maximum scales are assigned to the *conv4* and *conv9* layer, respectively. While the dataset itself dictates the minimum and maximum scales, there is no justification why the *conv4* or *conv9* feature map is the optimal – or even a good – choice.

Aspect ratio sampling Another obvious problem in the original SSD implementation is that not all anchor aspect ratios are predicted for every scale. This is an odd design choice which can only be explained if the original authors deliberately tried to keep the number of anchor boxes small. Since for most datasets, the majority of object instances have an aspect ratio of $1:1$, this decision makes some sense. However, if we make the (reasonable) assumption that the distribution of aspect ratios remains fixed across multiple scales, this decision seems sub-optimal.

5.2.2 Addressing the Weaknesses

In the previous section, we have identified a few design decisions in the original SSD framework which might not be optimal for company logo detection. Now, we want to discuss how to address these shortcomings.

One of the problems we have observed is the fixed input image size of the original SSD implementation. Since all images are resized to a standardized size, it effectively means that the scales of the anchor grid are defined relative to the image size. However, it is comparatively easy to address this problem. Since SSD is a fully-convolutional network architecture, there is no technical reason why a fixed-size input should be necessary. Most likely, considerations about the detection speed have contributed to the original design decision. In a fully-convolutional architecture, a larger input image automatically results in a larger output feature map. The only change that needs to be made is to the anchor grid. In contrast to the original SSD pipeline, the anchor grid cannot be static anymore. Instead, the size of the anchor grid needs to be adapted to the size of the feature map. By addressing the problem in this way, we effectively make our anchor scales absolute.

Another problem that is easily addressed is the problem of aspect ratio sampling. We can simply add the additional aspect ratios to all scales.

The only problems that remain to be addressed are the problems related to anchor grid scales. Here, we can apply the results from our previous investigations and our investigation into the role of the receptive field for classification performance. We use our results from our analysis of the RPN anchor grid (see Chapter 3.5.4) and drop the linear scale space sampling in favor of the scale space sampling scheme we used for our improved RPN. In this scheme, the side length of neighboring scales are separated by a factor of $\sqrt{2}$. We use the results from our investigation into the role of the receptive field for classification performance to remove the arbitrary assignment of scales to feature maps: We assign anchor scales to the feature map whose receptive field can fully cover the scale of the anchor with the least amount of slack.

We can see that our experimental results from analyzing the proposal and classification stage transfer very naturally into the SSD framework. In the following, we will describe the details of our implementation of the SSD framework and show the benefits of this design.

5.2.3 Implementation Details

Network architecture

Our SSD implementation greatly resembles SSD in its network architecture. We use our batch-normalized VGG16 (Chapter 2.2.1) as a base network which is converted into a fully-convolutional version. However, we perform the conversion to a fully-convolutional network in a different way: The original SSD implementation aims to preserve the fully-connected layers of the pre-trained VGG16 network. They achieve

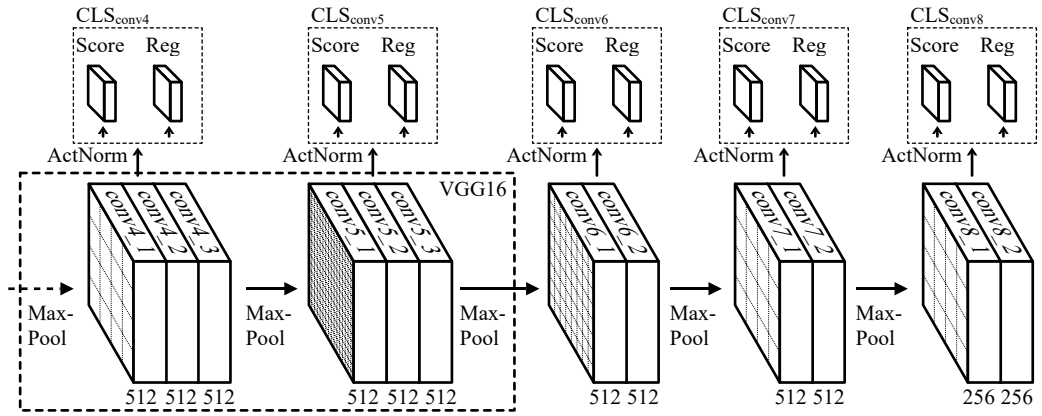


Figure 5.2: Network architecture of our SSD implementation which has been adopted for company logo detection. The overall structure is identical to the original SSD implementation. Unlike the original implementation *conv4* is our highest-resolution feature map used for predictions. Also, we do not require as many layers since the *conv8* layer already has a receptive field of sufficient size. Unlike the original SSD implementation we discard the *fc6* and *fc7* of the pre-trained network. These layers are replaced by the *conv6_1* and *conv6_2* layers which are trained from scratch.

this by discarding filters and by downsampling the remaining ones. This is such a radical modification of the layer that it is not clear that the pre-trained weights are of much use in this new configuration. Training a new layer from scratch might be just as effective. Our network architecture does exactly that: We simply choose to discard the *fc6*, *fc7* and *fc8* layers and replace them with convolutional layers that are learned from scratch. This also allows us to do away with *à trous* convolutions because we are not bound by the requirement that the new convolutional layers have the same receptive field as the old layers. Figure 5.2 gives an overview of our adapted network architecture.

Anchor assignment

We introduce a new anchor set $\mathcal{A}_{ssd}$ which follows the same scale space sampling scheme as for our improved RPN. Neighboring scales are roughly separated by a factor of $\sqrt{2}$. Also, we introduce the additional aspect ratios $\frac{1}{3}$ and 3 to better cover text-based company logos which tend to have more extreme aspect ratios. However, we omit these additional aspect ratios for the smallest and largest scales. Table 5.1 shows the exact specification of this anchor set. The reason for omitting these more extreme aspect ratios for larger scales is the small number of training examples: Large objects are comparatively rare in the FlickrLogos-47 dataset and even more so large objects with extreme aspect ratios. For small objects, the difference in aspect ratio only leads to a minor difference in the absolute width/height of anchor boxes. These differences in side lengths are small compared to the variation in

Feature map		Anchors				
Name	Rec. Field	Scale	ARs	Scale	ARs	Max side length
conv4	$92px$	$35px$	$\frac{1}{2}, 1, 2$	$50px$	$\frac{1}{2}, 1, 2$	$71px$
conv5	$196px$	$70px$	$\frac{1}{2}, 1, 2$	$100px$	$\frac{1}{2}, 1, 2$	$141px$
conv6	$340px$	$141px$	$\frac{1}{3}, \frac{1}{2}, 1, 2, 3$	$200px$	$\frac{1}{2}, 1, 2$	$346px$
conv7	$628px$	$200px$	$\frac{1}{3}, 3$	$282px$	$\frac{1}{3}, \frac{1}{2}, 1, 2, 3$	$489px$
conv8	$948px$	$400px$	$\frac{1}{3}, \frac{1}{2}, 1, 2, 3$	$565px$	$\frac{1}{2}, 1, 2$	$800px$

Table 5.1: Scales and aspect ratios of the $\mathcal{A}_{ssd}$ anchor boxes and their assignment to the individual feature maps. Anchors are assigned to the feature map whose receptive field can contain the side length of the anchor box with the least amount of slack space.

annotations. Objects instances do not have pixel-accurate annotations, and it is not always clear where the boundary of an object instance should be. For these reasons we refrain from adding more anchor boxes to small scales.

Similar to the original SSD implementation, we use the feature hierarchy of the network to predict anchors of different scales. The network consists of multiple prediction branches which operate on different feature maps. Unlike the original implementation, we assign anchors to these branches based on the receptive field of the underlying feature map. Each anchor box is assigned to the feature map whose receptive field can contain the side length of the anchor with the least amount of slack. Table 5.1 shows the details of this assignment.

Training protocol

We initialize the weights of all newly added layers using MSRA [35] initialization and the biases with zeros. We train our network for $100,000$ iterations using an initial learning rate of $\lambda = 0.001$ and perform a learning rate decay by a factor of 10 after $80,000$ iterations. Because we have introduced many new randomly initialized layers, the gradients can have a large magnitude during the first few iterations. To protect the pre-trained weights during these first iterations, we employ gradient clipping. After approximately 500 iterations, the newly added layers have adjusted sufficiently and the magnitude of the gradient hardly ever triggers the gradient clipping any more.

In every iteration, we process 3 images from the training dataset at once which is the maximum number of images we can fit into GPU memory at the same time. For each of these images, we perform data augmentation which we will discuss shortly. Since our SSD implementation does not use fixed-size images, the sampled images may differ in size. Therefore, we zero-pad the sampled images to the same size as the largest image in the batch. We then generate an anchor grid which completely covers the padded images for every feature map that is used for prediction.

(a) (b)

Figure 5.3: Two examples of training images before (left) and after data augmentation (right). Images are rotated, flipped vertically and adjusted in scale to supply enough training data for all loss function and to ensure enough variation.

Our matching strategy for object instances to anchors is simple: For every anchor, we compute the object instance which overlaps most strongly with this anchor. If the maximum overlap with an object instance is ≥ 0.5, we use this anchor as a positive example for the corresponding object class. Should no object instance overlap with this anchor with an $IoU \geq 0.5$, we assign this anchor to the *background* class. In the following, we will also refer to these anchors as *negative examples*.

Not all anchors are used for training during a single iteration. Each branch of the network has its own loss functions, one for classification and one for bounding box regression. All loss function are equally weighted. For every branch, we always use all positive examples for training, while negative examples are sampled in a ratio of $1:1$. The sampling of negative examples has no random component. We follow the original SSD implementation in purely sampling hard negative examples. For the bounding box regression task, all positive examples are always used for training.

Data augmentation

Our SSD network architecture has multiple loss functions operating on five separate branches of the network. Each branch is responsible for objects of a certain scale. If we were to assume our object instances to be uniformly distributed across all scales, it would mean that each loss function only receives a fifth of all training examples. Since the object scales do not follow a uniform distribution, the problem is even more noticeable. Therefore, data augmentation is absolutely critical to counter this problem.

We perform data augmentation in the following way: After an image is drawn from the training set, we randomly perform a flip around the vertical axis with a probability of 0.5. We then randomly rotate it in the interval $[-20°, 20°]$. The interval of allowed rotations is very limited because we use bounding boxes as object annotations: These bounding boxes are always aligned with the image axes. However, the transformed bounding boxes are usually not aligned with the image axes anymore. Therefore, we generate axes-aligned bounding boxes which circumscribe the transformed bounding boxes. As a result, bounding boxes which were tight

before the augmentation are not necessarily tight after applying the rotation. This blurs the meaning of concepts like an anchor box having an $IoU \geq 0.5$ with an object instance. Therefore, extreme rotations should not be allowed.

Finally, we a select a random object instance from the image and pick a target scale s_t in the interval $s_t \in [s_{min}, s_{max}]$ for this object at random. s_{min} and s_{max} are the scales of the smallest and largest anchor box, respectively. However, some restrictions apply: (1) When selecting an object, we prefer to pick non-difficult and non-truncated objects, provided there is a choice. (2) While we always allow an object instance to be downsampled, we do not allow upsampling beyond a scale factor of 4. This restriction protects the network branches which are responsible for large anchor boxes from overly degraded training images.

Especially the augmentation in object size is crucial for successfully training the network. However, there is still one more problem to overcome: It is possible that we select an object instance which is both small with respect to our anchor scales and small with respect to the image. In such a case, the image can be upscaled by a maximum factor of 4. Naively scaling the complete images by such a factor poses problems for our limited GPU memory. Instead, we perform the following transform $\boldsymbol{T}$ (given in homogeneous coordinates) on the image:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & \min(\frac{s_{max}}{2}, \frac{s_t}{s_c}\frac{l_\mathcal{I}}{2}) \\ 0 & 1 & \min(\frac{s_{max}}{2}, \frac{s_t}{s_c}\frac{l_\mathcal{I}}{2}) \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{s_t}{s_c} & 0 & 0 \\ 0 & \frac{s_t}{s_c} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ 0 & 0 & 1 \end{pmatrix} \tag{5.8}$$

p_x and p_y denotes the position of the selected object instance in the original image (measured from the object center). The current scale of the selected object instance is given by s_c while the picked target scale is given by s_t. $l_\mathcal{I} = \max(w_\mathcal{I}, h_\mathcal{I})$ represents the longer side of the input image $\mathcal{I}$ which has width $w_\mathcal{I}$ and height $h_\mathcal{I}$.

We limit the image size after applying $\boldsymbol{T}$ to $s_{max} \times s_{max}$ by cropping oversized images. Informally speaking, this transform scales the image by a factor of $\frac{s_t}{s_c}$ and centers the selected object instance at the center of the new image. The translation operation which centers the object instance distinguishes between the case in which the scaled image exceeds a side length of s_{max} and the case in which it does not. Since many images in the FlickrLogos-47 dataset already exceed $s_{max} \times s_{max}$ even without any augmentation, most images fall into the first case and get cropped during their augmentation. Figure 5.3 shows some examples of augmented image patches.

Although this augmentation scheme means that the network will always receive images in which the objects appear in the center of the images, the fully-convolutional nature of the network provides translation invariance. Training such a network can best be understood as training a set of highly non-linear filters, each of which is responsible for detecting objects at a particular scale. At test time, these filters are then simply applied at every location in the image.

Predicting object instances

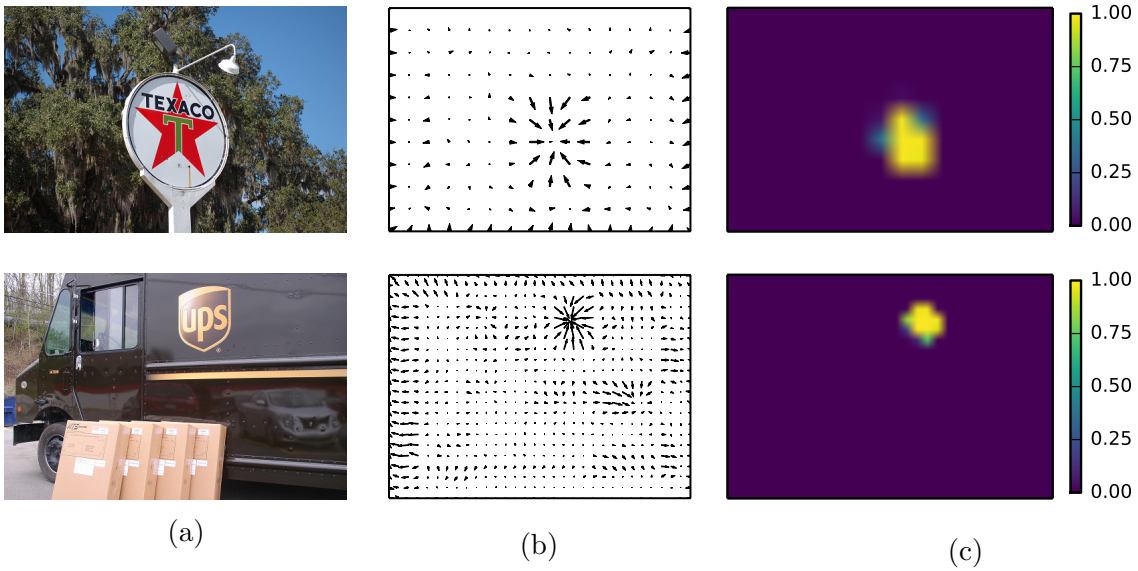
Unlike the original SSD implementation, we do not restrict ourselves to $512px \times 512px$ images. We can take advantage of the fully-convolutional nature of the network and resize it to fit any input image size. Our network produces probabilities for class membership and bounding box regression values for all anchor for every feature map. Figure 5.4 shows some examples of the raw network output. These predictions need to be converted into detections. Each detection consists of a class label, a confidence score and a location which corresponds to an anchor box of the anchor grid that has undergone bounding box regression. To speed up the post-processing, we try to avoid enumerating the detections for all layers. We perform a fast test, by determining the maximum value of non-*background* detection confidence $c_{max} \in]0.0, 1.0[$. This test is conducted directly on the network output, before its conversion into detections. We exclude features maps from further processing if c_{max} does not exceed a certain threshold t. We choose $t = 0.01$.

For every feature map that is processed, we convert all predictions into detections. We discard all detections whose confidence is lower than t and perform a non-maximum suppression in two steps: (1) Since a non-maximum suppression has a worst-case runtime of $\mathcal{O}(N^2)$ with N being the number of bounding boxes, we try to keep N small. Therefore we perform a feature map specific non-maximum suppression to prune our initial detections. (2) The pruned detections from every feature map are merged, and another non-maximum suppression is executed on the merged detections. All non-maximum suppression steps are class-specific, meaning that we only suppress bounding boxes of the same class.

5.2.4 Evaluation

We first perform a direct comparison between the performance of the original SSD implementation and our implementation. To do that, we evaluate the original SSD implementation on the FlickrLogos-47 dataset. The original authors [56] have introduced *SSD300* and *SSD512* which uses a fixed input size of $300px \times 300px$ and $512px \times 512px$, respectively. In its default configuration, SSD uses the *conv4* to *conv9* layers for predictions. We leave the network definition and the anchor assignment to the feature maps of the original implementation unchanged. However, we do adjust the minimum and maximum anchor scales to match the object sizes of the FlickrLogos-47 dataset better. For a fair comparison, we want to choose the minimum and maximum anchor scales to match the anchor scales of the $\mathcal{A}_{ssd}$ anchor set.

While this sounds like a straightforward task, it is not as simple to achieve as it might look on first glance and again reflects one of the weaknesses of the original implementation: Since all images are scaled to a fixed input size, all anchor scales are defined relative to the input size of the network. The anchor scale that is responsible for detecting an object of a given size is therefore not only dependent on the size of



(a) (b) (c)

Figure 5.4: Visualization of the detection maps and bounding box regression of SSD. (a) Input image. (b) Predicted translation component of the box regression. The vector field indicates the predicted offsets for a single anchor at every location in the image. (c) Detection map which visualizes the probability of a single anchor belonging to the correct object class for every location in the image.

the object itself, but also on the size of the original image. In order to approximate a fair comparison we assume a typical image in the FlickrLogos-47 to have size of $1024px \times 1024px$. This assumption is not completely accurate since the typical image has a size of $1024px \times 768px$. However, quadratic input images simplify the following considerations, which is why we use $1024px \times 1024px$ as a basis for our calculations. For the example of *SSD512*, this means that – on average – each image is downscaled by a factor of 2. For a minimum and maximum anchor scale of $35px$ and $565px$, this corresponds to anchor scales of approximately $17px$ and $282px$. Therefore, we configure the original *SSD512* implementation to use $17px$ and $282px$ as minimum and maximum anchor scale. For *SSD300* we adapt the anchor scales accordingly.

In order to get the original SSD implementation to converge, we had to add gradient clipping to protect the pre-trained network weights during the first iterations. Without gradient clipping, training diverged after a few iterations. Table 5.2 shows a comparison between the detection results of *SSD300*, *SSD512* and our SSD implementation.

Unsurprisingly, *SSD300* yields the lowest mAP score of all three approaches. However, it performs remarkably well if we consider the small size of the input images. *SSD512* does perform better than *SSD300*, and its detection performance is approximately on par with the performance we measured for R-CNN while exhibiting a runtime which is orders of magnitude lower than R-CNN's runtime. Interestingly, the runtimes of *SSD512* and *SSD300* do not differ by much, even though the areas

Impl.	Input size	Anchor Set	Scales min/max	F. maps min/max	Assignment	mAP	Runtime / img
orig.	$300px$	orig.	$35px^*$ $565px^*$	*conv4* *conv9*	linear$^+$	0.703	$0.03s$
orig.	$512px$	orig.	$35px^*$ $565px^*$	*conv4* *conv9*	linear$^+$	0.737	$0.03s$
ours	—	$\mathcal{A}_{ssd}$	$35px$ $565px$	*conv4* *conv8*	rec. field	**0.773**	$0.08s$

Table 5.2: Comparison between the original SSD implementation [56] and our implementation. ($^+$) Note that the anchor grids are not directly comparable since in the original implementation the density of the scale space sampling is tied to the feature maps. (*) The object scales for the original SSD implementation are estimates based on the assumption of a $1024px \times 1024px$ image (see text for details).

of the input images differ almost by a factor of 3. This is because for such small input sizes, applying the network to the images is not the bottleneck. Instead, data preparation and the data transfer between host and GPU memory dominate the execution time. Since our implementation does not rescale the images and the images tend to be quite large, it is hard to beat the baseline set by the original SSD approach. However, we are still able to beat the runtime of all our previous approaches while outperforming them with regard to detection performance. We are also able to outperform the original SSD implementation, even though we are using a smaller network and are using fewer feature maps for prediction.

Unfortunately, it is hard to achieve a side-by-side comparison between the original SSD implementation and our implementation that is fair in all aspects. Any direct comparison between the two approaches would have to use the same anchor grid. The direct link between network structure and the assignment of anchor scales to feature maps is fundamentally incompatible with the assignment of anchor scales based on the receptive field: Our network structure has to be much less deep than the original SSD implementation because the *conv8* feature map already has a receptive field of $948px$ which is more than enough to cover the maximum scale in our anchor set. On the other hand, the network depth in the original SSD implementation directly influences the density of the scale space sampling.

If we were to use the same linear assignment scheme as the original SSD implementation in our shortened network, it would result in less densely sampled anchor scales which puts our implementation at a disadvantage in a direct comparison. Such a network configuration would result in the smallest anchor scales to be spaced apart by more than a factor of 2. Since we have shown in Chapter 3.5.3 that a factor of 2 is the theoretical limit if we require an IoU ≥ 0.5 for a correct detection, this experiment is unlikely to produce satisfactory results. Nevertheless, we have performed

Dataset	ours	Fast-M [4]	DeepLogo [42]	SCL [73]	BD-FRCN-M [61]
FlickrLogos-32	0.837	**0.842**	0.744	0.811	0.735

Table 5.3: Comparison of our SSD implementation to other approaches to company logo detection. All values are mAP on the respective datasets. Since *FlickrLogos-32* is a retrieval dataset, the evaluation protocols the authors used for detection are often unclear. Therefore, a comparison between these values should be viewed with scepticism.

this experiment which yields a mAP of 0.720 which is surprisingly good for such an ill-defined detection problem.

We also cannot use our $\mathcal{A}_{ssd}$ anchor set in the original SSD implementation because our exponential distribution of anchor scales is incompatible with the linear distribution of anchor scales to feature maps which puts the original SSD implementation at an unfair disadvantage.

5.2.5 Comparison to other approaches in the literature

We want to compare our re-implemented SSD to other company logo detection approaches in the literature. Unfortunately, we are not aware of any baselines on the *FlickrLogos-47* dataset. We therefore use the *FlickrLogos-32* [68] dataset which is often used in the literature for the evaluation of company logo detection pipelines.

We compare our approach to four other approaches: (1) DeepLogo [42] by Iandola et al. was among the first to apply an object detector built on Fast R-CNN to company logo detection. (2) BD-FRCN-M [61] by Oliveira et al. is another Fast R-CNN based approach which employs data augmentation. (3) SCL [73] by Su et al. used a Faster R-CNN based detector on FlickrLogos-32 whose training set has been expanded with synthetically generated training images. (4) Fast-M [4] by Bao et. al. is a multi-scale approach to Fast R-CNN.

All of the papers mentioned above evaluate their approaches on the FlickrLogos-32 dataset. Although *FlickrLogos-32* has object-level annotations, it has been designed as an image retrieval dataset. It, therefore, does not offer an official evaluation script for object detection. We use our evaluation script for *FlickrLogos-47* for this task. Confusingly, *FlickrLogos-32* does offer an evaluation script for a task that it calls *object detection*. However, this task has nothing to do with the task of object detection as commonly used in the literature. The task that *FlickrLogos-32* calls *object detection* is more commonly known as *image classification*: Images which may or may not contain company logos have to be classified according to their content. No localization is required.

These complications make it difficult to interpret the results reported by the other authors. One notable exception are Iandola et. al. [42] who evaluate their

approach on *FlickrLogos-32* and make a clear distinction between non-localized and localized detections.

In Table 5.3 we compare our best-performing approach (our re-implemented SSD detector) to the above-mentioned approaches to object detection on the FlickrLogos-32 dataset. Our approach is able to beat all other approaches, except Fast-M [4]. Fast-M is a multi-scale Fast R-CNN based-approach that exploits the same observation that we made earlier, that upscaling the images improves the detection performance. Fast-M goes even further and not only re-scales the images once but five times. Although no timing results are reported for Fast-M, it is safe to assume that the runtime is not remotely comparable to our SSD-based approach.

5.3 Conclusion

In this chapter, we have considered single-stage object detection approaches for the problem of company logo detection. There are two major approaches to single-stage object detection: YOLO [65] and the Single Shot MultiBox Detector (SSD) [56]. Contrary to SSD, the fully-connected layers limit YOLO for fixed-size input images on a technical level. Since even the authors admit that YOLO struggles with small object instances, we have excluded YOLO from our evaluation.

Instead, we have focused our attention on the Single Shot MultiBox Detector. Although the original SSD implementation also uses fixed-size input images, it is not a technical limitation but a design choice. We have evaluated the original SSD implementation on the FlickrLogos-47 dataset and found its detection performance to be competitive with R-CNN but with a runtime that is orders of magnitude faster. We have analyzed SSD and have identified three potential areas for improvement: (1) Because of fixed-size input images, the anchor scales are always defined relative to the image size. (2) We have identified the linear scale space sampling as incompatible with a feature hierarchy of exponentially decreasing resolution. (3) The arbitrary assignment of anchor scales to feature maps.

We have re-written SSD to account for these shortcomings and have evaluated our approach. We were able to show that our re-formulated SSD framework is able to outperform all previously evaluated approaches both in terms of detection performance and in runtime.

Part IV

Conclusion

Chapter 6

Conclusions

6.1 Conclusions

In this work, we have applied three well-known object detection approaches to the task of company logo detection. We have evaluated both two-staged and single-staged approaches to object detection. For the two-staged object detection approaches, we have analyzed in detail both the proposal stage and the classification stage in the context of their ability to retrieve small object instances.

For our first contribution, we have looked at heuristic approaches to object proposals. In particular, we have looked at two popular algorithms, Selective Search and Edge Boxes. We have performed a detailed analysis of their respective strengths and weaknesses and found that both approaches are able to complement each other. Also, we have examined common failure modes of Selective Search on text-based company logos. To address these problems, we have introduced the VH-connect algorithm as a new lightweight heuristic which is specifically designed to retrieve text-based instances. We were able to show that by integrating VH-connect into the object proposal stage we can increase the maximum achievable recall at virtually no additional impact on overall runtime.

We have then shifted our focus to trainable approaches to object proposals. For our second contribution, we have examined Region Proposal Networks and have noticed that its performance regarding maximum achievable recall is inferior to the heuristic approaches at which we have looked. We then have analyzed the reasons behind this performance gap and have identified insufficiently dense scale space sampling and lacking feature map resolution as two primary culprits. We were able to show that by addressing these issues, the performance gap to heuristic proposals can be closed and even surpass them in both runtime and maximum achievable recall.

With our improved object proposals, we have turned our attention to the classification stage. We have evaluated three approaches to classify our object proposals: R-CNN, Fast R-CNN, and Faster R-CNN. However, since Faster R-CNN only differs from Fast R-CNN in the source of its object proposals, only the classification stages

of R-CNN and Fast R-CNN are genuinely unique. We have found that R-CNN performs much better in classifying our proposals than the newer Fast(er) R-CNN approach. For our third contribution, we analyze the reasons behind this behavior since in the literature Fast(er) R-CNN is almost universally reported to be superior to R-CNN. Our analysis has shown that the size of the object compared to the size of the receptive field of the network plays a vital role in the classification performance. We have shown that although no new information is added to the image, simple upscaling of the input images can increase the detection performance to the point where the performance of Fast(er) R-CNN is competitive with R-CNN. Since upscaling the image is associated with increased computational costs of applying the network, we have introduced Selective Magnification which aims to only upscale the parts of the image which are likely to contain objects. We were able to show, that using Selective Magnification, we can achieve most of the performance gains at only a fraction of the runtime needed for naive upscaling.

Finally, we have looked at the Single Shot MultiBox Detector as a single-stage object detection approach which circumvents the need to generate object proposals. For our fourth contribution, we have noticed that most of the lessons we have learned for improving trainable object proposals can also be applied in the context of the Single Shot MultiBox Detector. Also, we have identified some design decisions in the original implementation which are suboptimal. In particular, we have addressed the linear scale space sampling scheme and the arbitrary assignment of anchor scales to feature maps. Our re-implemented version of the Single Shot MultiBox Detector incorporates the lessons learned from our improved approach to trainable object proposals and addresses the problems mentioned above. We have shown that our re-implemented version is able to achieve superior detection performance not only to the original implementation but also to all other approaches to company logo detection that we have evaluated in this work. At the same time, it is also the fastest approach to company logo detection that we have evaluated in this work.

6.2 Outlook

Object detection has made major advances in the last years, even to the point that have led some people to claim that object detection is a solved problem. Certainly, there is some truth to this claim: Many of the difficulties of object detection like variations in viewpoint, partial occlusion, and deformable objects have become problems of the past with the advent of deep neural networks. Object detection has become advanced enough that most everyday objects can be detected somewhat reliably.

However, we feel that there is still much room for future research. An interesting area of research is to model the relationships of objects in a scene with each other. This could potentially help to improve detections in scenarios where there is hardly any direct visual evidence for the presence of an object. While in this work we were able to make some improvements in detecting small company logos, we still rely on

the company logo being visible and (somewhat) recognizable. There are instances where the object in question only consists of a few pixels of a specific color. Even a human observer would have trouble to recognize such a company logo when only given the visual evidence of the company logo itself.

However, humans are often able to infer the presence of an object by looking at the context in which it occurs: From a distance, it may be difficult to spot a cell phone that a pedestrian is using. However, we may be able to infer the presence of a cell phone based on the pose of the pedestrian itself. Similar inferences may be possible for the task of company logo detection. For example, company logos related to beer brands are often located on bottles which in turn have a distinctive design. R-CNN cannot use these relationships because the object proposals are cut out from the image. Fast(er) R-CNN and SSD are in principle able to capture these relationships. However, it might be beneficial to model them explicitly.

Also, we feel that using bounding boxes as a means of localizing detections is increasingly outdated as expectations on object detection systems grow. This can also be seen in our FlickrLogos-47 dataset: For objects with extreme aspect ratios (such as text-based company logos), the orientation of the objects becomes important. For objects which are oriented horizontally or vertically, a bounding box may still be sufficient. However, for objects with extreme aspect ratios and arbitrary orientations, a bounding box is a poor indicator of location. Therefore, we expect that future research will increasingly go into the direction of semantic instance segmentation which does not suffer from these problems.

Appendix A

FlickrLogos-47: Object Instances

	trainval			test		
Class name	total	difficult	truncated	total	difficult	truncated
Adidas (Symbol)	37	0	1	104	2	1
Adidas (Text)	34	2	0	71	7	1
Aldi	38	2	0	88	4	1
Apple	30	0	0	47	0	0
Becks (Symbol)	52	2	0	98	6	1
Becks (Text)	54	3	2	118	24	7
BMW	29	0	0	51	0	0
Carlsberg (Symbol)	30	3	0	92	10	0
Carlsberg (Text)	40	2	2	112	16	12
Chimay (Symbol)	45	3	0	79	10	0
Chimay (Text)	56	8	2	83	6	2
CocaCola	62	4	2	91	0	0
Corona (Symbol)	32	1	0	54	1	1
Corona (Text)	35	3	1	59	1	2
DHL	51	2	1	93	6	3
Erdinger (Symbol)	48	2	1	70	1	1
Erdinger (Text)	33	2	2	50	1	1
Esso (Symbol)	32	2	2	63	0	1
Esso (Text)	8	0	0	34	6	2
FedEx	36	0	2	60	1	4
Ferrari	29	0	0	44	0	0
Ford	30	0	0	47	0	0
Fosters (Symbol)	33	2	0	99	19	4
Fosters (Text)	43	2	12	98	21	3
Google	33	0	0	50	0	0
Guinness (Symbol)	37	4	1	80	4	0
Guinness (Text)	38	1	2	103	5	2

Heineken	63	5	2	106	12	6
HP	43	1	1	75	1	1
Milka	89	14	6	275	39	14
nVidia (Symbol)	40	2	0	97	8	0
nVidia (Text)	40	2	0	92	5	0
Paulaner (Symbol)	48	3	1	69	0	1
Paulaner (Text)	30	3	4	63	8	6
Pepsi (Symbol)	57	3	1	194	18	5
Pepsi (Text)	54	8	9	140	20	6
Rittersport	87	16	10	202	32	19
Shell	34	0	0	66	4	1
Singha (Symbol)	26	0	0	56	1	1
Singha (Text)	26	0	0	57	1	1
Starbucks	43	3	3	65	5	3
Stellaartois (Symbol)	43	8	0	72	1	0
Stellaartois (Text)	33	1	4	66	5	1
Texaco	33	0	0	56	0	0
Tsingtao (Symbol)	39	0	0	91	7	0
Tsingtao (Text)	49	6	2	95	9	0
UPS	34	0	2	57	0	0
Total	1936	125	78	4032	327	114

Table A.1: Number of object instances for the re-structured trainval and test sets of the FlickrLogos-47 dataset. For each class, the trainval set typically contains around 33% of all object instances.

Appendix B

Edge Grouping

Algorithm Grouping algorithm for edges

procedure GROUP_EDGES(E, O, t_Θ)
  $E \leftarrow$ Set of pixels (x, y) which are part of an edge
  $O(x, y) \leftarrow$ Edge orientations for pixels, $O(x, y) \in [0, \pi[$
  $A \leftarrow \emptyset$ ▷ Set of pixels which have already been assigned to a group
  $G \leftarrow \emptyset$ ▷ Result set containing pixel groups
  for each pixel $(x, y) \in E$ **do**
    if $(x, y) \in A$ **then continue**
    $G_i \leftarrow \{(x, y)\}$ ▷ Create a new pixel group
    $\Theta \leftarrow 0.0$ ▷ Sum of orientation differences in this group
    $(\bar{x}, \bar{y}) \leftarrow (x, y)$ ▷ Current center of the search neighborhood
    $D \leftarrow \emptyset$ ▷ Initialize list of discovered pixels
    while $\Theta < t_\Theta$ **do** ▷ Group until orientation threshold exceeded
      $N_{\bar{x},\bar{y}} \leftarrow$ 8-neighborhood around pixel $(\bar{x}, \bar{y})$
      $N_{\bar{x},\bar{y}} \leftarrow N_{\bar{x},\bar{y}} - (A \cup \overline{E})$ ▷ Only consider edges and unassigned pixels
      $D \leftarrow D \cup N_{\bar{x},\bar{y}}$ ▷ Add edges to discovered set
      if $D = \emptyset$ **then break**
      $(\hat{x}, \hat{y}) = \arg\min_{(\tilde{x},\tilde{y}) \in D} \{\text{anglediff}(O(\bar{x}, \bar{y}), O(\tilde{x}, \tilde{y}))\}$
      $G_i \leftarrow G_i \cup \{(\hat{x}, \hat{y})\}$ ▷ Add edge w. most similar orientation to group
      $\Theta \leftarrow \Theta + \text{anglediff}(O(\bar{x}, \bar{y}), O(\hat{x}, \hat{y}))$
      $A \leftarrow A \cup (\hat{x}, \hat{y})$
      $(\bar{x}, \bar{y}) \leftarrow (\hat{x}, \hat{y})$
    $G \leftarrow G \cup G_i$
  return G
procedure ANGLEDIFF(O_1, O_2)
  $d \leftarrow |O_1 - O_2|$
  if $d > \frac{\pi}{2}$ **then**
    $d \leftarrow \pi - d$
  return d

Appendix C

Sigmoid or Softmax

During our introduction of Region Proposal Networks (see Chapter 3.5.1) we briefly mentioned that the task of anchor prediction can be modelled through a Softmax cross-entropy loss or a Sigmoid cross-entropy loss. Here, we want to explicate the relationship between the two models in greater detail.

Anchor prediction in Region Proposal Networks (RPNs) is an example of a binary classification problem. On an abstract level, a binary classification problem tries to categorize input data into two classes $\mathcal{C} = \{c_0, c_1\}$. More formally, binary classification involves predicting the probabilities $p(c_0|\boldsymbol{x})$ and $p(c_1|\boldsymbol{x})$ of a data point $\boldsymbol{x}$ belonging to classes c_0 and c_0, respectively.

Since we have a binary classification problem, it suffices to predict $p(c_0|\boldsymbol{x})$ because $p(c_1|\boldsymbol{x}) = 1 - p(c_0|\boldsymbol{x})$. This is the choice we face when modelling such a problem in a neural network: We can choose to represent the problem as a network with a single output neuron which predicts $p(c_0|\boldsymbol{x})$ or we can represent the problem with two output neurons, each of which predict $p(c_0|\boldsymbol{x})$ and $p(c_1|\boldsymbol{x})$. For brevity, we will write the predicted confidence values $p(c_0|\boldsymbol{x})$ and $p(c_1|\boldsymbol{x})$ as p_0 and p_1.

In order to interpret the activation of a network with a single output neuron y_0 as probability, its activation needs to be squashed into the interval $p_0 \in]0, 1[$. This is usually achived through a sigmoid function. The logistic function $L(y_0)$ is the most frequently used sigmoid function – so frequently, that the terms *sigmoid function* and *logistic function* are often used interchangeably in the literature.

$$L(y_0) = \frac{1}{1 + e^{-y_0}} = \frac{e^{y_0}}{1 + e^{y_0}} \tag{C.1}$$

The logistic function has an important symmetry property which we will exploit shortly:

$$1 - L(y_0) = 1 - \frac{e^{y_0}}{1 + e^{y_0}} = \frac{1}{1 + e^{y_0}} = L(-y_0) \tag{C.2}$$

For networks which model binary classification problems with two output neurons $\boldsymbol{y} = (y_0, y_1)$, the softmax function $\boldsymbol{S}(\boldsymbol{y})$ is used to interpret the network activations as probability distribution.

$$\boldsymbol{S}(\boldsymbol{y})_i = \frac{e^{y_i}}{\sum\limits_{j \in \{0,1\}} e^{y_j}} = \frac{e^{y_i}}{e^{y_0} + e^{y_1}} \tag{C.3}$$

To show the relationship between a 2-class softmax and the sigmoid function, we first need to introduce an important property of the softmax function: If we add a constant term c to all activations, the probability distribution induced by the softmax does not change.

$$\boldsymbol{S}(\boldsymbol{y})_i = \frac{e^{y_i + c}}{\sum_{j=0}^{|C|-1} e^{y_j + c}} = \frac{e^{y_i} e^{c}}{\sum_{j=0}^{|C|-1} e^{y_j} e^{c}} = \frac{e^{y_i}}{\sum_{j=0}^{|C|-1} e^{y_j}} \tag{C.4}$$

For a binary classification problem, this property also means, that we can always shift our activations in such a way that one of our activations is set to zero. We can use this fact to transform our softmax into an equivalent sigmoid function by choosing $c = -y_1$.

$$\begin{aligned} \boldsymbol{S}(\boldsymbol{y})_0 &= \frac{e^{y_0}}{e^{y_0} + e^{y_1}} = \frac{e^{y_0 - y_1}}{e^{y_0 - y_1} + e^{y_1 - y_1}} = \frac{e^{y_0 - y_1}}{1 + e^{y_0 - y_1}} = L(y_0 - y_1) \\ \boldsymbol{S}(\boldsymbol{y})_1 &= \frac{e^{y_1}}{e^{y_0} + e^{y_1}} = \frac{e^{y_1 - y_1}}{e^{y_0 - y_1} + e^{y_1 - y_1}} = \frac{1}{1 + e^{-(y_1 - y_0)}} = L(y_1 - y_0) \end{aligned} \tag{C.5}$$

This relationship shows that it is possible to transform any network which solves a binary classification problem with two output neurons into an equivalent network with only a single output neuron without retraining. By subtracting the activations for both classes and applying a logistic function on the result instead of applying a softmax directly on the activations we can transform the network.

It is also possible to transform any network which solves a binary classification problem using a single output neuron into an equivalent network with two output neurons: In this case we need to choose a value for y_1 which for convenience should be set to zero. By applying the softmax $\boldsymbol{S}((y_0, 0)^T)$ we can obtain the same probability distribution as with a network using the logistic function.

However, the ability to construct networks which behave equivalently during evaluation does not mean that both approaches are also equivalent during training. To show this, we derive the gradients for both the sigmoid and the softmax case. The cross-entropy loss $C(\boldsymbol{p}, \hat{\boldsymbol{p}})$ is the most commonly used loss function for classification problems. It compares the predicted probabilities for every class $\hat{\boldsymbol{p}} = (\hat{p}_0, \hat{p}_1)$ against a target probability distribution $\boldsymbol{p} = (p_0, p_1)$. Since the class labels are known, the target probability for the correct class is 1 while the target probability for the

other class is 0. For a binary classification problem the cross entropy loss takes the following form:

$$C(\boldsymbol{p}, \hat{\boldsymbol{p}}) = -\sum_{k \in \{0,1\}} p_k \log \hat{p}_k = -p_0 \log \hat{p}_0 - (1 - p_0) \log(1 - \hat{p}_0) \tag{C.6}$$

Depending on whether we want to measure the cross entropy in *bits* or *nats* we use the logarithm to base 2 or base e. For convenience, we will assume a natural logarithm for the following calculations. To derive the gradient for the sigmoid cross entropy loss, we first derive the gradient for the sigmoid function itself.

$$\begin{aligned}
\frac{\partial L(y_0)}{\partial y_0} &= \frac{\partial}{\partial y_0} \frac{1}{1 + e^{-y_0}} \\
&= -\frac{1}{(1 + e^{-y_0})^2} \frac{\partial}{\partial y_0} \left(1 + e^{-y_0}\right) \\
&= \frac{e^{-y_0}}{(1 + e^{-y_0})^2} \\
&= \frac{1}{(1 + e^{-y_0})} \frac{e^{-y_0}}{(1 + e^{-y_0})} \\
&= \frac{1}{1 + e^{-y_0}} \frac{1}{1 + e^{y_0}} \\
&= L(y_0) L(-y_0)
\end{aligned} \tag{C.7}$$

We use this result to derive the gradient for the sigmoid cross entropy loss. Additionally, we make use of the symmetry property $1 - L(y_0) = L(-y_0)$ (see equation C.2).

$$\begin{aligned}
\frac{\partial C(\boldsymbol{p}, \hat{\boldsymbol{p}})}{\partial y_0} &= \frac{\partial}{\partial y_0} - p_0 \log L(y_0) - (1 - p_0) \log(1 - L(y_0)) \\
&= -p_0 \frac{1}{L(y_0)} \frac{\partial}{\partial y_0} L(y_0) - (1 - p_0) \frac{1}{1 - L(y_0)} \frac{\partial}{\partial y_0} (1 - L(y_0)) \\
&= -p_0 \frac{1}{L(y_0)} L(y_0) L(-y_0) - (1 - p_0) \frac{1}{1 - L(y_0)} \left(-L(y_0) L(-y_0)\right) \\
&= -p_0 L(-y_0) + (1 - p_0) \frac{L(y_0) L(-y_0)}{L(-y_0)} \\
&= -p_0 L(-y_0) + (1 - p_0) L(y_0) \\
&= -p_0 L(-y_0) + L(y_0) - p_0 L(y_0) \\
&= -p_0 \left(L(-y_0) + L(y_0)\right) + L(y_0) \\
&= -p_0 \left(1 - L(y_0) + L(y_0)\right) + L(y_0) \\
&= L(y_0) - p_0
\end{aligned} \tag{C.8}$$

The gradient of the sigmoid cross entropy loss is therefore simply the difference between the predicted probability $L(y_0)$ and the target probability p_0. In most cases the target probability is a binary value $p_0 \in \{0, 1\}$ since we usually assume the groundtruth class membership to be known with perfect confidence.

In order to derive the gradient for the softmax cross-entropy loss, we start by deriving the gradient for the softmax function itself. In the following, δ_{ij} represents the Kronecker delta.

$$\begin{aligned}\frac{\partial \boldsymbol{S}(\boldsymbol{y})_i}{\partial y_j} &= \frac{\partial}{\partial y_j}\frac{e^{y_i}}{\sum_{c_k \in \mathcal{C}} e^{y_k}} \\ &= \frac{1}{\sum_{c_k \in \mathcal{C}} e^{y_k}}\frac{\partial}{\partial y_j}e^{y_i} + e^{y_i}\frac{\partial}{\partial y_j}\frac{1}{\sum_{c_k \in \mathcal{C}} e^{y_k}} \\ &= \frac{1}{\sum_{c_k \in \mathcal{C}} e^{y_k}}(\delta_{ij}e^{y_i}) - \frac{e^{y_i}e^{y_j}}{\left(\sum_{c_k \in \mathcal{C}} e^{y_k}\right)^2}\end{aligned} \tag{C.9}$$

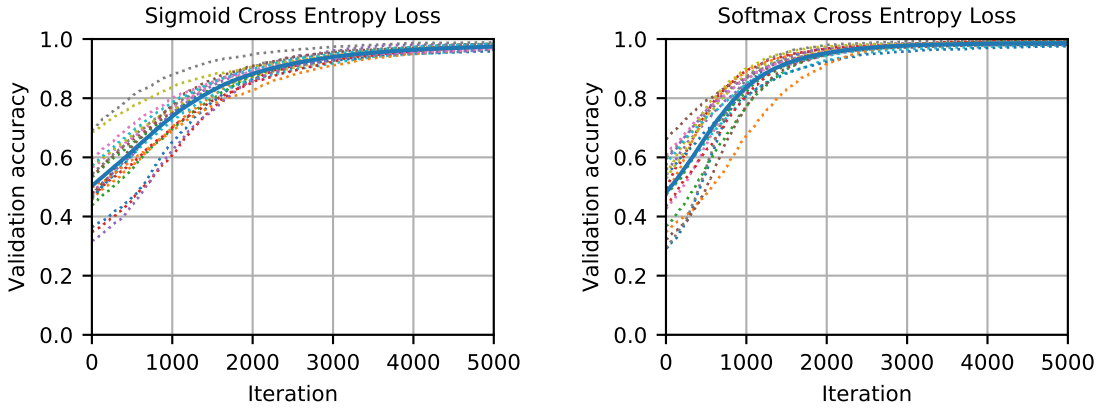
Depending on whether $i = j$ we can substitute the definition of the softmax function and obtain one of the following derivatives:

$$\frac{\partial \boldsymbol{S}(\boldsymbol{y})_i}{\partial y_j} = \begin{cases} \boldsymbol{S}(\boldsymbol{y})_i(1 - \boldsymbol{S}(\boldsymbol{y})_i) & \text{if } i = j \\ -\boldsymbol{S}(\boldsymbol{y})_i \boldsymbol{S}(\boldsymbol{y})_j & \text{if } i \neq j \end{cases} \tag{C.10}$$

Substituting the expression above into the derivative of the the cross entropy yields:

$$\begin{aligned}\frac{\partial C(\boldsymbol{p}, \boldsymbol{p})}{\partial y_j} &= \frac{\partial}{\partial y_j} - \sum_{c_k \in \mathcal{C}} p_k \log \boldsymbol{S}(\boldsymbol{y})_k \\ &= -\sum_{c_k \in \mathcal{C}} p_k \frac{1}{\boldsymbol{S}(\boldsymbol{y})_k}\frac{\partial \boldsymbol{S}(\boldsymbol{y})_k}{\partial y_j} \\ &= -p_j\left(1 - \boldsymbol{S}(\boldsymbol{y})_j\right) - \sum_{c_k \in \mathcal{C}, k \neq j} p_k \frac{-\boldsymbol{S}(\boldsymbol{y})_j \boldsymbol{S}(\boldsymbol{y})_k}{\boldsymbol{S}(\boldsymbol{y})_k} \\ &= -p_j + p_j \boldsymbol{S}(\boldsymbol{y})_j + \sum_{c_k \in \mathcal{C}, k \neq j} p_k \boldsymbol{S}(\boldsymbol{y})_j \\ &= \boldsymbol{S}(\boldsymbol{y})_j \left(p_j + \sum_{c_k \in \mathcal{C}, k \neq j} p_k\right) - p_j\end{aligned} \tag{C.11}$$

Since $\boldsymbol{p}$ represents a probability distribution it follows that $p_j + \sum_{c_k \in \mathcal{C}, k \neq j} p_k = 1$. Therefore, we obtain for the gradient of the softmax cross entropy loss:



(a) (b)

Figure C.1: Convergence speed of a linear model on 20 different linearly separable datasets using (a) sigmoid cross entropy loss and (b) softmax cross entropy loss. Each dotted line represents the validation accuracy of the model for a single training run on a dataset. The solid line represents the average validation accuracy for all training runs. On average, the softmax cross entropy loss converges faster than the sigmoid cross entropy loss.

$$\frac{\partial C(\boldsymbol{p}, \hat{\boldsymbol{p}})}{\partial y_j} = \boldsymbol{S}(\boldsymbol{y})_j - p_j \tag{C.12}$$

Therefore, the gradient for the softmax cross entropy loss looks very similar to the gradient of the sigmoid cross entropy loss. In both cases the gradient is the difference between the predicted probability and the target probability. However, there is a crucial difference between both loss functions: The sigmoid cross entropy loss can only change the strength of the output neuron activation to improve its predictions. On the other hand, the softmax cross entropy loss has two possibilities to improve the prediction: It can increase the strength of the activation for the positive class but it can also simultaneously decrease the strength of the activation for the negative class. The confidence of the classification depends on the difference between the two activations as we have shown in equation C.5.

We therefore conjecture that the softmax cross entropy loss tends to converge faster than the sigmoid cross entropy loss, even for binary classification problems. Since complicated network structures might cloud the interpretability of the results, we test our hypothesis by considering an extremely simple machine learning problem:

We generate binary classification problems whose classes $\mathcal{C} = \{c_0, c_1\}$ can be perfectly separated by a hyperplane. More specifically, we generate $d = 20$ artificial datasets in $\mathbb{R}^{10}$ containing $n = 2000$ points each. Each datapoint $\boldsymbol{x}$ can be perfectly classified by learning a linear model $\boldsymbol{H}$

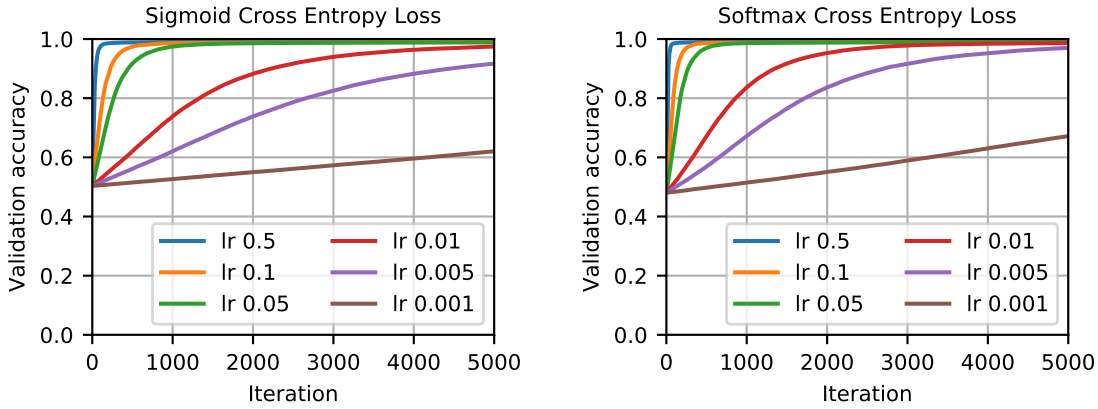


Figure C.2: Convergence speed of a linear model for a wide range of learning rates using (a) sigmoid cross entropy loss and (b) softmax cross entropy loss. Shown is the average validation accuracy for 20 linearly separable datasets. In every case, the softmax-based model converges faster than a sigmoid-based model with the same learning rate.

$$\boldsymbol{p} = f(\boldsymbol{H}\boldsymbol{x}) \tag{C.13}$$

For each dataset we train two models: One model with a single output neuron based on a sigmoid cross entropy loss and one with two output neurons based on a softmax cross entropy loss. For the sigmoid-based model $\boldsymbol{H}$ has the form $\boldsymbol{H} \in \mathbb{R}^{1\times 10}$ and f is the sigmoid function. A datapoint $\boldsymbol{x}$ is classified as belonging to class c_0 if $f(\boldsymbol{H}\boldsymbol{x}) > 0.5$ and as c_1 otherwise. For the softmax-based model $\boldsymbol{H} \in \mathbb{R}^{2\times 10}$ and f is the softmax function. Classification is performed by $\arg\max_{c_k \in \mathcal{C}} p_k$.

We train both models using a simple gradient descent optimizer without momentum to not complicate the interpretability of the results. Both models are trained for 5000 epochs using the same learning rate $\lambda = 0.01$. After each epoch, we evaluate the classification performance of both models on a validation set.

Figure C.1 shows the results of this experiment. It becomes clear, that – on average – the model using the softmax cross entropy loss converges faster than the model using the sigmoid cross entropy loss. Although the softmax model contains more parameters than the sigmoid model, no models is more powerful than the other: Both solve a linear classification problem and – as we have shown above – we can convert the sigmoid model into a softmax model and vice versa for evaluation.

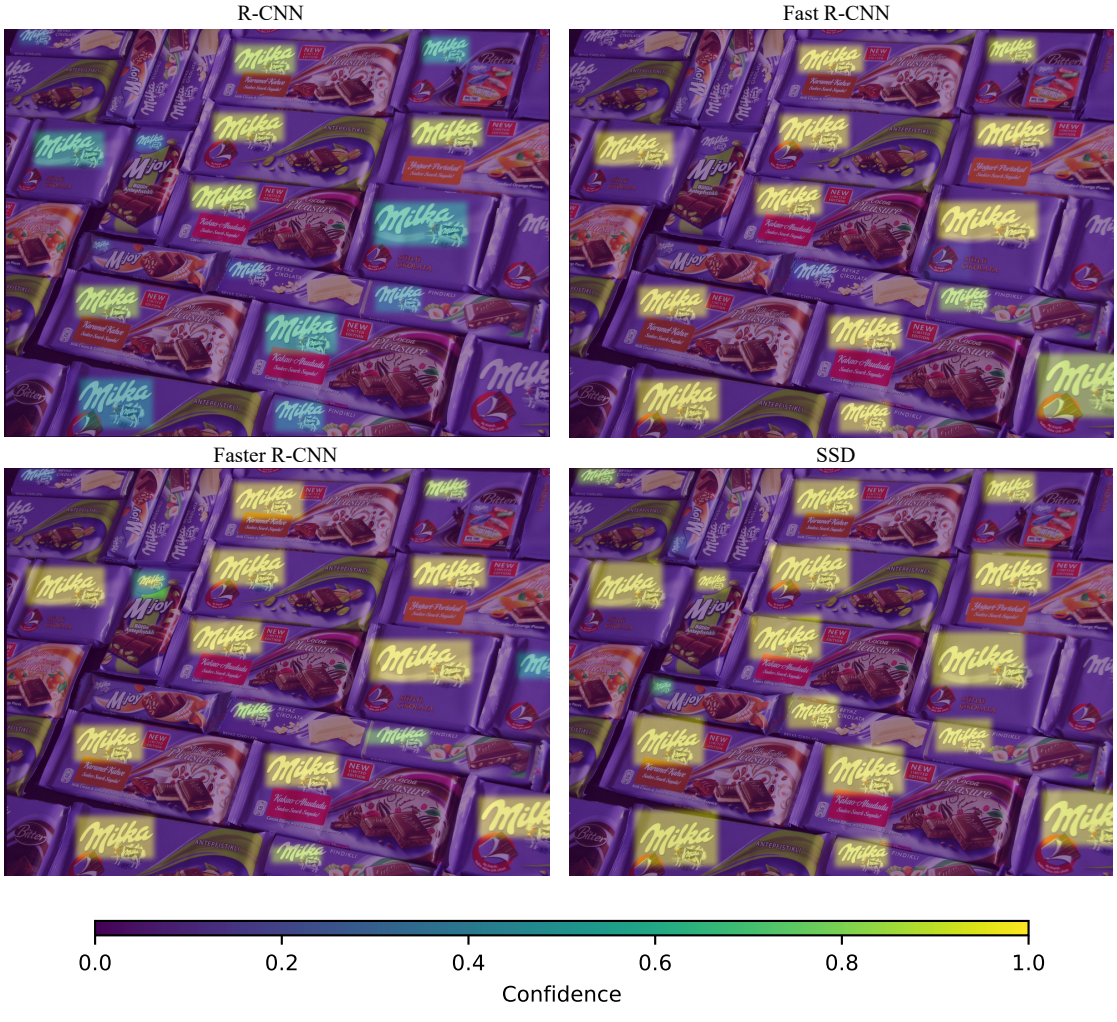
So far we have only looked at a single learning rate. In order to exclude the possibility that the optimal learning rate for a softmax-based model is different from the optimal learning rate for a sigmoid-based model, we repeat the experiment for a wide range of different learning rates. The results of this experiment are show in Figure C.2. It is apparent that for every learning rate, the softmax-based model converges faster than a sigmoid-based model with the same learning rate.

This suggests, that we can always choose a softmax-based model that converges faster than a sigmoid-based model. We therefore feel justified in concluding that even for a binary classification problem it is beneficial to use softmax-based models instead of sigmoid-based models.

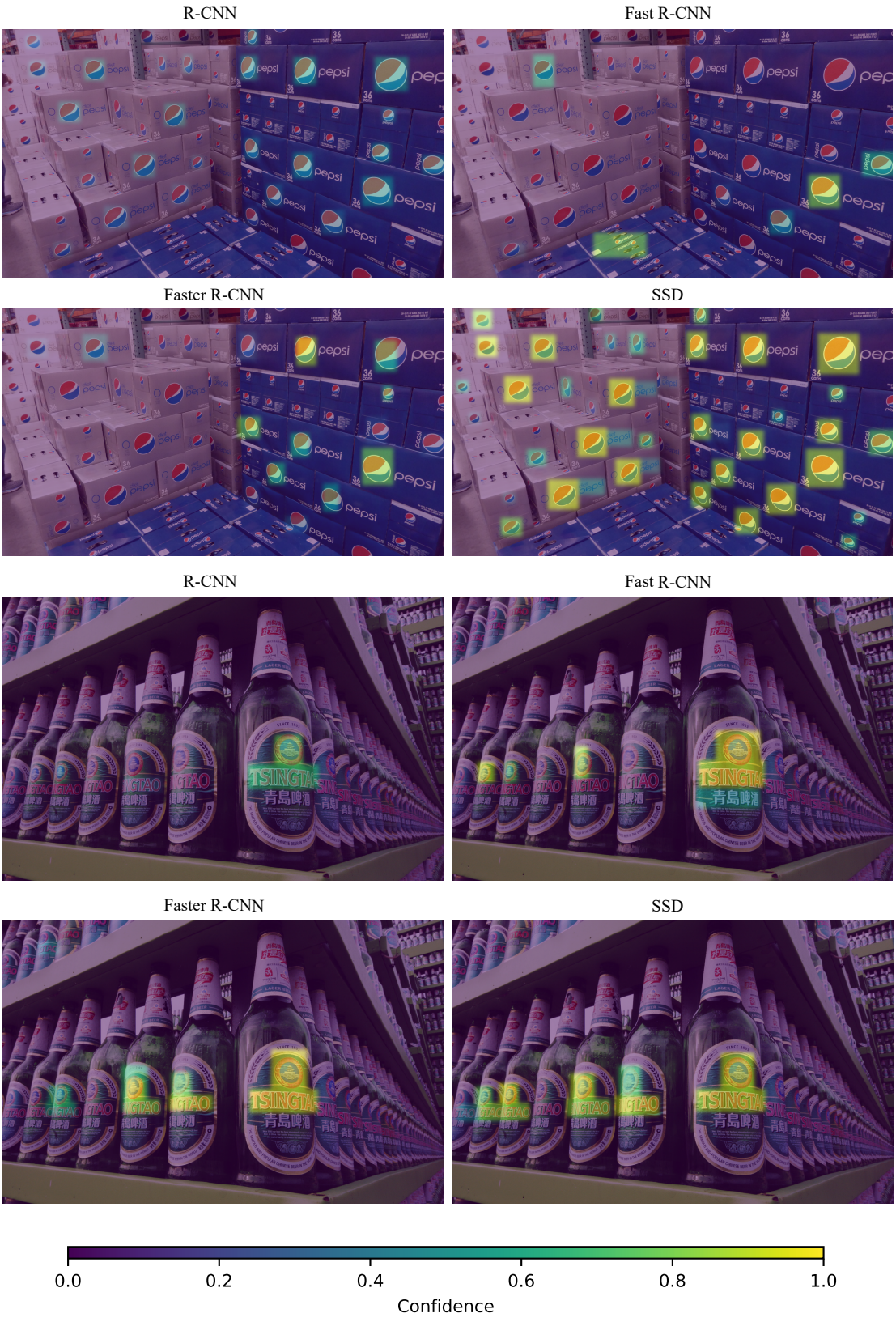
Appendix D

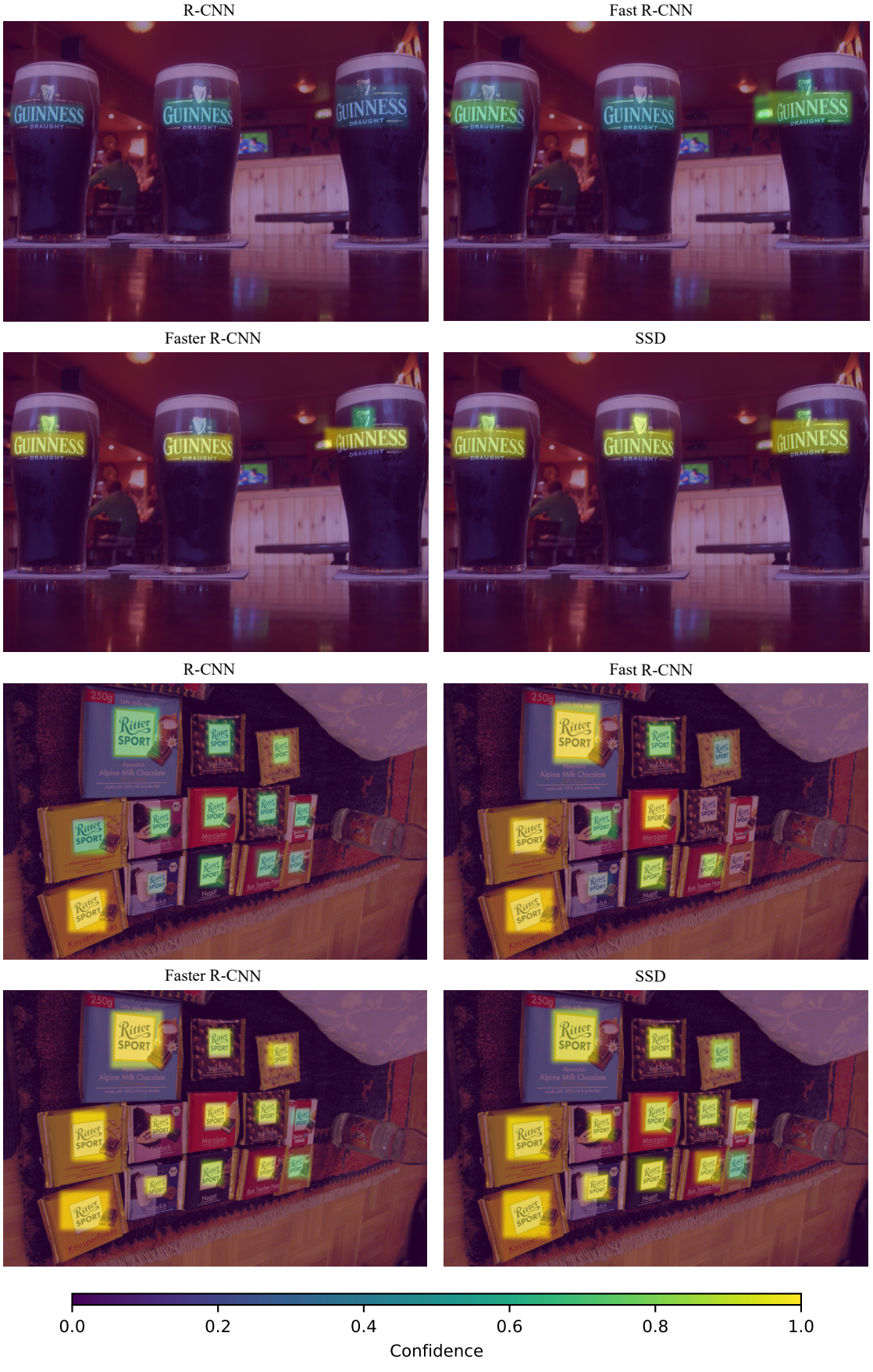
Detection Examples

Here, we show some exemplary detections to illustrate the differences between our implementations of R-CNN, Fast(er) R-CNN and SSD. Since Fast(er) R-CNN and SSD use a softmax classifier their confidence scores are not directly comparable to R-CNN which uses SVMs as a classifier. The scores shown here are SVM scores which have been min/max-normalized into the interval of $[0, 1]$. This normalization has been performed across all detections in the dataset but separately for every class.




R-CNN
Fast R-CNN
Faster R-CNN
SSD
R-CNN
Fast R-CNN
Faster R-CNN
SSD
0.0
0.2
0.4
0.6
0.8
1.0
Confidence


R-CNN
Fast R-CNN
Faster R-CNN
SSD
R-CNN
Fast R-CNN
Faster R-CNN
SSD
0.0
0.2
0.4
0.6
0.8
1.0
Confidence


R-CNN
Fast R-CNN
Faster R-CNN
SSD
R-CNN
Fast R-CNN
Faster R-CNN
SSD
0.0
0.2
0.4
0.6
0.8
1.0
Confidence

List of Figures

Bibliography

[1] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2189–2202, November 2012.

[2] S. Y. Arafat, S. A. Husain, I. A. Niaz, and M. Saleem. Logo detection and recognition in video stream. In *IEEE International Conference on Digital Information Management*, pages 163–168, July 2010.

[3] A. D. Bagdanov, L. Ballan, M. Bertini, and A. Del Bimbo. Trademark matching and retrieval in sports video databases. In *ACM International Workshop on Multimedia Information Retrieval*, MIR '07, pages 79–86, 2007.

[4] Y. Bao, H. Li, X. Fan, R. Liu, and Q. Jia. Region-based cnn for logo detection. In *ACM International Conference on Internet Multimedia Computing and Service*, ICIMCS'16, pages 319–322, 2016.

[5] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2874–2883, June 2016.

[6] S. Belongie, G. Mori, and J. Malik". *Matching with Shape Contexts"*, pages 81–105. Birkhäuser Boston, Boston, MA, 2006.

[7] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[8] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis. Soft-nms — improving object detection with one line of code. In *IEEE International Conference on Computer Vision*, pages 5562–5570, October 2017.

[9] L. Bombonato, G. Camara-Chavez, and P. Silva. Real-time brand logo recognition. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 111–118, Cham, 2018. Springer International Publishing.

[10] Z.i Cai and N. Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[11] J. Carreira and C. Sminchisescu. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1312–1328, July 2012.

[12] M. Cheng, Z. Zhang, W. Lin, and P. Torr. Bing: Binarized normed gradients for objectness estimation at 300fps. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3286–3293, June 2014.

[13] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[14] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893 vol. 1, June 2005.

[15] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.

[16] P. Dollar and C. L. Zitnick. Structured forests for fast edge detection. In *IEEE International Conference on Computer Vision*, pages 1841–1848, December 2013.

[17] C. Eggert, S. Brehm, A. Winschel, D. Zecha, and R. Lienhart. A closer look: Small object detection in faster r-cnn. In *IEEE International Conference on Multimedia and Expo*, pages 421–426, July 2017.

[18] C. Eggert, A. Winschel, and R. Lienhart. On the benefit of synthetic data for company logo detection. In *ACM International Conference on Multimedia*, MM '15, pages 1283–1286, 2015.

[19] C. Eggert, A. Winschel, D. Zecha, and R. Lienhart. Saliency-guided selective magnification for company logo detection. In *IAPR International Conference on Pattern Recognition*, pages 651–656, December 2016.

[20] C. Eggert, D. Zecha, S. Brehm, and R. Lienhart. Improving small object proposals for company logo detection. In *ACM International Conference on Multimedia Retrieval*, ICMR '17, pages 167–174, 2017.

[21] I. Endres and D. Hoiem. Category-independent object proposals with diverse ranking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(2):222–234, Feb 2014.

[22] M Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[23] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

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

[25] P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, January 2005.

[26] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. *Theory of Computing*, 8(19):415–428, 2012.

[27] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages II–II, June 2003.

[28] M. A. Fischler and R. A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, C-22(1):67–92, January 1973.

[29] D. Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, 93(26):429–441, November 1946.

[30] R. Girshick. Fast r-cnn. In *IEEE International Conference on Computer Vision*, pages 1440–1448, December 2015.

[31] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, June 2014.

[32] A. Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69(3):331–371, September 1910.

[33] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 447–456, June 2015.

[34] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *IEEE International Conference on Computer Vision*, pages 2980–2988, October 2017.

[35] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*, pages 1026–1034, December 2015.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, June 2016.

[37] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645, Cham, 2016. Springer International Publishing.

[38] S. C. H. Hoi, X. Wu, H. Liu, Y. Wu, H. Wang, H. Xue, and Q. Wu. Logo-net: Large-scale deep logo detection and brand recognition with deep region-based convolutional networks. volume abs/1511.02462, 2015.

[39] M. Holschneider, R. Kronland-Martinet, J. Morlet, and Ph. Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets*, pages 286–297, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[40] S. Honari, J. Yosinski, P. Vincent, and C. Pal. Recombinator networks: Learning coarse-to-fine feature aggregation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5743–5752, June 2016.

[41] P. J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, March 1964.

[42] F. N. Iandola, A. Shen, P. Gao, and K. Keutzer. Deeplogo: Hitting logo recognition with the deep neural network hammer. *CoRR*, abs/1510.02131, 2015.

[43] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ILMS International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[44] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang. Acquisition of localization confidence for accurate object detection. In *European Conference on Computer Vision*, September 2018.

[45] A. Joly and O. Buisson. Logo retrieval with a contrario visual query expansion. In *ACM International Conference on Multimedia*, pages 581–584, 2009.

[46] T. Kong, A. Yao, Y. Chen, and F. Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 845–853, June 2016.

[47] R. E. Korf. Optimal rectangle packing: Initial results. In *International Conference on Automated Planning and Scheduling*, pages 287–295, 2003.

[48] R. E. Korf, M. D. Moffitt, and M. E. Pollack. Optimal rectangle packing. *Annals of Operations Research*, 179(1):261–295, September 2010.

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105. Curran Associates, Inc., 2012.

[50] H. Kuang, K. Yang, L. Chen, Y. Li, L. L. H. Chan, and H. Yan. Bayes saliency-based object proposal generator for nighttime traffic images. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):814–825, March 2018.

[51] H. Law and J. Deng. Cornernet: Detecting objects as paired keypoints. In *European Conference on Computer Vision*, September 2018.

[52] P. Letessier, A. Joly, and O. Buisson. Scalable mining of small visual objects. In *ACM International Conference on Multimedia*, 2012.

[53] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *IEEE International Conference on Image Processing*, volume 1, September 2002.

[54] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 936–944, July 2017.

[55] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755, Cham, 2014. Springer International Publishing.

[56] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37, Cham, 2016. Springer International Publishing.

[57] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[58] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[59] S. Manen, M. Guillaumin, and L. V. Gool. Prime object proposals with randomized prim's algorithm. In *IEEE International Conference on Computer Vision*, pages 2536–2543, Dec 2013.

[60] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pages 483–499, Cham, 2016. Springer International Publishing.

[61] G. Oliveira, X. Frazão, A. Pimentel, and B. Ribeiro. Automatic graphic logo detection via fast region-based convolutional networks. In *International Joint Conference on Neural Networks*, pages 985–991, July 2016.

[62] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, Jan 1979.

[63] P. Rantalankila, J. Kannala, and E. Rahtu. Generating object segmentation proposals using global and local search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2417–2424, June 2014.

[64] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 512–519, June 2014.

[65] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, June 2016.

[66] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, June 2017.

[67] S. Romberg and R. Lienhart. Bundle min-hashing. *International Journal of Multimedia Information Retrieval*, 2(4):243–259, November 2013.

[68] S. Romberg, L. G. Pueyo, R. Lienhart, and R. van Zwol. Scalable logo recognition in real-world images. In *ACM International Conference on Multimedia Retrieval*, ICMR '11, pages 25:1–25:8, 2011.

[69] R. Ronfard, C. Schmid, and B. Triggs. Learning to parse pictures of people. In *European Conference on Computer Vision*, ECCV '02, pages 700–714, 2002.

[70] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, Cham, 2015. Springer International Publishing.

[71] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations*, April 2014.

[72] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference of Learning Representations*, 2015.

[73] H. Su, X. Zhu, and S. Gong. Deep learning logo detection with data expansion by synthesising context. In *IEEE Winter Conference on Applications of Computer Vision*, pages 530–539, March 2017.

[74] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, June 2015.

[75] O. Tursun and S. Kalkan. Metu dataset: A big dataset for benchmarking trademark retrieval. In *IAPR International Conference on Machine Vision Applications*, pages 514–517, May 2015.

[76] A. Tüzkö, C. Herrmann, D. Manger, and B. Jürgen. Open Set Logo Detection and Retrieval. In *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2018.

[77] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, September 2013.

[78] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

[79] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, December 2001.

[80] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. Hoggles: Visualizing object detection features. In *IEEE International Conference on Computer Vision*, pages 1–8, December 2013.

[81] M. Weber, M. Welling, and P. Perona. Towards automatic discovery of object categories. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 101–108 vol.2, June 2000.

[82] A. Winschel, R. Lienhart, and C. Eggert. Diversity in object proposals. *CoRR*, abs/1603.04308, 2016.

[83] Z. Zhang, Y. Liu, X. Chen, Y. Zhu, M. Cheng, V. Saligrama, and P. H. S. Torr. Sequential optimization for efficient high-quality object proposal generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1209–1223, May 2018.

[84] Q. Zhu, L. Wang, Y. Wu, and J. Shi. Contour context selection for object detection: A set-to-set contour matching approach. In *European Conference on Computer Vision*, pages 774–787, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[85] C. L. Zitnick and P. Dollar. Edge boxes: Locating object proposals from edges. In *European Conference on Computer Vision*, pages 391–405, Cham, 2014. Springer International Publishing.