

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Olavi Stenroos

Object detection from images using convolutional neural networks

Master's Thesis
Espoo, July 28, 2017

Supervisor: Assistant Professor Juho Kannala
Advisor: Assistant Professor Juho Kannala

Author:	Olavi Stenroos	
Title:	Object detection from images using convolutional neural networks	
Date:	July 28, 2017	Pages: 75
Major:	Computer Science	Code: SCI3042
Supervisor:	Assistant Professor Juho Kannala	
Advisor:	Assistant Professor Juho Kannala	
<p>Object detection is a subfield of computer vision that is currently heavily based on machine learning. For the past decade, the field of machine learning has been dominated by so-called deep neural networks, which take advantage of improvements in computing power and data availability. A subtype of a neural network called a convolutional neural network (CNN) is well-suited for image-related tasks. The network is trained to look for different features, such as edges, corners and colour differences, across the image and to combine these into more complex shapes. For object detection, the system has to both estimate the locations of probable objects and to classify these.</p> <p>For this master's thesis, we reviewed the current literature on convolutional object detection and tested the implementability of one of the methods. We found that convolutional object detection is still evolving as a technology, despite outranking other object detection methods. By virtue of free availability of datasets and pretrained networks, it is possible to create a functional implementation of a deep neural network without access to specialist hardware. Pretrained networks can also be used as a starting point for training new networks, decreasing costly training time.</p> <p>For the experimental part, we implemented Fast R-CNN using MATLAB and MatConvNet and tested a general object detector on two different traffic-related datasets. We found that Fast R-CNN is relatively precise and considerably faster than the original convolutional object detection method, R-CNN, and can be implemented on a home computer. Advanced methods, such as Faster R-CNN and SSD, improve the speed of Fast R-CNN. We also experimented with a geometry-based scene estimation model, which was reported to improve the precision of a previous generation object detection method. We found that with our implementation of Fast R-CNN there was no such improvement, although further adjustments are possible. Combining whole scene modelling with convolutional networks is a potential subject of further study.</p>		
Keywords:	computer vision, object detection, machine learning, neural networks, convolutional neural networks	
Language:	English	

Aalto-yliopisto

Perustieteiden korkeakoulu

Master's Programme in Computer, Communication and Information Sciences

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Olavi Stenroos
Työn nimi: Kohteentunnistus kuvista konvoluutioneuroverkoilla	
Päiväys:	28. heinäkuuta 2017
Pääaine:	Computer Science
Valvoja:	Apulaisprofessori Juho Kannala
Ohjaaja:	Apulaisprofessori Juho Kannala
<p>Kohteentunnistus on tietokonenäön osa-alue, joka pohjautuu vahvasti koneoppimiseen. Koneoppimisen tämän vuosikymmenen trendi ovat niin kutsutut syväoppivat neuroverkot, jotka perustuvat laskentatehon ja datan saatavuuden kasvuun. Konvoluutioneuroverkko on neuroverkon alatyyppi, joka sopii erityisesti kuvien liittyvien ongelmien ratkaisuun. Verkko opetetaan etsimään yksinkertaisia kuvapiirteitä ja yhdistelemään näitä monimutkaisemmiksi muodoiksi. Kohteentunnistusongelmassa menetelmän tulee sekä paikallistaa että luokitella kiinnostavat kohteet.</p> <p>Diplomityöni sisältää kirjallisuskatsauksen konvoluutioon perustuviin kohteentunnistusmenetelmiin sekä selostuksen erään tällaisen menetelmän toteuttamisesta. Konvoluutioon perustuva kohteentunnistus kehittyi tällä hetkellä kiivaasti ja on muita menetelmiä tarkempi ja nopeampi. Vapaasti saatavilla olevien opetusaineistojen ja esiopetetuujen verkkojen avulla syvä neuroverkko on mahdollista toteuttaa suhteellisen vaivattomasti ja ilman erikoislaitteita. Esiopetettuja verkkuja voidaan käyttää pohjana uusien verkkojen kouluttamiseen.</p> <p>Kokeellisessa osassa toteutin Fast R-CNN:n MATLABin ja MatConvNetin avulla ja kokeilin kahden liikennedata-aineiston avulla, kuinka yleisellä datalla opetettu verkko suoriutui erityisongelmasta. Fast R-CNN suoritti tunnistuksen kohtuullisen tarkasti ja on edeltäjäänsä R-CNN:ää sen verran nopeampi, että on toteutettavissa kotitietokoneella. Kehittyneemmät menetelmät, kuten Faster R-CNN ja SSD, olisivat tätäkin nopeampia, mutta eivät juurikaan tarkempia. Kokeilin myös yhdistää Fast R-CNN geometriantunnistusmenetelmän kanssa, jota on käytetty aikaisemman sukupolven menetelmän tarkkuuden parantamiseen. Konvoluutio-menetelmän kanssa tarkkuus ei noussut, mutta tutkin työssäni, mistä tämä johtui ja kuinka koko näkymän estimointia voidaan mahdollisesti hyödyntää konvoluutioneuroverkoissa.</p>	
Asiasanat:	tietokonenäkö, kohteentunnistus, koneoppiminen, neuroverkot, konvoluutioneuroverkot
Kieli:	Englanti

Acknowledgements

I would like to thank my supervisor Juho Kannala for suggesting the topic and for guidance. I would also like to thank my wife Niina Stenroos for posing for the example photo.

Espoo, July 28, 2017

Olavi Stenroos

Abbreviations and Acronyms

CNN	Convolutional Neural Network
CPU	Central Processing Unit
FC	Fully Connected (layer or network)
FCN	Fully Convolutional Network
FPS	Frames Per Second
GPU	Graphics Processing Unit
IoU	Intersection over Union
NMS	Non-maximum suppression
OiP	Putting Objects in Perspective
R-CNN	Convolutional Neural Network with Region proposals
RoI	Region of Interest
RPN	Region Proposal Network
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machine

Contents

Abbreviations and Acronyms	5
1 Introduction	9
1.1 Problem statement	9
1.2 Structure of the thesis	10
2 Background	11
2.1 Machine learning	11
2.1.1 Types	11
2.1.2 Features	12
2.1.3 Generalization	12
2.2 Neural networks	13
2.2.1 Origins	13
2.2.2 Multi-layer networks	14
2.2.3 Back-propagation	15
2.2.4 Activation function types	16
2.2.5 Deep learning	16
2.3 Computer vision	17
2.3.1 Overview	18
2.3.2 Object detection	18
2.4 Convolutional neural networks	19
2.4.1 Justification	19
2.4.2 Basic structure	20
2.4.3 Pooling and stride	21
2.4.4 Additional layers	22
2.4.5 Regularization and data augmentation	23
2.4.6 Development	23
3 Convolutional object detection	25
3.1 R-CNN	25
3.1.1 General description	25

3.1.2	Drawbacks	26
3.2	Fast R-CNN	26
3.2.1	General description	27
3.2.2	Classification performance	27
3.2.3	Training	28
3.3	Region proposal generation and use	28
3.3.1	Overview	28
3.3.2	Selective Search	29
3.3.3	Edge Boxes	30
3.4	Advanced convolutional object detection	31
3.4.1	Faster R-CNN	31
3.4.2	SSD	32
3.5	Comparing the methods	32
4	Experimental setup	34
4.1	Overview and selection criteria	34
4.2	Object detection	35
4.3	Region generation	37
4.3.1	Selective Search parameters	37
4.3.2	Edge Boxes parameters	38
4.4	Evaluating objects in context	39
4.4.1	Geometric Context	39
4.4.2	Putting Objects in Perspective	40
4.4.3	Substituting the object detector	41
4.5	Datasets	41
4.5.1	Standard benchmarks	41
4.5.2	Test data	42
4.6	Evaluation metrics	44
5	Implementation	46
5.1	Environment	46
5.2	MatConvNet and Fast R-CNN	46
5.3	Selective Search and Edge Boxes	47
5.4	Geometric inference	48
5.5	Evaluation and visualization	49
5.6	Challenges and additional details	49
6	Evaluation	51
6.1	Fast R-CNN with different regions	51
6.2	False positives and false negatives	53
6.3	Non-maximum suppression	54

6.4	Geometric inference	55
7	Discussion	59
7.1	Overlap	59
7.2	Built-in invariance	61
7.3	Network depth	62
7.4	Region generation	63
7.5	Region pooling and processing	64
7.6	Object detection in context	64
8	Conclusions	67
8.1	Theory	67
8.2	Practice	68
8.3	Results	68
8.4	The future	69

Chapter 1

Introduction

There is an ever-increasing amount of image data in the world, and the rate of growth itself is increasing. Infotrends estimates that in 2016 still cameras and mobile devices captured more than 1.1 trillion images [3]. According to the same estimate, in 2020 the figure will increase to 1.4 trillion. Many of these images are stored in cloud services or published on the Internet. In 2014, over 1.8 billion images were uploaded daily to the most popular platforms, such as Instagram and Facebook [4].

Going beyond consumer devices, there are cameras all over the world that capture images for automation purposes. Cars monitor the road, and traffic cameras monitor the same cars. Robots need to understand a visual scene in order to smartly build devices and sort waste. Imaging devices are used by engineers, doctors and space explorers alike.

To effectively manage all this data, we need to have some idea about its contents. Automated processing of image contents is useful for a wide variety of image-related tasks. For computer systems, this means crossing the so-called semantic gap between the pixel level information stored in the image files and the human understanding of the same images. Computer vision attempts to bridge this gap.

1.1 Problem statement

Objects contained in image files can be located and identified automatically. This is called *object detection* and is one of the basic problems of computer vision. As we will demonstrate, convolutional neural networks are currently the state-of-the-art solution for object detection. The main task of this thesis is to review and test convolutional object detection methods.

In the theoretical part, we review the relevant literature and study how

convolutional object detection methods have improved in the past few years. In the experimental part, we study how easily a convolutional object detection system can be implemented in practice, test how well a detection system trained on general image data performs in a specific task and explore, both experimentally and based on the literature, how the current systems can be improved.

1.2 Structure of the thesis

The thesis begins with two theoretical chapters. Since convolutional object detection is a combination of several fields of computer science, we need to discuss several theoretical topics that seem disparate at first. In chapter 2, we begin with a short introduction to machine learning and neural networks. Next, we discuss computer vision and object detection as its subfield. We end the chapter by introducing convolutional neural networks as a combination of machine learning and computer vision. In chapter 3, we discuss how convolutional networks can be used for object detection and review the relevant literature and methods.

In chapter 4, we move to the experimental part. We discuss what kind of experimental setup we used for testing a convolutional network. We discuss not only the details of the experiments, but also the details of the datasets. Further, we discuss how we will evaluate the results. In chapter 5, we discuss the practical implementation of the experiments by discussing the required software and hardware and how these were used.

In chapter 6, we evaluate the results. We provide not only the numerical results, but also some analysis of them. However, the more high-level analysis is left to chapter 7, where we discuss potential improvements to current methods. We base this discussion on the results of our experiments as well as topics known from the literature. In chapter 8, we provide a review of the thesis and some concluding remarks.

Chapter 2

Background

In this chapter, we provide the theoretical background necessary for understanding the methods discussed in the next chapter. First, we discuss relevant details of machine learning, neural networks, and computer vision. Finally, we explain how these disciplines are combined in convolutional neural networks.

2.1 Machine learning

Learning algorithms are widely used in computer vision applications. Before considering image related tasks, we are going to have a brief look at basics of machine learning.

Machine learning has emerged as a useful tool for modelling problems that are otherwise difficult to formulate exactly. Classical computer programs are explicitly programmed by hand to perform a task. With machine learning, some portion of the human contribution is replaced by a learning algorithm. [22, p. 2] As availability of computational capacity and data has increased, machine learning has become more and more practical over the years, to the point of being almost ubiquitous.

2.1.1 Types

A typical way of using machine learning is *supervised learning* [11, p. 3]. A learning algorithm is shown multiple examples that have been annotated or labelled by humans. For example, in the object detection problem we use training images where humans have marked the locations and classes of relevant objects. After learning from the examples, the algorithm is able to predict the annotations or labels of previously unseen data. *Classification*

and *regression* are the most important task types [11, p. 3]. In classification, the algorithm attempts to predict the correct class of a new piece of data based on the training data. In regression, instead of discrete classes, the algorithm tries to predict a continuous output.

In *unsupervised learning*, the algorithm attempts to learn useful properties of the data without a human teacher telling what the correct output should be. Classical example of unsupervised learning is clustering [11, p. 3]. More recently, especially with the advent of deep learning technologies, unsupervised *preprocessing* has become a popular tool in supervised learning tasks for discovering useful representations of the data [9].

2.1.2 Features

Some kind of preprocessing is almost always needed. Preprocessing the data into a new, simpler variable space is called *feature extraction* [11, p. 2]. Often, it is impractical or impossible to use the full-dimensional training data directly. Rather, detectors are programmed to extract interesting features from the data, and these features are used as input to the machine learning algorithm.

In the past, the feature detectors were often hand-crafted. The problem with this approach is that we do not always know in advance, which features are interesting. The trend in machine learning has been towards learning the feature detectors as well, which enables using the complete data [22, pp. 3–5].

2.1.3 Generalization

Since the training data cannot include every possible instance of the inputs, the learning algorithm has to be able to *generalize* in order to handle unseen data points [11, p. 2]. Too simple model estimate can fail to capture important aspects of the true model. On the other hand, too complex methods can *overfit* by modelling unimportant details and noise, which also leads to bad generalization [11, p. 9]. Typically, overfitting happens when a complex method is used in conjunction with too little training data. An overfitted model learns to model the known examples but does not understand what connects them.

The performance of the algorithm can be evaluated from the quality and quantity of errors. A *loss function*, such as mean squared error, is used to assign a cost to the errors [11, p. 41]. The objective in the training phase is to minimize this loss.

2.2 Neural networks

Neural networks are a popular type of machine learning model. A special case of a neural network called the *convolutional neural network* (CNN) is the primary focus of this thesis. Before discussing CNNs, we will discuss how regular neural networks work.

2.2.1 Origins

Neural networks were originally called artificial neural networks, because they were developed to mimic the neural function of the human brain. Pioneering research includes the threshold logic unit by Warren McCulloch and Walter Pitts in 1943 and the perceptron by Frank Rosenblatt in 1957 [41].

Even though the inspiration from biology is apparent, it would be misleading to overemphasize the connection between artificial neurons and biological neurons or neuroscience. The human brain contains approximately 100 billion neurons operating in parallel [37]. Artificial neurons are mathematical functions implemented on more-or-less serial computers. Research into neural networks is mostly guided by developments in engineering and mathematics rather than biology [22, p. 169].

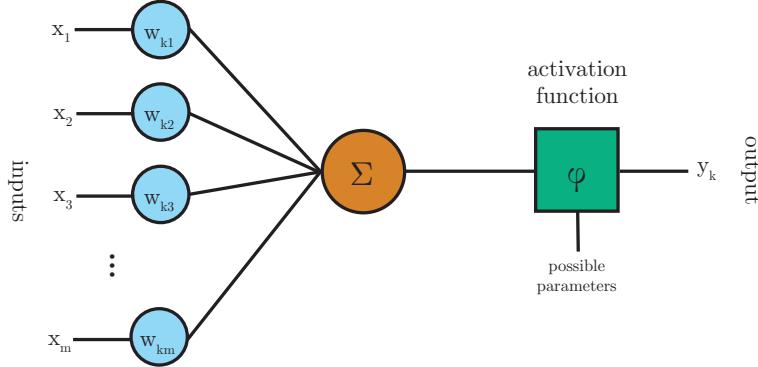


Figure 2.1: An artificial neuron.

An artificial neuron based on the McCulloch-Pitts model is shown in Figure 2.1 [11, pp. 227–229]. The neuron k receives m input parameters x_j . The neuron also has m weight parameters w_{kj} . The weight parameters often include a bias term that has a matching dummy input with a fixed value of 1. The inputs and weights are linearly combined and summed. The sum is then fed to an activation function φ that produces the output y_k of the neuron:

$$y_k = \varphi(s_k) = \varphi\left(\sum_{j=0}^m w_{kj}x_j\right).$$

The neuron is trained by carefully selecting the weights to produce a desired output for each input.

2.2.2 Multi-layer networks

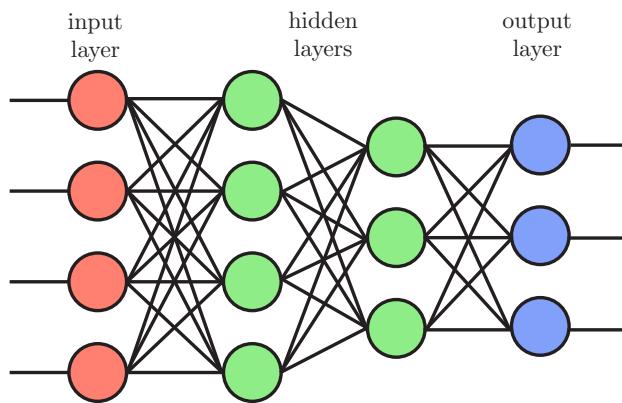


Figure 2.2: A fully-connected multi-layer neural network.

A neural network is a combination of artificial neurons. The neurons are typically grouped into layers. In a fully-connected feed-forward multi-layer network, shown in Figure 2.2, each output of a layer of neurons is fed as input to each neuron of the next layer. Thus, some layers process the original input data, while some process data received from other neurons. Each neuron has a number of weights equal to the number of neurons in the previous layer. [11, pp. 227–229]

A multi-layer network typically includes three types of layers: an input layer, one or more hidden layers and an output layer [11, p. 227]. The input layer usually merely passes data along without modifying it. Most of the computation happens in the hidden layers. The output layer converts the hidden layer activations to an output, such as a classification. A multi-layer feed-forward network with at least one hidden layer can function as a universal approximator i.e. can be constructed to compute almost any function [27].

In this thesis, we will mostly discuss fully-connected networks and convolutional networks (see section 2.4). Convolutional networks utilize parameter

sharing and have limited connections compared to fully-connected networks [22, p. 335]. Other network types, such as recurrent networks, are outside the scope of this thesis.

2.2.3 Back-propagation

A neural network is trained by selecting the weights of all neurons so that the network learns to approximate target outputs from known inputs. It is difficult to solve the neuron weights of a multi-layer network analytically. The *back-propagation algorithm* [22, pp. 204–210] [11, pp. 241–245] provides a simple and effective solution to solving the weights iteratively. The classical version uses gradient descent as optimization method. Gradient descent can be quite time-consuming and is not guaranteed to find the global minimum of error, but with proper configuration (known in machine learning as hyperparameters) works well enough in practice.

In the first phase of the algorithm, an input vector is propagated forward through the neural network. Before this, the weights of the network neurons have been initialized to some values, for example small random values. The received output of the network is compared to the desired output (which should be known for the training examples) using a loss function. The gradient of the loss function is then computed. This gradient is also called the error value. When using mean squared error as the loss function, the output layer error value is simply the difference between the current and desired output.

The error values are then propagated back through the network to calculate the error values of the hidden layer neurons. The hidden neuron loss function gradients can be solved using the chain rule of derivatives. Finally, the neuron weights are updated by calculating the gradient of the weights and subtracting a proportion of the gradient from the weights. This ratio is called the *learning rate* [11, p. 240]. The learning rate can be fixed or dynamic. After the weights have been updated, the algorithm continues by executing the phases again with different input until the weights converge.

In the above description, we have described *online learning* that calculates the weight updates after each new input [22, pp. 277–279]. Online learning can lead to “zig-zagging” behaviour, where the single data point estimate of the gradient keeps changing direction and does not approach the minimum directly. Another way of computing the updates is *full batch learning*, where we compute the weight updates for the complete dataset [22, pp. 277–279]. This is quite computationally heavy and has other drawbacks. A compromise version is *mini-batch learning*, where we use only some portion of the training set for each update [54].

Mathematical descriptions of the algorithm are readily available in other works [42] [22, p. 204–210].

2.2.4 Activation function types

The activation function φ determines the final output of each neuron. It is important to select the function properly in order to create an effective network.

Early researchers found that perceptrons and other linear systems had severe drawbacks, being unable to solve problems that were not linearly separable, such as the XOR-problem. Sometimes, linear systems can solve these kinds of problems using hand-crafted feature detectors, but this is not the most advantageous use of machine learning. Simply adding layers does not help either, because a network composed of linear neurons remains linear no matter how many layers it has. [22, pp. 171–172]

A light-weight and effective way of creating a non-linear network is using *rectified linear units* (ReLU) [22, pp. 173–177]. A rectified linear function generates the output using a ramp function such as:

$$\varphi(s) = \max(0, s).$$

This type of function is easy to compute and differentiate (for back-propagation). The function is not differentiable at zero, but this has not prevented its use in practice. ReLus have become quite popular lately, often replacing sigmoidal activation functions, which have smooth derivatives but suffer from gradient saturation problems and slower computation.

For multi-class classification problems, the *softmax* activation function [11, pp. 115, 203] is used in the output layer of the network:

$$\varphi(s) = \frac{\exp s_k}{\sum_{k=1}^K \exp s_k}.$$

The softmax function takes a vector of K arbitrarily large values and outputs a vector of K values that range between 0...1 and sum to 1. The values output by the softmax unit can be utilized as class probabilities.

2.2.5 Deep learning

Modern neural networks are often called *deep neural networks*. Even though multi-layer neural networks have existed since the 1980s, several reasons prevented the effective training of networks with multiple hidden layers [22, p. 226].

One of the main problems is the *curse of dimensionality* [22, p. 155]. As the number of variables increases, the number of different configurations of the variables grows exponentially. As the number of configurations increases, the number of training samples should increase in equal measure. Collecting a training dataset of sufficient size is time-consuming and costly or outright impossible.

Fortunately, real-world data is not uniformly distributed and often involves a structure, where the interesting information lies on a low-dimensional manifold. The *manifold hypothesis* assumes that most data configurations are invalid or rare [22, p. 162]. We can decrease dimensionality by learning to represent the data using the coordinates of the manifold. Another way to improve generalization is to assume *local constancy* [22, p. 157]. This means assuming that the function that the neural network learns to approximate should not change much within a small region.

In the past ten years, neural networks have had a renaissance, mainly because of the availability of more powerful computers and larger datasets. In early 2000s, it was discovered that neural networks could be trained efficiently using graphics processing units. GPUs are more efficient for the task than traditional CPUs and provide a relatively cheap alternative to specialist hardware [48]. Today, researchers typically use high-end consumer graphic cards, such as NVIDIA Tesla K40 [20].

Other more theoretical breakthroughs include replacing mean-squared error functions with cross-entropy based functions and replacing sigmoidal activation functions with rectified linear units [22, p. 226].

With deep learning, there is less need for hand-tuned machine learning solutions that were used previously [22, p. 5]. A classical pattern detection system, for example, includes a hand-tuned feature detection phase before a machine learning phase. The deep learning equivalent consists of a single neural network. The lower layers of the neural network learn to recognize the basic features, which are then fed forward to higher layers of the network.

2.3 Computer vision

Next, we are going to discuss computer vision in general and explore the primary subject of this thesis, object detection, as a subproblem of computer vision.

2.3.1 Overview

Computer vision deals with the extraction of meaningful information from the contents of digital images or video. This is distinct from mere image processing, which involves manipulating visual information on the pixel level. Applications of computer vision include image classification, visual detection, 3D scene reconstruction from 2D images, image retrieval, augmented reality, machine vision and traffic automation [49].

Today, machine learning is a necessary component of many computer vision algorithms [44]. Such algorithms can be described as a combination of image processing and machine learning. Effective solutions require algorithms that can cope with the vast amount of information contained in visual images, and critically for many applications, can carry out the computation in real time [28].

2.3.2 Object detection

Object detection is one of the classical problems of computer vision and is often described as a difficult task. In many respects, it is similar to other computer vision tasks, because it involves creating a solution that is invariant to deformation and changes in lighting and viewpoint. What makes object detection a distinct problem is that it involves both locating and classifying regions of an image [20]. The locating part is not needed in, for example, whole image classification.

To detect an object, we need to have some idea where the object might be and how the image is segmented. This creates a type of chicken-and-egg problem, where, to recognize the shape (and class) of an object, we need to know its location, and to recognize the location of an object, we need to know its shape. [53] Some visually dissimilar features, such as the clothes and face of a human being, may be parts of the same object, but it is difficult to know this without recognizing the object first. On the other hand, some objects stand out only slightly from the background, requiring separation before recognition. [51]

Low-level visual features of an image, such as a saliency map, may be used as a guide for locating candidate objects [53]. The location and size is typically defined using a *bounding box*, which is stored in the form of corner coordinates. Using a rectangle is simpler than using an arbitrarily shaped polygon, and many operations, such as convolution, are performed on rectangles in any case. The sub-image contained in the bounding box is then classified by an algorithm that has been trained using machine learning [21]. The boundaries of the object can be further refined iteratively, after

making an initial guess [49].

During the 2000s, popular solutions for object detection utilized feature descriptors, such as scale-invariant feature transform (SIFT) [38] developed by David Lowe in 1999 and histogram of oriented gradients (HOG) [14] popularized in 2005. In the 2010s, there has been a shift towards utilizing convolutional neural networks [21] [20] [40].

Before the widespread adoption of CNNs, there were two competing solutions for generating bounding boxes. In the first solution, a dense set of region proposals is generated and then most of these are rejected [36]. This typically involves a sliding window detector. In the second solution, a sparse set of bounding boxes is generated using a region proposal method, such as Selective Search [51]. Combining sparse region proposals with convolutional neural networks has provided good results and is currently popular [20].

2.4 Convolutional neural networks

Next, we are going to discuss why and how *convolutional neural networks* (CNN) are used and describe their history.

2.4.1 Justification

The problem with solving computer vision problems using traditional neural networks is that even a modestly sized image contains an enormous amount of information (see section 2.2.5 on deep learning and the curse of dimensionality).

A monochrome 620x480 image contains 297 600 pixels. If each pixel intensity of this image is input separately to a fully-connected network, each neuron requires 297 600 weights. A 1920x1080 full HD image would require 2,073,600 weights. If the images are polychrome, the amount of weights is multiplied by the amount of colour channels (typically three). Thus, we can see that the overall number of free parameters in the network quickly becomes extremely large as the image size increases. Too large models cause overfitting and slow performance [11, p. 9].

Furthermore, many pattern detection tasks require that the solution is translation invariant. It is inefficient to train neurons to separately recognize the same pattern in the left-top corner and in the right-bottom corner of an image. A fully-connected neural network fails to take this kind of structure into account [33].

2.4.2 Basic structure

The basic idea of the CNN was inspired by a concept in biology called the receptive field [19]. Receptive fields are a feature of the animal visual cortex [29]. They act as detectors that are sensitive to certain types of stimulus, for example, edges. They are found across the visual field and overlap each other.

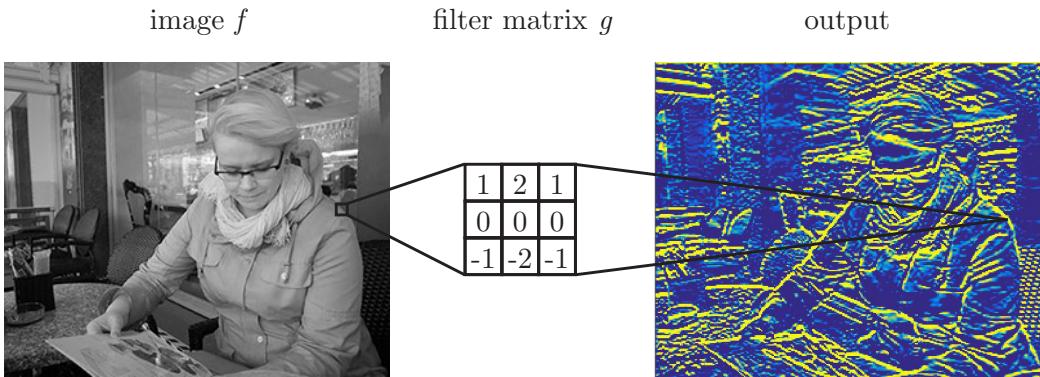


Figure 2.3: Detecting horizontal edges from an image using convolution filtering.

This biological function can be approximated in computers using the convolution operation [39]. In image processing, images can be filtered using convolution to produce different visible effects. Figure 2.3 shows how a hand-selected convolutional filter detects horizontal edges from an image, functioning similarly to a receptive field.

The discrete convolution operation between an image f and a filter matrix g is defined as:

$$h[x, y] = f[x, y] * g[x, y] = \sum_n \sum_m f[n, m]g[x - n, y - m].$$

In effect, the dot product of the filter g and a sub-image of f (with same dimensions as g) centred on coordinates x, y produces the pixel value of h at coordinates x, y [22, pp. 331–332]. The size of the receptive field is adjusted by the size of the filter matrix. Aligning the filter successively with every sub-image of f produces the output pixel matrix h . In the case of neural networks, the output matrix is also called a *feature map* [22, p. 332] (or an *activation map* after computing the activation function). Edges need to be treated as a special case [22, p. 349]. If image f is not padded, the output size decreases slightly with every convolution.

A set of convolutional filters can be combined to form a *convolutional layer* of a neural network [19]. The matrix values of the filters are treated as neuron parameters and trained using machine learning. The convolution operation replaces the multiplication operation of a regular neural network layer. Output of the layer is usually described as a volume. The height and width of the volume depend on the dimensions of the activation map. The depth of the volume depends on the number of filters.

Since the same filters are used for all parts of the image, the number of free parameters is reduced drastically compared to a fully-connected neural layer [33]. The neurons of the convolutional layer mostly share the same parameters and are only connected to a local region of the input. Parameter sharing resulting from convolution ensures translation invariance. An alternative way of describing the convolutional layer is to imagine a fully-connected layer with an infinitely strong prior placed on its weights [22, pp. 345–347]. This prior forces the neurons to share weights at different spatial locations and to have zero weight outside the receptive field.

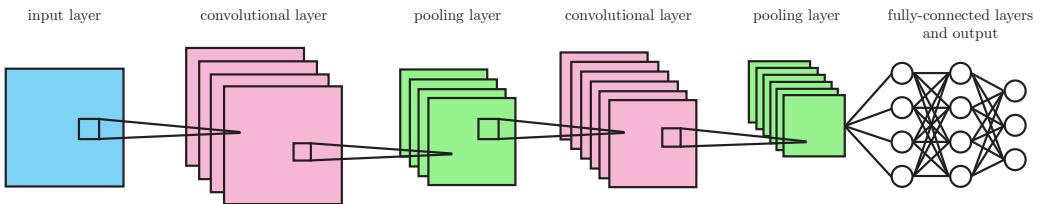


Figure 2.4: An example of a convolutional network.

Successive convolutional layers (often combined with other types of layers, such as pooling described below) form a *convolutional neural network* (CNN). An example of a convolutional network is shown in figure 2.4. The back-propagation training algorithm, described in section 2.2.3, is also applicable to convolutional networks [22, p. 372]. In theory, the layers closer to the input should learn to recognize low-level features of the image, such as edges and corners, and the layers closer to the output should learn to combine these features to recognize more meaningful shapes [19]. In this thesis, we are interested in studying whether convolutional networks can learn to recognize complete objects.

2.4.3 Pooling and stride

To make the network more manageable for classification, it is useful to decrease the activation map size in the deep end of the network. Generally,

the deep layers of the network require less information about exact spatial locations of features, but require more filter matrixes to recognize multiple high-level patterns [22, p. 342]. By reducing the height and width of the data volume, we can increase the depth of the data volume and keep the computation time at a reasonable level.

There are two ways of reducing the data volume size. One way is to include a *pooling layer* after a convolutional layer [22, pp. 339–345]. The layer effectively down-samples the activation maps. Pooling has the added effect of making the resulting network more translation invariant by forcing the detectors to be less precise. However, pooling can destroy information about spatial relationships between subparts of patterns. Typical pooling method is *max-pooling*. Max-pooling simply outputs the maximum value within a rectangular neighbourhood of the activation map [22, pp. 339–342].

Another way of reducing the data volume size is adjusting the *stride* parameter of the convolution operation. The stride parameter controls whether the convolution output is calculated for a neighbourhood centred on every pixel of the input image (stride 1) or for every n th pixel (stride n) [13]. Research has shown that pooling layers can often be discarded without loss in accuracy by using convolutional layers with larger stride value [46]. The stride operation is equivalent to using a fixed grid for pooling.

2.4.4 Additional layers

The convolutional layer typically includes a non-linear activation function, such as a rectified linear activation function (see subsection 2.2.4). Activations are sometimes described as a separate layer between the convolutional layer and the pooling layer.

Some systems, such as [45], also implement a layer called local response normalization, which is used as a regularization technique. Local response normalization mimics a function of biological neurons called lateral inhibition, which causes excited neurons to decrease the activity of neighbouring neurons. However, other regularization techniques are currently more popular and these are discussed in the next section.

The final hidden layers of a CNN are typically fully-connected layers [11, p. 269] [40]. A fully-connected layer can capture some interesting relationships parameter-sharing convolutional layers cannot. However, a fully-connected layer requires a sufficiently small data volume size in order to be practical. Pooling and stride settings can be used to reduce the size of the data volume that reaches the fully-connected layers. A convolutional network that does *not* include any fully-connected layers, is called a *fully convolutional network* (FCN) [40].

If the network is used for classification, it usually includes a softmax output layer [11, p. 269] (see also section 2.2.4). The activations of the topmost layers can also be used directly to generate a feature representation of an image. This means that the convolutional network is used as a large feature detector. [33]

2.4.5 Regularization and data augmentation

Regularization refers to methods that are used to reduce overfitting by introducing additional constraints or information to the machine learning system [22, pp. 228–228]. A classical way of using regularization in neural networks is adding a penalty term to the objective/loss function that penalizes certain types of weights. The parameter sharing feature of convolutional networks is another example of regularization.

There are several regularization techniques that are specific to deep neural networks. A popular technique called dropout [47] attempts to reduce the co-adaptation of neurons. This is achieved by randomly dropping out neurons during training, meaning that a slightly different neural network is used for each training sample or minibatch. This causes the system not to depend too much on any single neuron or connection and provides an effective yet computationally inexpensive way of implementing regularization [22, pp. 258–259]. In convolutional networks, dropout is typically used in the final fully-connected layers [45].

Overfitting can also be reduced by increasing the amount of training data. When it is not possible to acquire more actual samples, data augmentation is used to generate more samples from the existing data [22, pp. 240–241]. For classification using convolutional networks, this can be achieved by computing transformations of the input images that do not alter the perceived object classes, yet provide additional challenge to the system. The images can be, for example, flipped, rotated or subsampled with different crops and scales. Also, noise can be added to the input images [22, pp. 242].

2.4.6 Development

Convolutional neural networks were one of the first successful deep neural networks. The Neocognitron, developed by Fukushima in 1980s, provided a neural network model for translation-invariant object recognition, inspired by biology [19]. Le Cun et al. combined this method with a learning algorithm, i.e. back-propagation [33]. These early solutions were mostly used for hand-written character recognition.

After providing some promising results, the neural network methods faded in prominence and were mostly replaced by support vector machines [21]. Then, in 2012, Krizhevsky et al. [32] achieved excellent results on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset by combining Le Cun’s method with recent fine-tuning methods for deep learning. These results popularised CNNs and led to the development of new powerful object detection methods described in chapter 3 [21].

For the 2014 ImageNet challenge, Simonyan and Zisserman [45] explored the effect of increasing the depth of a convolutional network on localisation and classification accuracy. The team achieved results that improved the then state-of-the-art by using convolutional networks 16 and 19 layers deep. The 16-layer architecture includes 13 convolutional layers (with 3x3 filters), 5 pooling layers (2x2 neighbourhood max-pooling) and 3 fully-connected layers. All hidden layers use rectified (ReLU) activations. The fully-connected layers reduce 4096 channels down to 1000 softmax outputs and are regularized using dropout. This form of network is referred to as VGG-16 later in this thesis.

The current (2016) winner [2] of the object detection category in the ImageNet challenge is also CNN-based. The method uses a combination of CRAFT region proposal generation [55], gated bi-directional CNN [56], clustering, landmark generation and ensembling.

Chapter 3

Convolutional object detection

In this chapter, we discuss and compare different object detection methods that utilize convolutional neural networks. In particular, we are going to look at methods that combine CNNs with *region proposal* classification. We further discuss, how the region proposals, also called *regions of interest* (RoI), are generated.

3.1 R-CNN

In 2012, Krizhevsky et al. [32] achieved promising results with CNNs for the general image classification task, as mentioned in section 2.4.6. In 2013, Girshick et al. published a method [21] generalizing these results to object detection. This method is called R-CNN (“CNN with region proposals”).

3.1.1 General description

R-CNN forward computation has several stages, shown in figure 3.1. First, the regions of interest are generated. The RoIs are category-independent bounding boxes that have a high likelihood of containing an interesting object. In the paper, a separate method called Selective Search [52], is used for generating these, but other region generation methods can be used instead. Selective Search, along with other region proposal generation techniques, is discussed in further detail in section 3.3.

Next, a convolutional network is used to extract features from each region proposal. The sub-image contained in the bounding-box is warped to match the input size of the CNN and then fed to the network. After the network has extracted features from the input, the features are input to support vector machines (SVM) that provide the final classification.

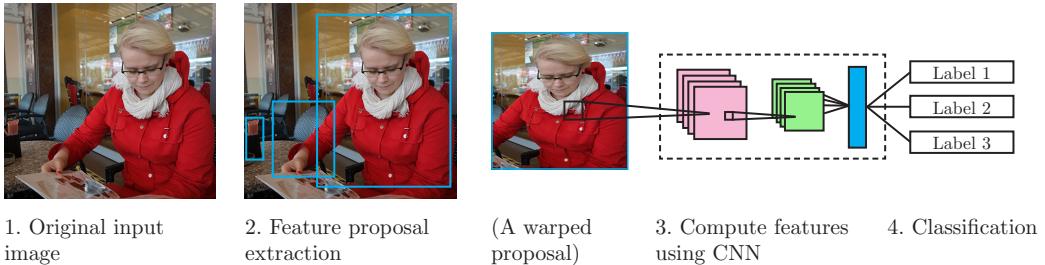


Figure 3.1: Stages of R-CNN forward computation.

The method is trained in multiple stages, beginning with the convolutional network [20]. After the CNN has been trained, the SVMs are fitted to the CNN features. Finally, the region proposal generating method is trained.

3.1.2 Drawbacks

R-CNN is an important method, because it provided the first practical solution for object detection using CNNs. Being the first, it has many drawbacks that have been improved upon by later methods.

In his 2015 paper for Fast R-CNN [20], Girshick lists three main problems of R-CNN:

First, training consists of multiple stages, as described above. Second, training is expensive. For both SVM and region proposal training, features are extracted from each region proposal and stored on disk. This requires days of computation and hundreds of gigabytes of storage space.

Third, and perhaps most important, object detection is slow, requiring almost a minute for each image, even on a GPU. This is because the CNN forward computation is performed separately for every object proposal, even if the proposals originate from the same image or overlap each other.

3.2 Fast R-CNN

Fast R-CNN [20] published in 2015 by Girshick provides a more practical method for object recognition. The main idea is to perform the forward pass of the CNN for the entire image, instead of performing it separately for each ROI.

3.2.1 General description

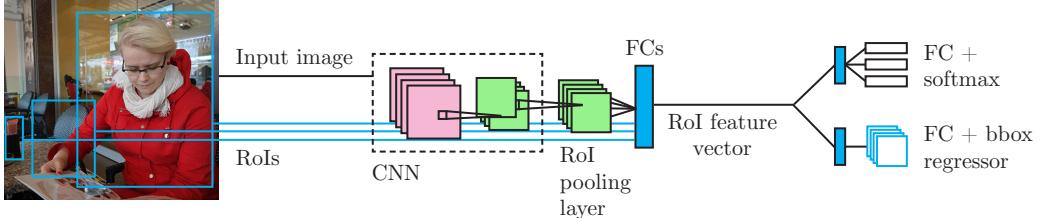


Figure 3.2: Stages of Fast R-CNN forward computation.

The general structure of Fast R-CNN is illustrated in figure 3.2. The method receives as input an image plus regions of interest computed from the image. As in R-CNN, the RoIs are generated using an external method. The image is processed using a CNN that includes several convolutional and max pooling layers.

The convolutional feature map that is generated after these layers is input to a RoI pooling layer. This extracts a fixed-length feature vector for each RoI from the feature map. The feature vectors are then input to fully-connected layers that are connected to two output layers: a softmax layer that produces probability estimates for the object classes and a real-valued layer that outputs bounding box co-ordinates computed using regression (meaning refinements to the initial candidate boxes).

3.2.2 Classification performance

According to the authors, Fast R-CNN provides significantly shorter classification time compared to regular R-CNN, taking less than a second on a state-of-the-art GPU [20]. This is mainly due to using the same feature map for each RoI.

As the detection time decreases, the overall computation time begins to depend significantly on the performance of the region proposal generation method. The RoI generation can thus form a computational bottleneck [40]. Additionally, when there are many RoIs, the time spent on evaluating the fully-connected layers can dominate the evaluation time of the convolutional layers. Classification time can be accelerated by approximately 30% if the fully-connected layers are compressed using truncated singular value decomposition [20]. This results in a slight decrease in precision, however.

3.2.3 Training

According to the original publication [20], Fast R-CNN is more efficient to train than R-CNN, with nine-fold reduction in training time. The entire network (including the RoI pooling layer and the fully-connected layers) can be trained using the back-propagation algorithm and stochastic gradient descent. Typically, a pre-trained network is used as a starting point and then fine-tuned. Training is done in mini-batches of N images. R/N RoIs are sampled from each mini-batch image. The RoI samples are assigned to a class, if their intersection over union (see section 4.6) with a ground-truth box is over 0.5. Other RoIs belong to the background class.

As in classification, RoIs from the same image share computation and memory usage. For data augmentation, the original image is flipped horizontally with probability 0.5. The softmax classifier and the bounding box regressors are fine-tuned together using a multi-task loss function, which considers both the true class of the sampled RoI and the offset of the sampled bounding box from the true bounding box.

3.3 Region proposal generation and use

To use R-CNN and Fast R-CNN, we need a method for generating the class-agnostic regions of interest. Next, we are going to discuss general principles of RoI generation, and have a closer look at two popular methods: Selective Search and Edge Boxes.

3.3.1 Overview

The aim of region proposal generation in object detection is to maximize recall i.e. to generate enough regions so that all true objects are recovered [57]. The generator is less concerned with precision, since it is the task of the object detector to identify correct regions from the output of the region proposal generator.

However, the amount of proposals generated affects performance. As mentioned in section 2.3.2, there are two main approaches to region generation: dense set generation and sparse set generation.

Dense set solutions attempt to generate by brute force an exhaustive set of bounding boxes that includes every potential object location [51]. This can be achieved by sliding a detection window across the image. However, searching through every location of the image is computationally costly and requires a fast object detector. Additionally, different window shapes and

sizes need to be considered. Thus, most sliding window methods limit the amount of candidate objects by using a coarse step-size and a limited number of fixed aspect ratios.

Most region proposals in a dense set do not contain interesting objects. These proposals need to be discarded after the object detection phase. Detection results can be discarded, if they fall behind a predefined confidence threshold or if their confidence value is below a local maximum (*non-maximum suppression*) [36].

Instead of discarding the regions *after* the object detection stage, the region proposal generator itself can rank the regions in a class-agnostic way and discard low-ranking regions. This generates a *sparse set* of object detections [55]. Similarly to dense set methods, thresholding and non-maximum suppression can be implemented after the detection phase to further improve the detection quality. Sparse set solutions can be grouped into *unsupervised* and *supervised* methods.

One of the most popular unsupervised methods is Selective Search [51] (see section 3.3.2), which utilizes an iterative merging of superpixels. There are also other methods that use the same approach [57]. Another approach is to rank the objectness of a sliding window. A popular example of this is Edge Boxes [57] (see section 3.3.3), which calculates the objectness score by calculating the number of edges within a bounding box and by subtracting the number of edges that overlap the box boundary. There is also a third group of methods based on seed segmentation [57].

Supervised methods treat region proposal generation as a classification or a regression problem. This means using a machine learning algorithm, such as a support vector machine [55]. It is also possible to use a convolutional network to generate the regions of interest. An example of using a CNN for calculating the bounding boxes is Multi-Box [16].

Certain advanced object detection methods, such as Faster R-CNN [40] described in 3.4.1, use parts of the same convolutional network both for generating the region proposals and for detection. We call these kinds of methods *integrated methods*.

3.3.2 Selective Search

Selective Search [51] utilizes a hierarchical partitioning of an image to create a sparse set of object locations. The main design philosophy is not to use a single strategy, but to combine the best features of bottom-up segmentation and exhaustive search. The authors had three main design considerations: the search should capture all scales, be diverse i.e. not use any single strategy for grouping regions and be fast to compute.

The algorithm begins by creating a set of small initial regions using a method called Graph Based Image Segmentation [18] designed by Felzenszwalb and Huttenlocher. The method creates a set of regions called *superpixels*. The superpixels are internally nearly uniform. Combined, they span the entire image, but individually they should not span different objects.

Selective Search then continues by iteratively grouping the regions together using a greedy algorithm, beginning with the two most similar regions. Many complimentary measures are used to compute the similarity. These measures consider colour similarity (by computing a colour histogram), texture similarity (by computing a SIFT-like measure), size of the regions (small regions should be merged earlier) and how well the regions fit together (gaps should be avoided). The grouping phase ends when every region has been combined.

The hypothetical object locations thus generated are then ordered by the likelihood of the location containing an object. In practice, the locations are ordered based on the order in which they were grouped together by the different measures. A certain element of randomness is added to prevent large objects from being favoured too much. Lower-ranking duplicates are removed.

Both the region generating method and the similarity measures were selected to be fast to compute, making the method fast in general. In addition to using diverse similarity measures, the search can be further diversified by using complementary colour spaces (to ensure lighting invariance) and using complementary starting regions.

3.3.3 Edge Boxes

As the name suggests, Edge Boxes [57] is based on detecting objects from edge maps. The main contribution of the authors of the method is the observation that the number of edge contours wholly enclosed by a bounding box is correlated with the likelihood that the box contains an object.

First, the edge map is calculated using a method by the same authors called Structured Edge Detector [15]. Then, thick edge lines are thinned using non-maximum suppression. Instead of operating on the edge pixels directly, the pixels are grouped using a greedy algorithm. An affinity measure is devised to calculate whether edge groups are part of the same contour.

The region proposals are found by scanning the image using the traditional sliding window method and calculating an objectness score at each position, aspect ratio and scale. The score is calculated by summing the edge strength of edge groups that lie completely within the box and subtracting the strength of edge groups that are part of a contour that cross the

box boundary. Promising regions are then further refined.

3.4 Advanced convolutional object detection

In the experimental section of this thesis, we will focus mostly on Fast R-CNN. There are, however, several state-of-the-art algorithms with an improved computation time or accuracy. Next, we will describe two of these algorithms. See also chapter 7 for discussion of improvements of convolutional object detection.

3.4.1 Faster R-CNN

Faster R-CNN [40] by Ren et al. is an integrated method. The main idea is to use shared convolutional layers for region proposal generation and for detection. The authors discovered that feature maps generated by object detection networks can also be used to generate the region proposals. The fully convolutional part of the Faster R-CNN network that generates the feature proposals is called a *region proposal network* (RPN). The authors used Fast R-CNN architecture for the detection network.

A Faster R-CNN network is trained by alternating between training for RoI generation and detection. First, two separate networks are trained. Then, these networks are combined and fine-tuned. During fine-tuning, certain layers are kept fixed and certain layers are trained in turn.

The trained network receives a single image as input. The shared fully convolutional layers generate feature maps from the image. These feature maps are fed to the RPN. The RPN outputs region proposals, which are input, together with the said feature maps, to the final detection layers. These layers include a RoI pooling layer and output the final classifications.

Using shared convolutional layers, region proposals are computationally almost cost-free. Computing the region proposals on a CNN has the added benefit of being realizable on a GPU. Traditional RoI generation methods, such as Selective Search, are implemented using a CPU.

For dealing with different shapes and sizes of the detection window, the method uses special *anchor boxes* instead of using a pyramid of scaled images or a pyramid of different filter sizes (see section 7.2 for discussion of scale invariance). The anchor boxes function as reference points to different region proposals centred on the same pixel.

3.4.2 SSD

The Single Shot MultiBox Detector [36] (SSD) takes integrated detection even further. The method does not generate proposals at all, nor does it involve any resampling of image segments. It generates object detections using a single pass of a convolutional network.

Somewhat resembling a sliding window method, the algorithm begins with a default set of bounding boxes. These include different aspect ratios and scales. The object predictions calculated for these boxes include offset parameters, which predict how much the correct bounding box surrounding the object differs from the default box.

The algorithm deals with different scales by using feature maps from many different convolutional layers (i.e. larger and smaller feature maps) as input to the classifier. Since the method generates a dense set of bounding boxes, the classifier is followed by a non-maximum suppression stage that eliminates most boxes below a certain confidence threshold.

3.5 Comparing the methods

Above, we described how Fast R-CNN is faster and more accurate than regular R-CNN. But how does Fast R-CNN perform compared to the above-mentioned advanced methods?

Liu et al. [36] compared the performance of Fast R-CNN, Faster R-CNN and SSD on the PASCAL VOC 2007 test set (see section 4.5 for discussion of the standard benchmarks). When using networks trained on the PASCAL VOC 2007 training data, Fast R-CNN achieved a mean average precision (mAP) of 66.9 (see section 4.6 for discussion of evaluation methods). Faster R-CNN performed slightly better, with a mAP of 69.9. SSD achieved a mAP of 68.0 with input size 300 x 300 and 71.6 with input size 512 x 512. As the standard implementations of Fast R-CNN and Faster R-CNN use 600 as the length of the shorter dimension of the input image, SSD seems to perform better with similarly sized images. However, SSD requires extensive use of data augmentation to achieve this result [36]. Fast R-CNN and Faster R-CNN only use horizontal flipping, and it is currently unknown, whether they would benefit from additional augmentation.

While the advanced methods are more precise than Fast R-CNN, the real improvements come from speed. When most of the detections with a low probability are eliminated using thresholding and non-maximum suppression (see section 4.6 for details), SSD512 can run at 19 FPS on a Titan X GPU. Meanwhile, Faster R-CNN with a VGG-16 architecture performs at 7

FPS [36]. The original authors of Faster R-CNN [40] report a running time of 5 FPS i.e. 0.2 s per image. Fast R-CNN has approximately the same evaluation speed, but requires additional time for calculating the region proposals. Region generation time depends on the method, with Selective Search requiring 2 seconds per image on a CPU and Edge Boxes requiring 0.2 seconds per image [40].

Chapter 4

Experimental setup

In this chapter, we begin discussing the experimental part of the thesis. First, we will discuss selection criteria for methods and datasets. Then we will describe the selected methods, their parameters and the selected datasets. Finally, we will discuss postprocessing and evaluation. The implementation of the methods is mostly discussed in the following chapter. However, some implementation details are also discussed in this chapter, since they influence method selection.

4.1 Overview and selection criteria

The main tasks for the experimental part were to implement a convolutional object detector, to test how well a detector trained on general data performs in a specific task and to find potential sources for improvement. These tasks as well as the practical implementation environment determined the selection criteria for the methods and datasets used for experiments.

Major problems limiting the use of deep learning methods are the availability of computing power and training data (see section 2.2.5). For this thesis, we did not have access to a server farm or a high-end GPU used for research purposes. Rather, we needed to implement the methods on an ordinary consumer laptop. Training a convolutional network from start to finish on such hardware would be enormously time consuming. Thus, we favoured methods that were established enough to have pre-trained networks available to be used as a starting point. We also favoured methods that had MATLAB implementations available.

We chose not to perform exact evaluation of execution time and concentrated on evaluating average precision (see section 4.6 for discussion of the evaluation metrics). Evaluating execution time would require a more stan-

dardized environment (and preferably several different computers of different performance levels) for providing results with scientific value. However, we do report if we found some methods impractically slow.

In addition to computing power, deep convolutional networks require a large amount of training data. Since collecting and annotating a dataset of sufficient size is laborious, most research is performed on few available datasets. For the same reasons, we ruled out collecting data ourselves for this thesis. Instead, we chose to experiment with several different available datasets.

A generic object detector should be trained on diverse data and support multiple object categories. The standard benchmark datasets (PASCAL VOC and ImageNet) provided a suitable starting point. A generic object detector should also be readily available and easy to implement. There are many available networks that have been pretrained on the standard datasets.

For evaluating the performance of the generic object detector on a specific task, we collected test data from various sources. The main criterion was that the object annotations of the test data should be approximately compatible with the object annotations of the standard benchmarks. This means that the test datasets should have classes in common with the standard benchmarks and include objects annotations as bounding boxes.

For finding potential improvements of convolutional systems, we implemented a geometric scene estimation model published in the paper “Objects in Perspective” by Hoiem et al. [26]. The model fine-tunes the probabilities of the detections based on whether the objects are located in a geometrically probable part of the image. Since convolutional networks mainly consider the local neighbourhood of objects, consideration of the whole scene provides interesting contrast and could potentially improve detection quality.

4.2 Object detection

We selected Fast R-CNN as the core method for object detection experiments. Fast R-CNN is already well-established and has implementations and pretrained networks available for several different platforms. Even though the advanced methods, such as Faster R-CNN, provide a minor increase in accuracy, their main contribution is improved speed. Since the evaluation of execution-time was mostly left outside the scope of this thesis, these methods would have provided little additional value to the experiments.

Because of the implementation environment (which is described in more detail in the following chapter), we chose to skip the initial training and use a pretrained VGG-16 network. VGG-16 is currently one of the standard

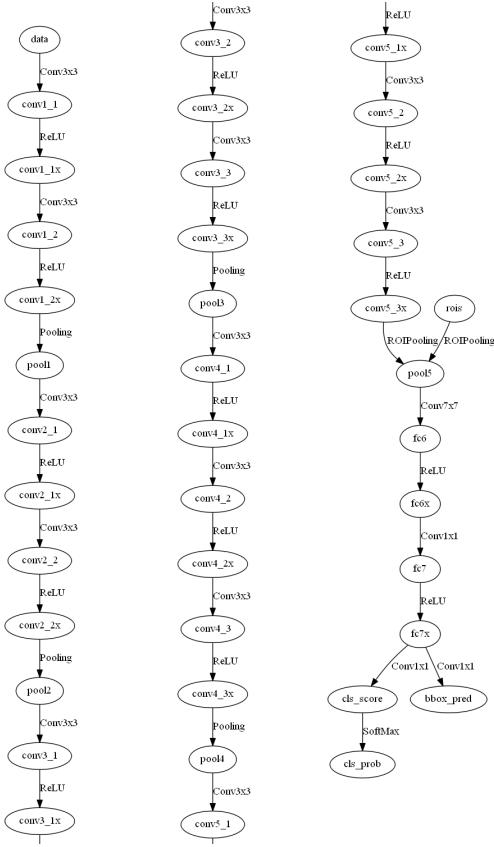


Figure 4.1: Structure of the Fast R-CNN VGG-16 network. The network has been cut into three parts to fit it to a single page. Reading order: top-down, then left-to-right.

models for implementing convolutional networks, as mentioned in section 2.4.6. In the original Fast R-CNN publication [20], VGG-16 is mentioned as an example of a “very deep network” and is used as the large network model (L) in all of the experiments. We used a pretrained network with the same network model, the structure of which is shown in figure 4.1. Most of the operations of the network are shown as separate layers. The fully-convolutional subnetwork, shown beginning at top left, includes alternating convolutional layers and ReLu activation layers. After every two or three pairs of these, pooling is performed. The filter size of each convolutional layer is 3x3 (defined as a kernel size of 3). The kernel size of the pooling layers is 2. The convolutional layers have a stride of 1 and the pooling layers have a stride of 2.

The regions of interest (RoIs) are introduced at the final pooling layer of

the fully-convolutional subnetwork. This *RoI pooling layer* selects the RoI-specific part of the activation map of the final convolutional layer and outputs this part to the fully-connected subnetwork. This network includes two fully-connected layers and two ReLu activation layers. The fully-connected layers include 4096 output parameters, reflecting the size of the final activation map. The layers use dropout regularization with a ratio of 0.5. After the second activation layer, the processing branches out to calculate the output probabilities and bounding box refinements.

4.3 Region generation

Because Fast R-CNN was chosen as the object detector, separate region generation methods were also needed. Since these were relatively easy to implement, we chose to experiment with both Selective Search [51] and Edge Boxes [57].

Next, we will discuss how the parameters settings for the methods were chosen. Both methods have parameter settings that alter the quality and quantity of the region candidates. The authors of the methods have provided a thorough study of the effect of these parameters. We selected the parameters based on their results, rather than running our own experiments using a validation set.

4.3.1 Selective Search parameters

As explained in section 3.3.2, Selective Search can use multiple diversification strategies. According to the original paper [51], adding diversification strategies generally provides better results but increases computation time. Since computation time is not evaluated in this thesis, we chose to favour quality. We chose a setting mode that the authors call “Selective Search Quality”. This includes using all four diversification strategies (explained in section 3.3.2) and five different colour spaces: *HSV*, *Lab*, *rgI* (*rg* channels of normalized *RGB* plus intensity), *H* (hue channel from *HSV*) and *I* (intensity alone).

The settings of the superpixel generation method [18] used by Selective Search also influence the output. The parameter k defines the number of initial segmentations used. The authors of Selective Search found that the algorithm was most accurate for object detection purposes with k values between 50 and 300. We selected the middle value $k = 200$. Before superpixel segmentation, the image is smoothed using a Gaussian filter, which is controlled by parameter σ . The original authors of [18] always use $\sigma = 0.8$,

which they found helpful in removing digitalization artefacts without otherwise altering the image. This setting is also used here.

4.3.2 Edge Boxes parameters

As explained in section 3.3.3, Edge Boxes [57] finds the initial region candidates using a sliding window search. The step size of the search is controlled by parameter α , which defines the intersection over union (IoU) of neighbouring boxes. The same parameter defines the step size for translation, scale and aspect ratio. After the boxes have been found and refined, they are sorted and non-maximum suppression (NMS) is performed. During NMS, boxes are discarded if their IoU with a higher-ranking box is more than β . IoU and NMS are explained in more detail in section 4.6.

α and β are the most important parameters of Edge Boxes. The authors of the method studied the effect of these parameters from the perspective of finding a suitable candidate accuracy for the object detector. Even though an IoU value higher than 0.5 is typically used as the criterion for evaluating the performance of an object detector (as the limit for determining true matches), the object detector often performs better, if the candidate objects are a closer match to true objects. Thus, the *target* IoU should be set higher than the minimum acceptable IoU.

The authors define the target IoU as δ and performed experiments with δ values of 0.5, 0.7 and 0.9. Since Fast R-CNN in any case calculates refinements of the bounding boxes [20], we decided that increasing δ above 0.9 is not needed for the present experiments. The resulting increase in the number of candidates would only slow down the algorithm. We chose a more practical value $\delta = 0.7$ as the target IoU.

α and β are set according to δ . If a higher value of δ is required, α and β are selected to output denser sets of bounding boxes around apparent objects. The authors found that a value of $\alpha = 0.65$ provided the best results for $\delta < 0.9$. The optimal value of β was determined to be $\delta + 0.05$. Thus, we chose $\alpha = 0.65$ and $\beta = 0.75$ as the parameter values.

For the minimum objectness score limit h_b^{in} (used for discarding uninteresting windows before the refinement stage), we used the default value of 0.01. We also used the default value of 10,000 as the maximum number of object proposals.

4.4 Evaluating objects in context

Fast R-CNN performs classification mostly on basis of the local neighbourhood of the object. This is partly because of the method of using region proposals and partly due to the inherent translation invariance of the convolutional network. The receptive fields reach their maximum size in the deep end of the network, where the size of the activation map has been lowered using pooling and stride. The final fully-connected layers are then allowed to make deductions across the full activations. However, the fully-connected subnetwork is typically shallow. In the VGG-16 version of Fast R-CNN, there are only two fully-connected layers. By our reasoning, the fully-connected layers merely learn to combine activations from different parts of the sub-image into object classes.

Convolutional regional object detection could potentially be improved by other methods that take the whole scene into consideration more thoroughly. One way to detect the scene is to estimate the 3D geometry of an 2D image and to use this model to segment the image into basic parts. In this section, we will explain how we combined scene detection with convolutional object detection.

4.4.1 Geometric Context

In 2005, Hoiem et al. published two papers called “Geometric Context from a Single Image” [25] and “Automatic Photo Pop-up” [24], which describe a method for 3D model reconstruction from a single image. The 3D models resemble children’s pop-up books. The image is divided into rough segments (ground, sky and vertical), which are then cut and folded into texture mapped 3D planes. These planes can be displayed from slightly different angles to alter the viewpoint of the original image.

For object detection purposes, we are more interested in the segmentation of the image, not in displaying the scene. The segmentation method of [25] is based on machine learning. The method first creates a set of superpixels using Graph Based Image Segmentation [18] (which is also used by Selective Search). The superpixels are merged into groups called *constellations*. Training data is used to estimate which superpixels are likely to belong in the same labelled group. Since the superpixels can be combined in several ways, several overlapping constellations are formed. The label of each superpixel (ground, vertical or sky) is estimated by calculating the label likelihood of each constellation that the superpixel belongs to and by calculating how homogenous the said constellations are. The vertical class is further divided

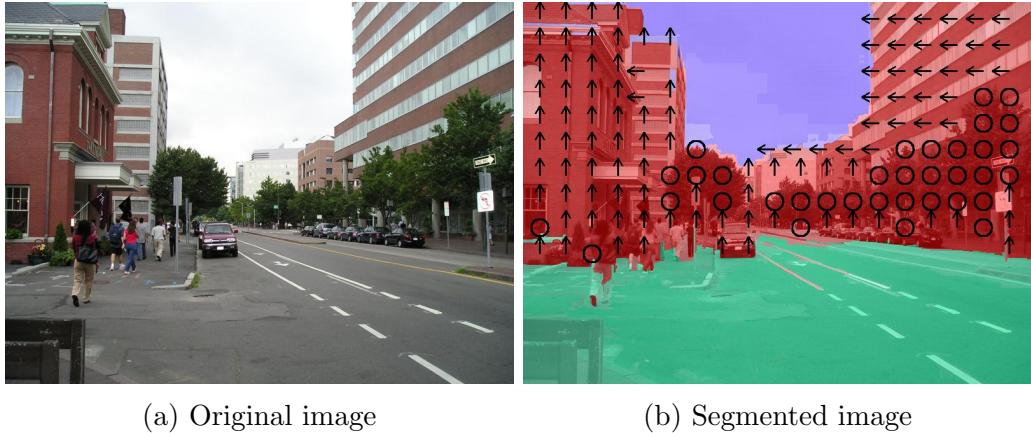


Figure 4.2: An example of image segmentation using “Geometric Context”

into *left-facing*, *centre-facing*, *right-facing*, *porous* and *solid* subclasses.

4.4.2 Putting Objects in Perspective

The 2008 publication “Putting Objects in Perspective” [26] (referred to as OiP below) by Hoiem et al. utilizes the afore mentioned geometry segmentation as an aid for object detection. The main idea of the method is to alter the probability of a detected object based on how geometrically likely the location of the object is.

The method uses training data to estimate the probabilities of different surface geometries (composed of the object surface and neighbouring surfaces) given the object classes. The method also estimates the camera viewpoint. This means estimating the camera height in the scene and the position of the horizon on the image plain. Based on these, it is possible to calculate the heights of the objects in the image. If the expected heights of the object classes are known, the probabilities of the object instances can be adjusted based on whether they fall inside the expected height range. It is also possible to detect, whether the object is located on the ground plane and adjust the probability accordingly (pedestrians and cars are expected to touch the ground and not fly in the sky).

Taken together, the surface geometry and viewpoint include a large amount of information about the probable locations of objects. An object detector is plugged into the middle of these estimators. The interactions between the viewpoint, object detections and surface geometry are considered in calculating the final inference, which is computed using a belief propagation algorithm.

4.4.3 Substituting the object detector

The object detection method used in the OiP paper is similar to methods used in the 2000s, such as [50], and is based on hand-crafted features. The authors demonstrate that geometric inference significantly increases the accuracy of this method. By substituting the object detection method for Fast R-CNN, we can study how object detection has changed in the past ten years and whether geometric inference is still useful for the current generation of object detectors. The authors describe substituting the object detector for another as “painless”. Basically, the detections need to be converted to the input format used by the OiP geometric inference function (details of this conversion are discussed in the following chapter).

Instead of performing regular non-maximum suppression (see section 4.6 for details of how NMS is used with Fast R-CNN), the detections with high overlap are grouped together before the geometric inference phase. Regular NMS discards overlapping detections and keeps only the detection with the highest confidence. By retaining all detections, the geometric inference module is free to switch the selected bounding box at a later point. This grouping operation requires implementation when substituting the object detector.

4.5 Datasets

Next, we will discuss the datasets used in the experiments. First, we will discuss the standard benchmark datasets that were used to train the pretrained Fast R-CNN object detector. Then, we will discuss additional datasets that we used for testing the object detector in practice.

4.5.1 Standard benchmarks

The standard benchmarks used for object detection are the PASCAL Visual Object Classes (VOC) challenge [17] dataset and the ImageNet large scale visual recognition challenge [43] dataset.

The main training dataset was the PASCAL VOC dataset. The PASCAL VOC challenge was organized yearly between 2005 and 2012. In 2007, the dataset included 9,963 annotated images with 20 object classes (person, 6 different animals, 6 different household objects and 7 different vehicles, including cars). The 2007 challenge was the final time the ground-truth for the test data was published. By the final year, the dataset had grown to 54,900 images, around half of which formed the test and validation sets. The amount and type of different tasks also varied yearly, but, for this thesis, we

were only interested in the object detection challenge data.

The pretrained PASCAL VOC network that we used was itself initialized on a network trained on the ImageNet dataset. The ImageNet challenge has been organized yearly since 2010. The challenge is based on a subset of the ImageNet database, which currently includes 14,197,122 images [1], 1,034,908 of which include bounding box annotations. In 2012, the challenge was organized concurrently with the PASCAL challenge. Since 2013, the challenge has included an object detection challenge, which currently [2] includes 200 fully labelled categories (including the same categories as the PASCAL VOC set). The training set consists of 456,567 images and the validation set consists of 21,121 images.

4.5.2 Test data

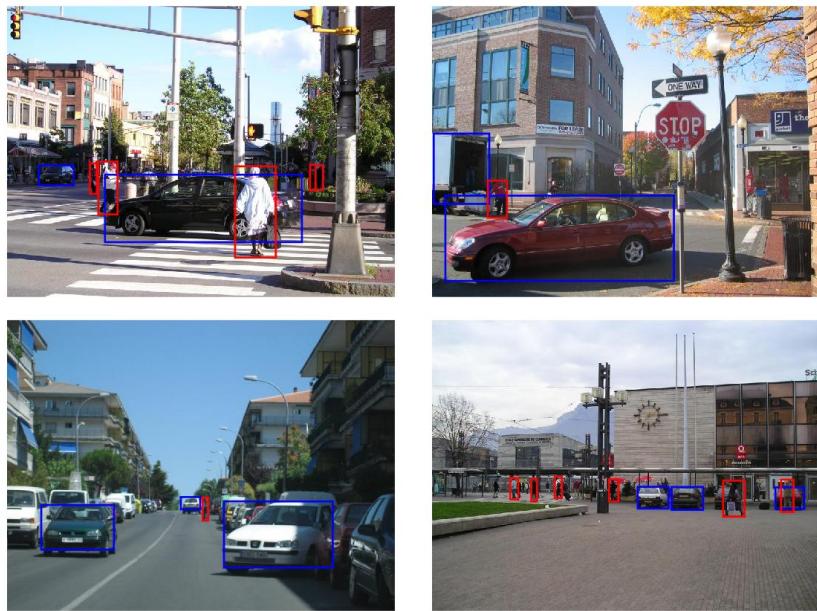


Figure 4.3: Example images from the “Objects in Perspective” dataset, with car objects marked in blue and pedestrian objects marked in red.

Traffic related object detection, such as pedestrian and vehicle detection, are popular research topics in computer vision. Such objects are annotated in many publicly available collections of street view data. This provided an excellent source of data to test the generic object detector on. Cars and persons are also annotated in the benchmark datasets, providing cross-compatibility.

First test dataset that we used is the same dataset that was used in the “Putting Objects in Perspective” (OiP) publication [26]. The set includes 600 test set images (422 of which include valid objects) and 60 validation set images selected from the LabelMe database. Objects included in the images have been marked with bounding boxes. We used the 60 images of the validation set as our first test set. This part includes 126 cars and 84 pedestrians.

Examples of the OiP data is shown in figure 4.3. There are two object classes: cars (including regular passenger cars as well as vans and trucks) and pedestrians (upright persons). The PASCAL VOC dataset includes approximately matching categories. The vehicles of the PASCAL dataset include (passenger) cars and buses as separate classes, but no other four-wheeled vehicles. However, it is a reasonable hypothesis that a detector trained on these two classes would also learn to recognize the more general car class of the OiP dataset. The other OiP class, pedestrian, is strictly speaking a subclass of the PASCAL person category. It is, however, reasonable to assume that majority of people in street view images are pedestrians. Possible exceptions include cyclists and people located inside vehicles or buildings.



Figure 4.4: Examples images from the Street Scenes dataset, with car objects marked in blue and pedestrian objects marked in red. The polygonal object annotations have been converted to maximal bounding boxes.

The Street Scenes dataset [10] provided by Massachusetts Institute of

Technology includes 3,547 images that have been labelled for 9 different objects (car, pedestrian, bicycle, building, tree, road, sky, sidewalk and store). We used 60 images selected from this set as our second test set. This part of the set includes 131 cars and 51 pedestrians.

Object locations are annotated using polygons that surround the object tightly. The polygons can be converted to bounding box form by finding the minimum and maximum values for both x and y co-ordinates. The car and pedestrian classes resemble the similarly named OiP classes. The car class includes all motorized vehicles (with more than two wheels) that are more than 64 pixels in width. Pedestrians include persons who are walking or standing and more than 32 pixels in height. Both objects are labelled only if they are 75 % visible.

Figure 4.4 shows examples of the Street Scenes data. The polygons have been converted to bounding boxes. In the top left image, we can see that only the larger cars in the foreground are annotated. In the top right image, the cyclist is not annotated (since she is not a pedestrian). In the bottom right image, only select individuals out of the group of pedestrians are annotated.

4.6 Evaluation metrics

The object detections are evaluated using the standard *Intersection over Union* (IoU) metric [57]. The bounding boxes of the detections are rarely a pixel-perfect match to the ground-truth boxes. In practice, we are interested in finding detections that are merely close enough to be called true positive matches. IoU is calculated by dividing the intersection (the overlap) of the detection box and the ground-truth box by the area of their union.

Generally, an IoU score over 0.5 is counted as a true positive detection [57] [17] and this definition is used in this thesis as well. There can be only one true positive match per each ground-truth box. If several detections match a single ground-truth, the box with the highest likelihood is selected as the true positive match and the other detections are marked as false positives. Detections with no matching ground-truth box are marked false positives as well. Ground-truth boxes with no matching detections are called false negatives.

The performance of the object detection algorithm is evaluated by calculating the *precision-recall curve* and the *interpolated average precision* of the detections for each class, similarly to the PASCAL VOC challenge [17]. We calculate the curve directly for an entire dataset (i.e. over all objects in the test data), instead of calculating an average curve sampled at certain recall values from each image. We create a combined list of detections for each

class (with the detections marked as true or false positives), sorted by the estimated class probability. Precision and recall are calculated cumulatively at each detection’s location in the list. Precision is defined as the number of true positive detections retrieved, divided by the total number of true and false positives retrieved. Recall is defined as the number of true positive detections retrieved, divided by the total number of positive samples known from the ground-truth data.

An interpolated (monotonically decreasing) curve is computed for visualization purposes and for calculating the interpolated average precision. The interpolated curve is created by replacing the actual precision values with maximum precision values from the remaining part of the curve. Regularly, average precision is calculated by computing a discrete approximation of the area under the precision-recall curve (by calculating a cumulative sum of all precisions multiplied by recall delta-values). In the PASCAL VOC challenge, interpolated average precision is used instead, which reduces the impact of perturbations caused by slight ordering differences [17]. We use the same method to produce comparable results. In the following chapters, the term average precision is to be understood as interpolated average precision.

The Fast R-CNN evaluations are not discarded based on a probability threshold (as they would be in a practical application), because we are interested in seeing whether the detection method ever reaches complete recall (as the precision decreases). However, for the Fast R-CNN detections, non-maximum suppression (NMS) is performed before evaluation. NMS is performed by discarding detections that have an IoU larger than a pre-set parameter value with a higher-probability detection. The purpose is to remove multiple detections of the same object before evaluation. NMS is performed using several parameter values in order to find the optimal IoU threshold.

For the geometric inference, NMS is not performed as such, as explained previously. Instead, detections with high IoU are grouped together and the highest scoring detection of the group is selected after performing the inference. We also tested different values of this IoU grouping parameter.

Chapter 5

Implementation

In this chapter, the practical implementation of Fast R-CNN and peripheral methods is discussed along with problems encountered and lessons learned during implementation.

5.1 Environment

The implementation environment was a ASUS X550L laptop computer with an Intel Core i5-4210U 1.70 GHz CPU, 8 GBs of RAM and NVIDIA GeForce 840M GPU. The operating system was Windows 10.

The main software tool was MATLAB 2014b. The object detection system and its related methods were implemented as a combination of pre-existing and self-programmed MATLAB tools.

5.2 MatConvNet and Fast R-CNN

We implemented the convolutional network using MatConvNet [5], which is a MATLAB toolbox developed specifically for this purpose. Another alternative would have been Caffe deep learning framework [31], which is more popular, more versatile and also has a MATLAB interface. However, MatConvNet provided all the required functionality and is easier to install.

MatConvNet is a collection of MATLAB functions that implement different building blocks of a convolutional network. A CNN wrapper combines these blocks into a complete network. The wrapper is either a light-weight SimpleNN wrapper or a more versatile DagNN wrapper.

The DagNN wrapper creates a MATLAB object that includes functions providing access to training and evaluation of the network and parameters

that store the network structure and weights. The object can be stored on disk, facilitating sharing of pretrained networks.

The MatConvNet website provides pretrained networks that have been converted to the MatConvNet format from original implementations and that match the originals up to a certain numerical precision. We used a DagNN VGG-16 implementation of Fast R-CNN that has been pretrained on the PASCAL VOC 2007 [17] dataset. The network also utilizes Imagenet Challenge 2012 [43] data, since the Fast R-CNN network has been initialized using a network trained on the ImageNet data.

We created a MATLAB script for calling the CNN functions and for implementing pre- and postprocessing not provided by MatConvNet. The script begins by loading the network, the images and the bounding boxes from disk (see section 5.3 for bounding box generation). The network is put into test mode, which disables learning. The bounding boxes are then converted to the coordinate format used by the network. Next, the images and boxes are scaled to match the input size of the network (600x800 pixels with 3 colour channels). The images are also preprocessed by subtracting the average colour value.

After the data has been prepared, the evaluation function of the network is called using the image and the boxes as input arguments. After evaluation, the class probabilities (softmax outputs) and refinement delta values for each box are “squeezed” out of the network. Next, the bounding box refinements are calculated by combining the original values and the delta values. The refined boxes and probabilities for the interesting classes (cars, busses and persons) are stored in a struct. The car and bus classes are combined in evaluation to create an approximation of a more general car class, which includes larger cars as well as passenger cars. Finally, the detections are stored on disk.

5.3 Selective Search and Edge Boxes

A MATLAB implementation of Selective Search is published by the authors of the original paper [51]. The package includes a C++ implementation of an anisotropic gaussian filter and the superpixel generation method [18]. After compiling these, no additional setup is needed. The main function of Selective Search requires, in addition to the image data, certain initial parameters as input. These are set as explained in the previous chapter. After the initial bounding boxes have been calculated, duplicates are removed using a separate function.

Edge Boxes is similarly available as MATLAB code by the method’s [57]

authors. MATLAB image processing toolbox and Piotr’s computer vision MATLAB toolbox [7] are required in addition to the main code. Several functions used by Edge Boxes are only provided as C++ code, which is compiled during setup. The package also includes a MATLAB implementation of Structured Edge Detector (used for calculating edge strengths and orientations), with a pretrained edge detection model for RGB images, which is suitable to the datasets used in this thesis. The Edge Boxes main function takes this model, the method parameters (explained in the previous chapter) and an image as input.

For both methods, MATLAB scripts were created to batch process selected parts of the datasets and to store the computed bounding boxes on disk, to be used by the object detection method.

5.4 Geometric inference

MATLAB implementations of the “Putting Objects in Perspective” (O iP) [26] and “Geometric Context” [25] methods are available at [8]. The Geometric Context package includes a C++ implementation of the superpixel generation method [18]. Implementing O iP requires both packages. The O iP package includes a modified version of the Automatic Photo Pop-up code, which is substituted for the directory processing function of the Geometric Context package. This function is input a directory of image files, superpixels computed for the said files and pretrained classifiers for the image segments. The function then segments the images in the directory and stores them in an output directory.

The actual geometric inference was implemented by modifying a script included in the O iP package. The script first loads the image segmentations from disk and stores them in a struct. Next, the probabilities of the different geometric segments (given an object) are loaded. These are provided in the package for cars and pedestrians. Then, the viewpoint priors are initialized. For these, we used the provided default distributions (most likely camera height is 1.67 m and most likely horizon position is in the middle of the image).

Next, the Fast R-CNN detections are loaded from disk and converted into the “candidates” format used by the O iP implementation. Implementing this conversion is required when substituting the object detector used by the original O iP paper. In practice, the conversion mostly involves copying the detections into a suitable struct format. As explained in the previous chapter, the O iP method expects that non-maximum suppression has not been performed on the detections. Rather, as part of the detection conversion, the

overlapping detections are grouped together into a combined detection that initially has the same probability as the highest-confidence bounding box in the group. Selection of the bounding box can change as part of the geometric inference.

Then, we call the inference function provided in the OiP package. The function takes as input the detections in the candidates format, the image segmentations and the viewpoint priors. The function outputs new candidates and the probability distributions of horizon location and object heights. We convert the new candidates back into the detection format used previously and store the results on disk, ready to be evaluated.

5.5 Evaluation and visualization

The scripts for calculating the precision-recall curves and average precision mostly used our own code, except for NMS, which used a implementation by Tomasz Malisiewicz provided with MatConvNet [5]. Further auxiliary functions were also written for visualizing an image with its detections and for printing false positive and false negative detections.

Certain processing steps were dataset-specific. For instance, in the OiP data, cars have a minimum height of 14 pixels and pedestrians have a minimum height of 36 pixels. In the Street Scenes dataset, cars have a minimum width of about 64 pixels and persons have a minimum height of 32 pixels. Detections smaller than these were discarded.

5.6 Challenges and additional details

Since deep learning is an emerging field, there are no straightforward “out of the box” solutions for implementation. Most of the useful tools are aimed for specialists and researchers. Installing and testing these tools before the actual implementation provided additional challenge.

Implementing GPU acceleration with MatConvNet requires that the toolbox is installed together with CUDA, which is a parallel computing platform developed by NVIDIA. CUDA can be further supplemented with cuDNN, which is a CUDA deep neural network library. MatConvNet requires a GPU with *compute capability* (measurement used by NVIDIA for ranking their graphics hardware for CUDA use) greater than 2.0 [5]. The GeForce 840M used for implementation has a compute capability of 5.0 [6].

Thus, it would have been possible to compute (at least the convolutional part) of the object detection pipeline on a GPU. However, compiling MatCon-

vNet for GPUs and setting up CUDA proved to be time consuming. One particular problem was the mutual version requirements that the different tools (including MATLAB) had. On a side note, similar version requirements prevented further experiments with Caffe.

Since it was difficult to predict how much additional work setting up GPU acceleration would have required and since runtime evaluation was outside the scope of this thesis in any case, we chose to perform all computations on a CPU once we had a working system set up.

Ultimately, the CNN was not the most computationally intensive part of the experiments. Most time-consuming part was calculating the geometry segmentations, which required approximately 10 to 20 minutes for each image. For the Street Scenes data, it was necessary to downscale the original 1280x960 images to 640x480, since using the original image size the segmentation tended to never terminate or take several hours. Downscaling presumably did not affect detection performance, since the downscaled images were close to the CNN input size.

Minor challenge was converting the images and bounding boxes between different datatypes and formats. For instance, it was necessary to scale the images to the input size of the VGG-16 network. Since the bounding boxes were generated for the original image size, the boxes needed to be scaled as well. The boxes generated by Selective Search and Edge Boxes were incompatible with each other and incompatible with the ground truth boxes and the input types of the functions used for calculating IoU and NMS. Individually these sorts of conversions were trivial, but together they took up surprisingly large amount of time.

Chapter 6

Evaluation

In this chapter, we will present the results of the experiments. First, we will provide general results with Fast R-CNN and compare the two region generation methods. Then, we will discuss the effects of non-maximum suppression parameter setting. Finally, we will evaluate and discuss the geometric inference method.

6.1 Fast R-CNN with different regions

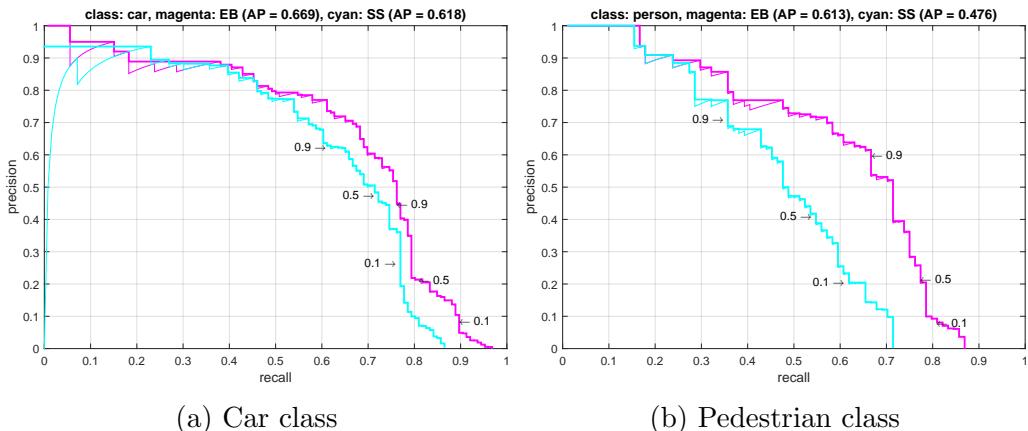


Figure 6.1: Precision-recall curves of Fast R-CNN detections of OiP data using Selective Search (cyan) and Edge Boxes (magenta) as region generation methods

Figure 6.1 shows the precision-recall curves and average precisions of Fast R-CNN detections of the OiP dataset. Before evaluation, non-maximum suppression was performed with $\text{IoU} = 0.45$ (see section 6.3 for parameter selec-

tion). The detections were computed using two different region generation methods: Selective Search (cyan) and Edge Boxes (magenta). As we can see, Edge Boxes performs better for both classes (increase in average precision 0.0510 for cars and 0.1370 for pedestrians). Note, however, that both methods use preselected parameters. It is feasible that fine tuning the parameters would provide better results for either or both methods.

In general, the Fast R-CNN detection results are quite good, keeping in mind that the convolutional network has been trained on a completely different dataset, which does not include many traffic-related training images. For cars, up to 40% of correct instances are recalled with 90% precision. If the method were to be used in a practical application, we would need to select some confidence threshold, above which we treat the detections as trusted ones. Three examples of threshold locations (0.1, 0.5, 0.9) are shown next to the curves. As we can see, Edge Boxes provides more results in each case, with better recall but lower precision. This is probably due to Edge Boxes returning many more candidate boxes (on average, 8000 boxes for the OiP images, versus 1000 boxes returned by Selective Search).

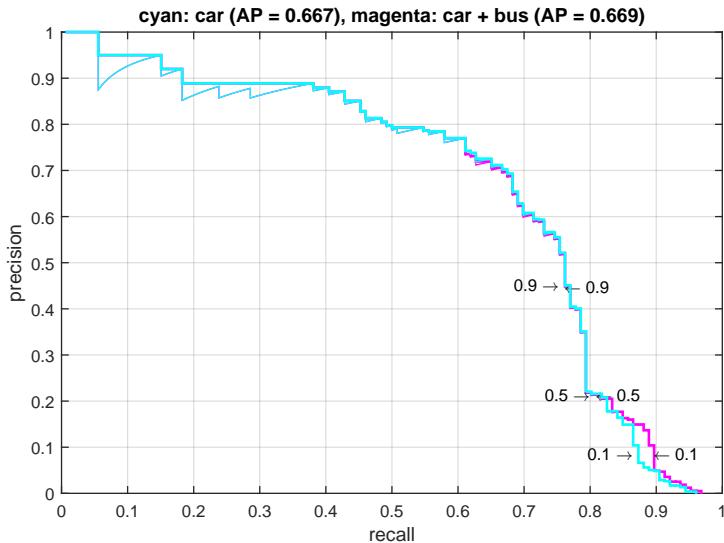


Figure 6.2: Adding the bus class to car detections slightly improves accuracy.

As explained above, the detections for cars are a combination of the car and bus classes of PASCAL VOC because cars are defined more broadly in the test datasets. We also performed the same experiments using only the car class of PASCAL VOC. This resulted in a very slight drop in average precision (0.667 vs. 0.669). The corresponding precision-recall curves are

displayed in figure 6.2. We can see that the bus detector only has an effect at the difficult end of the curve. Still, the differences do show that the differences between passenger cars and busses are understood by the object detector and substantial enough to make a difference, even if it is slight.

6.2 False positives and false negatives

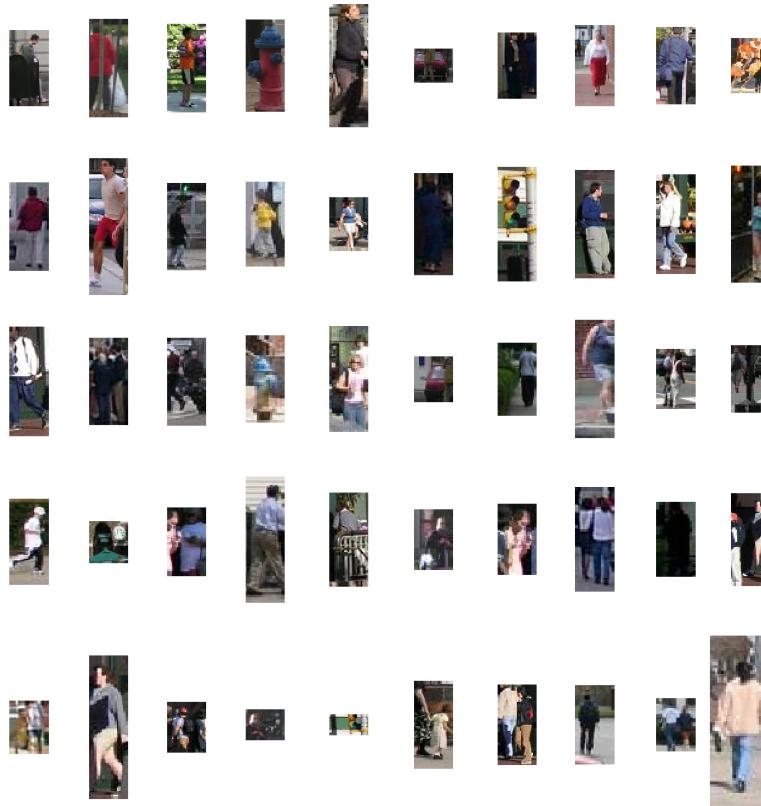


Figure 6.3: Top 50 false positive person detections from the OiP dataset.

Figure 6.3 shows the first 50 false positive person detections from the OiP dataset (using Edge Boxes and Fast R-CNN). By visual examination, about 40 of the boxes contain humans. The rest are water posts, traffic signs, cones and unknown objects. Apparently, Fast R-CNN performs significantly better than the precision-recall curve indicates. The object detector detects unannotated humans. Most of these are pedestrians who are partly occluded

or stand too close to other humans. The presence of non-human objects is probably due to a lack of hard negative examples.



Figure 6.4: False negative cases from the OiP dataset.

The corresponding false negative detections are displayed in figure 6.4. These are annotated pedestrians that were never found by Fast R-CNN. Some of the cases are people standing in the shadows. These samples are difficult to detect from the complete image, even by humans. A few others are high-contrast subjects standing in front of similarly coloured background. For the last two, there is not a clear immediate explanation for why the algorithm missed them.

The results for the car class are similar. For brevity, they are not presented here.

6.3 Non-maximum suppression

Figure 6.5 shows the effect of tuning the strictness of non-maximum suppression. The curves display the average precision of car detection from the OiP data. On the right end of the curve, where $\text{IoU} = 1$, we do not perform NMS at all (except for removing exact duplicate boxes, which should have happened already). As we move to the left on the curve, the IoU parameter decreases, which means that we begin removing detections with decreasing overlap with high-confidence detections.

We can see that Edge Boxes benefits from NMS from “both sides”, in that both strict and lenient suppression degrades the results. On the other hand, Selective Search shows improvement starting at around $\text{IoU} = 0.6$, but after that, decreasing the parameter value does not provide significant changes in accuracy. This is, perhaps, due to Selective Search outputting less results and performing less well. At $\text{IoU} = 0.6$, most of the distracting

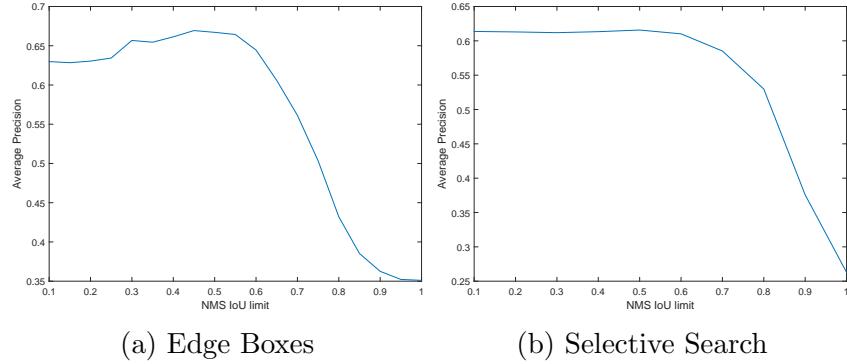


Figure 6.5: The effect of altering the NMS IoU parameter on the average precision of Fast R-CNN detections.

secondary detections have been removed. By lowering the limit further, NMS theoretically should, at some point, start removing overlapping true positive detections and lowering precision. However, in the Selective Search results these boxes are missing for some reason.

The results for person detection were similar. Based on these results, $\text{IoU} = 0.45$ was selected as the optimal NMS value for the experiments.

6.4 Geometric inference

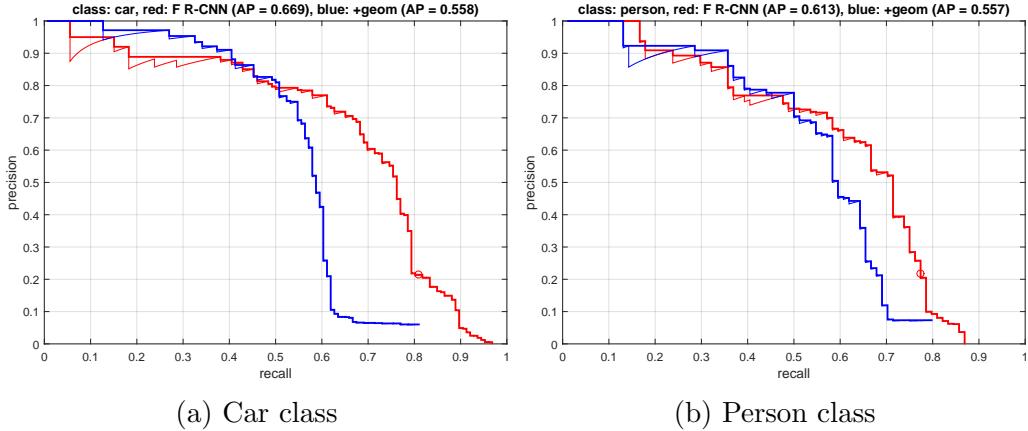


Figure 6.6: Comparison of plain Fast R-CNN (red) and OiP + Fast R-CNN (blue) using OiP dataset

Figures 6.6 and 6.7 show the effect of the above described geometric inference method on object detection. In figure 6.6, the precision-recall curve

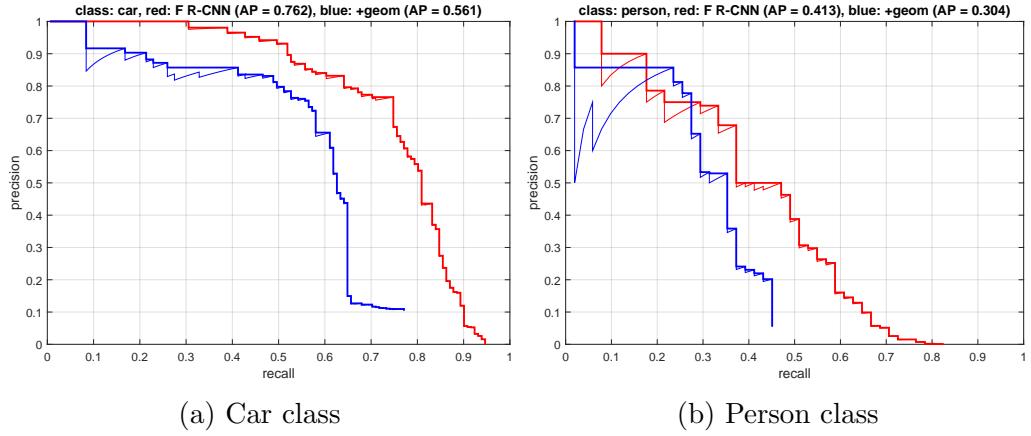


Figure 6.7: Comparison of plain Fast R-CNN (red) and OiP + Fast R-CNN (blue) using Street Scenes dataset

of Fast R-CNN detections (red) of the OiP data is shown in comparison with the curve of the same results after geometric inference (blue). Figure 6.7 shows similar comparison for the Street Scenes data.

As above, the Fast R-CNN detections have been postprocessed using NMS with IoU threshold = 0.45. Since the geometric inference method does not use regular NMS (see section 4.4.3), it is not meaningful to perform a comparison using the same threshold value for both methods. Instead, we chose the optimal threshold values for both NMS and the grouping method individually and compared the results. Figure 6.8 shows, how the person detection precision of the geometric method changes as we alter the grouping IoU threshold. Based on these results, we selected 0.3 as the optimal grouping threshold, and the precision-recall curves were drawn using this value. For performance reasons, Fast R-CNN detections with probability under 0.01 were filtered out before geometric inference. This explains why the curve stops slightly before the Fast R-CNN curve.

The precision-recall curves clearly show that, at least on these datasets

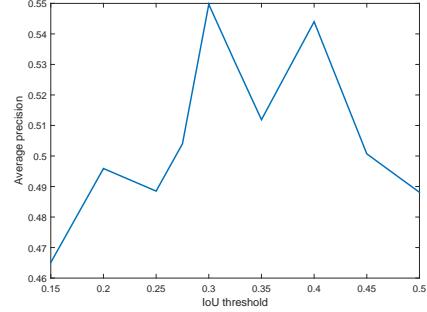


Figure 6.8: The effect of altering the grouping IoU threshold parameter on average precision after geometric inference.

and annotations, the geometric inference mostly degrades the performance of Fast R-CNN. Car detection of the OiP dataset is the only case where the blue curve is initially above the red curve, meaning that the most trusted results are better after geometric inference. Note here that since the NMS-like grouping method can alter the bounding box selection, the best results do not necessarily consist of the same boxes.

However, by visual inspection of individual results, we can clearly find examples of geometric inference working as intended. Some detection examples are displayed in figure 6.9. Here, detections are only visualized if their probability is above 0.5. The ground truth boxes are shown in a darker colour. In the top image (taken from the OiP dataset), we can see how geometric inference correctly lowers the score of the tarpaulins (hanging from the side of the building) below the threshold. Some of the cars and the single pedestrian are detected correctly by both methods. The yellow car is not detected by either method, probably because, being occluded by other objects, it is not detected as a candidate object in the first place.

However, the geometric model also brings forth incomplete additional detections of cars, which are not annotated in the original data (there are, in fact, not that many cars in the image). This illustrates typical behaviour of the geometric method. Fast R-CNN already works quite well in determining the object classes, but due to occlusions and incomplete annotations, there are many moderately probable partial detections that are not counted as true positives in evaluation. If these detections also happen to be geometrically plausible, the geometric inference method increases their probability even further. So, while the method correctly demotes some false positive detections, at the same time, it promotes many others that are geometrically in the correct place, but are in fact incorrect (for example traffic signs and water posts mistaken for humans), part of an object that has been already detected or are not annotated at all. The presence of large groups or lines of people and cars often results in detections that are both partial (because they are occluded) and not annotated in the original data, and these are promoted by the geometric model.

With better annotations we could perform a more meaningful comparison. The bottom image of figure 6.9 shows some interesting cases of problems caused by mismatches in the class definitions. First example is the cyclist, who is not counted as a pedestrian (as already mentioned). There is also another unannotated person in the image, who is standing behind the open door of the lorry. Fast R-CNN (quite remarkably) does detect him as a person, but the geometric inference module demotes the detection, presumably because the person is not standing on the ground (working as intended and, this time, matching the annotation). Also, both methods only detect the

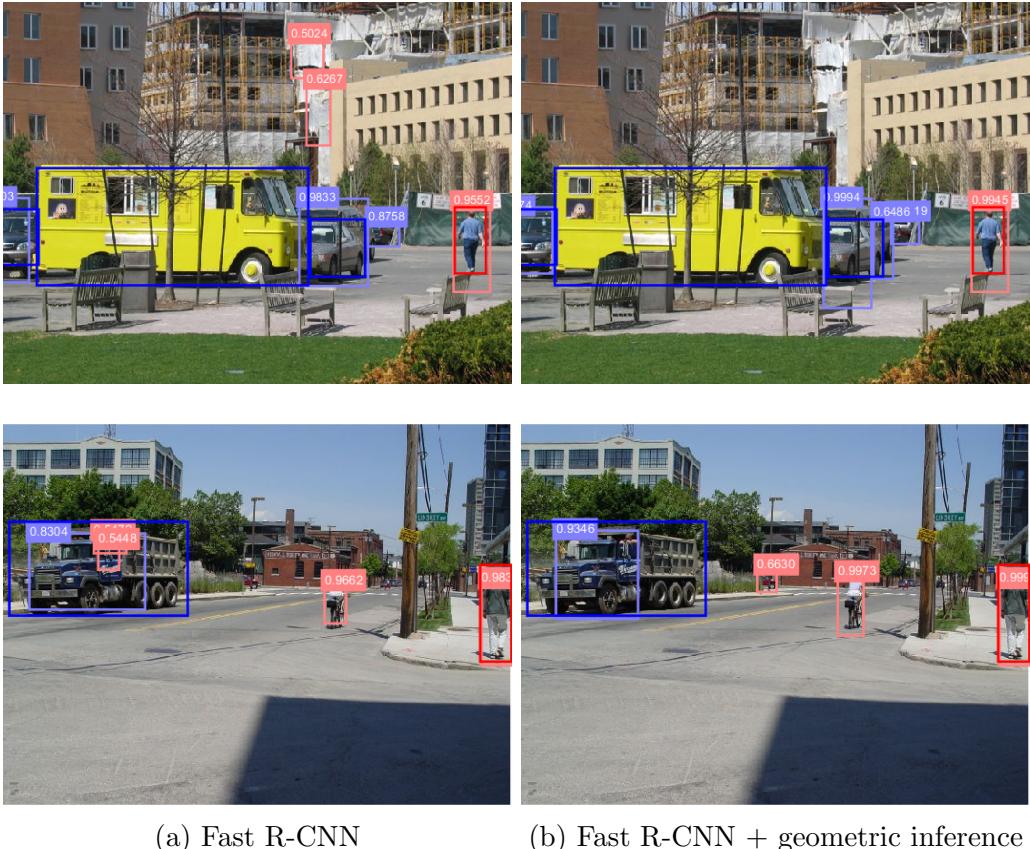


Figure 6.9: Detection examples. Person class is marked with red and car class is marked with blue.

front end of the lorry, mistaking it for a smaller car. This is because the car detections are a combination of passenger car and bus detections. Further in the background, the geometric inference incorrectly marks an occluded car as a person. The height of the object is probably right for a person. All in all, this is a type of image that is very difficult to classify unambiguously given the slight differences in class definitions. Fortunately, most of the images in the datasets include clearer examples of the different classes, making meaningful evaluation of the results possible.

In short, the geometric inference method does not improve the average results of Fast R-CNN “out of the box”, but certain cases where the method does work point to improvements that could be implemented in either convolutional object detection or in geometric inference. In the next chapter, we will discuss various improvements to convolutional object detection.

Chapter 7

Discussion

In this chapter, we will analyse our results further and discuss, with references to the literature, how convolutional object detection can be refined.

7.1 Overlap

Overlapping objects and bounding boxes clearly cause problems for the tested object detection pipeline. First of all, overlap of the true objects makes it difficult to place the bounding boxes unambiguously. The bounding boxes are selected by the region generation method and, typically, they should surround the complete object. However, if the object is occluded, we cannot know the dimensions of the object before we have classified it.

It depends on the application whether it is important to detect multiple instances of the same object close together in the first place. A self-driving car should start breaking the instant it detects the presence of a person in the middle of the road. In this case, it is not crucial to know immediately whether the detection consists of one or two humans (although it would be preferable to know it before long). On the other hand, if the task is to count the number of participants in a public event, we need to be able to confidently tell people apart.

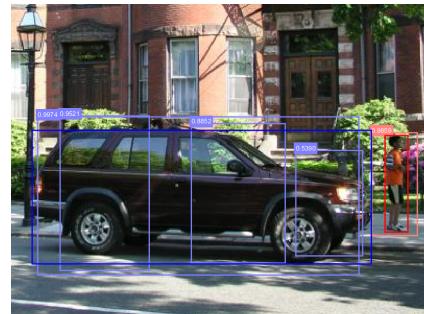


Figure 7.1: Multiple detections of the same car not removed by NMS.

The tested convolutional object detector is often good enough to classify an occluded object correctly from a partial object. However, recognizing partial objects can cause problems. Non-maximum suppression (and the related grouping method) was used in the experiments to remove multiple detections of the same object. In figure 7.1, we see examples of multiple car detections that are not removed by NMS. The partial detections are small enough compared to the entire car to provide a low intersection over union score. Fast R-CNN gives these detections a high probability value, which means that the partials can outrank whole cars that are more difficult to detect.

Lowering the IoU parameter of NMS further does not help, since it degrades overall performance. Instead, the small detections should be removed, either in the region generation stage already or by an alternative post-processing method. In the case of this single image, removing detections that are (almost) completely enclosed by a detection of the same object class would increase average precision. However, this principle does not hold universally, since it is possible to imagine examples where multiple smaller objects are enclosed by a parent object of the same class. To distinguish smaller objects from partial detections of the same object, we would need more information. Human observer already knows which objects can be enclosed in themselves, which suggests that the problem could be solved by high level reasoning. Alternatively, the convolutional network could be trained to estimate the “partialness” of an object.

At present, Fast R-CNN does attempt to adjust the dimensions of the bounding box after classification. Clearly, this does not help if the initial bounding box is nowhere near the size of the complete object. Further iterative adjusting of the box is currently too computationally costly, since it would require evaluating the convolutional network several times.

Another problem with NMS stems from sorting out which overlapping detections belong to the same object and which belong to different objects. NMS may remove true detections of neighbouring objects, if the true objects in question have a high overlap.

One suggested solution [12] is to use a “soft” version NMS instead of regular NMS. In soft NMS, the confidence of the overlapping boxes is lowered instead of discarding the boxes completely. The authors of soft NMS mention that regular NMS usually outperforms competing methods when using average precision as the evaluation metric. However, soft NMS outperformed regular NMS in their experiments even when using this metric. We experimented with this method, but the resulting average precision was very similar to regular NMS. This is probably due to limited annotation of overlapping objects in the datasets.

7.2 Built-in invariance

A fundamental problem of computer vision is learning to separate the properties of the image from the inherent properties of the object itself. In practical terms, this means that the object detector has to be invariant to shifts in viewpoint, lighting, noise and other conditions caused by the properties and placement of the camera and by the environment of the object, rather than the object itself. We have shown that using deep learning techniques we can acquire most of this invariance automatically from the training data, given that the training data is sufficiently diverse. This way, it is up to the data to show which invariance types are useful to learn. The developer of the system does not have to take each one into consideration separately.

However, while learning the invariance is a useful property, it has to be balanced with practical considerations, such as efficiency. Convolutional networks differ somewhat from “blank slate” deep learning in that they force translation invariance (shown in figure 7.2) to hold as a structural property of the network. Pooling and stride operations cause invariance to small changes in pixel-level information while keeping the network down to a manageable size. In the experiments of this thesis, invariance assumptions were also made by the region proposal methods. Selective Search and Edge Boxes evaluate the objectness based on hand-crafted techniques. Faster R-CNN and other integrated methods sidestep this by learning to generate the regions.

The topic of scale invariance (shown in figure 7.3) is of interest to researchers. Naturally, it is possible to learn the concept of scale from the training data, but this is not necessarily the most efficient solution. If we completely disregard the context of the bounding box (i.e. the surrounding full image), we cannot usually deduce the size of the object from the contents of the box. Thus, the scale of the object within the image cannot change its class. Under this supposition, scale invariance is a required property of the object detector.

Traditional solutions for built-in scale invariance are image pyramids and filter pyramids [40]. In the former solution, images and feature maps are created at multiple scales, and the classification is performed for each scale. In the latter solution, multiple filter sizes are used for a fixed-size image and feature map. Both solutions require additional computation time for each

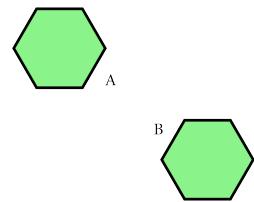


Figure 7.2: In a translation invariant system, objects A and B are identical.

scale level.

By default, Fast R-CNN does not implement built-in scale invariance, and this holds for the tested system as well. The original authors experimented with image pyramids (of five scales) and found that they provided a very slight improvement in precision compared to the baseline of one scale [20]. However, the precision-time trade-off was not advantageous.

More efficient methods already exist [35]. As mentioned above, Faster R-CNN uses special anchor boxes instead of image or filter pyramids [40]. The scale invariance method used by SSD belongs to a method family that is currently a topic of active research. These methods extract a feature pyramid directly from the feature maps of the convolutional network [36]. This method is faster than either of the traditional methods, because it effectively utilizes the computation of the convolutional network to create the feature pyramid from a single image scale.

Problem with the SSD method is that the low level activation maps of the convolutional network contain less semantical information than the high level activation maps. Thus, high resolution scales of the feature pyramid contain less relevant information than low resolution scales of the pyramid. A system called Feature Pyramid Network [35] proposes solving this problem by performing a top-down calculation step after the feature maps have been calculated in a bottom-up manner. The semantically more interesting high level low resolution activation maps are upscaled and summed with the low level high resolution activation maps to create new low level maps. According to the authors of the system, this method provides improved precision with marginal extra computation time. The method is independent of the architecture of the convolutional network.

7.3 Network depth

Since deep networks have been responsible for much of the improvement in object detection during the present decade, it is relevant to ask whether the networks could be made deeper still. For VGG-style networks, it has been experimentally found that, for a given problem, there is a certain optimal number of layers, after which training and test error start to increase. This is called the *degradation problem* [23]. The reasons for this behaviour are

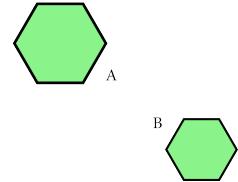


Figure 7.3: In a translation and scale invariant system, objects A and B are identical.

currently not wholly known. In the past, the back-propagation algorithm suffered from vanishing gradients, but this problem was mitigated by replacing sigmoidal activation functions with rectified linear units [22, p. 226]. It has been speculated that very deep networks simply do not converge in a reasonable amount of time [23].

One of the methods developed to enable training of deeper networks is a regularization technique called batch normalization [30]. The aim of the method is to keep the distribution of the layer inputs stable during training. In general, it is beneficial to preprocess the neural network input to have zero mean and unit variance, and this is almost always done. In batch normalization, instead of performing normalization once in the beginning, it is also performed during training and between layers. Batch normalization enables using a higher learning rate.

Another method for creating very deep networks is to use a residual network, also called a ResNet [23]. In a residual network, some of the layers do not operate directly on the input from the previous layer. Rather, the input is passed on unchanged for several layers, and the layers operate on the residual of the passed input and the output of the previous residual layer. One of the strengths of residual networks is that they can be trained using existing training algorithms and tools. A 100-layer ResNet outperforms VGG-16 by a few percentage points.

7.4 Region generation

In the method review chapter, we already demonstrated how the computational bottleneck created by the separate RoI generation phase can be removed by generating the regions in the convolutional network. This result has led to Fast R-CNN being phased out in favour of Faster R-CNN and other advanced methods.

Another interesting research topic related to the RoIs is how the amount of region proposals affects classification precision. In our results, Edge Boxes generated more boxes than Selective Search and performed better. However, the original authors of Fast R-CNN [20] have shown that generating more boxes only works up to a point. In the case of Fast R-CNN, replacing the smart region generator with a sliding window detector (i.e. a dense set solution) is not only slower, but also less precise.

Sparse set region generation methods continue to be popular in general. However, the success of methods such as SSD demonstrates that dense set solutions have not disappeared. Since slight changes in the network architecture can affect which method works best, it is difficult to predict which one

will be more popular in the future.

7.5 Region pooling and processing

RoI generation is not the only potential computational bottleneck. Even in systems such as Faster R-CNN, where regions are generated almost for free and the entire image is processed at once for all regions in the convolutional layers, there is a need for separate processing of the regions after the RoI pooling layer. In Fast R-CNN, this processing takes place in the fully connected layers. Since even a sparse set of region proposals typically contains thousands of bounding boxes, the processing can take up a significant portion of the forward-computation time of the neural network [20].

From a theoretical point of view, the topic of separately processing the regions is connected to the topic of translation invariance. The convolutional network up to the RoI pooling layer is translation invariant by design, and is thus well suited for locating object features from any part of the image. However, once the RoIs are extracted from the final convolutional activation map, translation begins to matter, because the objective is to produce boxes that correctly surround the objects. If the system is operating correctly, moving the box or the object within the box from its optimal location should decrease confidence of the detection [34].

Methods have been devised for making the detection network completely convolutional and enabling processing of all RoIs at once. In R-FCN [34], the translation-sensitivity of the RoIs is addressed by using position-sensitive score maps. The RoI is divided into rectangular sections or bins. Each bin has a corresponding score map. The pooled scores of the bins determine, by voting, whether the region is placed correctly over the object. According to the authors, R-FCN has similar precision to Faster R-CNN, but is approximately twice as fast (for both training and testing).

7.6 Object detection in context

Since many interesting lines of inquiry exist for improving convolutional object detection systems, is it worthwhile to study the lessons learned from testing the geometric inference method of the "Putting Objects in Perspective" publication? The most immediate lesson is that the method in its current form does not improve the performance of a convolutional object detector, except in certain marginal cases. These cases are difficult to separate from the numerous cases where the method degrades performance. From a

practical point of view, the method is also inefficient, because it requires a long computation time, which would have made it impractical even if it had performed as expected.

On one hand, the negative results from the geometric inference can be perceived as a reassertment of the performance capabilities of state-of-the-art systems. Fast R-CNN already works well enough to render irrelevant the effects of a system designed for the previous generation object detectors, and as we have demonstrated, many methods exist for improving the detection speed and accuracy of Fast R-CNN.

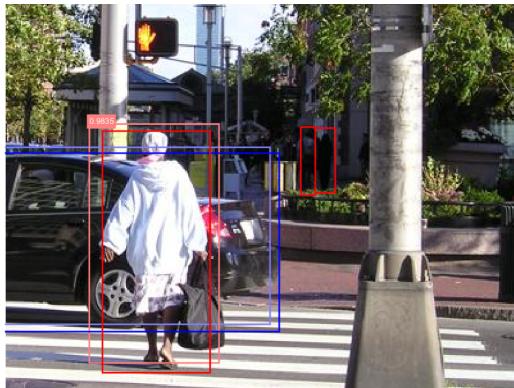


Figure 7.4: False negative cases in context (specifically, the two small red boxes in the background). True boxes are shown in darker colour than detections.

On the other hand, the starting point of the original authors of "Putting Objects in Perspective" would still appear to be valid. The improved convolutional methods still consider the object proposals (mostly) out of context. However, we know from practical examples that sometimes objects are only detectable from their context. Looking back at the false negative cases in figure 6.4, we can see that the first two human forms are almost impossible to visually detect as humans from the cropped images. However, from the complete image in figure 7.4, we can, with some difficulty, identify the figures as humans from their general shape, their location in the street and their slightly different colour compared to the surrounding environment.

Is there a possible way of integrating context-sensitivity into Fast R-CNN (or a similar system)? In general, this would appear to go against the grain of current object detectors, which have been developed to be fast and context-insensitive. We have already demonstrated how translation invariance is an integral part of convolutional networks and that extra steps

(such as fully-connected layers or position-sensitive score maps) are required to reimplement translation sensitivity for RoI processing. In Fast R-CNN, the fully-connected layers are computed once for each RoI. Adding context-sensitive processing to these layers should be possible, but would probably be prohibitively slow.

However, systems such as R-FCN and Feature Pyramid Network demonstrate how position-sensitivity can be implemented in a fully-convolutional network and how higher-level semantical information can be brought down to the lower layers of the network. Developing from these same ideas, could a fully-convolutional network be devised that, in addition to recognizing objects, learns to recognize the interrelationships between the objects and the scene in general? With increasing network depth, further layers could be added above the context-independent object detection layers. This would probably require hierarchical annotations.

Object detection time has been reduced drastically in the past few years with improvements in region-based convolutional methods. Eventually, the methods will be fast enough to free up computation time for additional methods of improving accuracy.

Chapter 8

Conclusions

In this chapter, we will review our results and provide some concluding remarks. We set out to review the current methods of convolutional object detection, to implement one such method and to explore potential improvements.

8.1 Theory

We began the thesis with a review of the theoretical background. We explained how neural networks function and what object detection entails. We demonstrated why regular neural networks are insufficient to image-related tasks and how translation invariant convolutional networks provide an effective solution to many computer vision problems.

Next, we demonstrated how convolutional object detection has evolved from the relatively slow R-CNN to the current optimized methods. This development is mostly not related to the structure of the convolutional network itself. Rather, it is related to how the convolutional network is used and to computation that takes place before and after the convolutional network. In the previous methods, there were many more separate phases involving preprocessing, region generation, computation of the fully connected layers and the final classification. In the latest methods, these phases have been increasingly integrated into the convolutional network itself, while keeping the basic CNN model intact. On the other hand, the 2016 winner of the ImageNet challenge [2] is again a model composed of many separate components. Nonetheless, several computational bottlenecks have disappeared. Over the past few years, the speed of object detection has improved more dramatically than precision.

8.2 Practice

To experiment with a convolutional method in practice, we created a working MATLAB implementation of Fast R-CNN. We learned that the most challenging part of implementing a deep learning system is collecting the training data and performing the training itself. The publicly available benchmark datasets serve as a useful starting point for both research and practical implementations. Training time can be further shortened by using a pretrained network. Even if the final system does not feature the same objects classes as the benchmark data, visual problems are universal enough to benefit from detectors trained for a different problem. The optimal bottom-layers of a convolutional network are often similar regardless of the problem, just like human eye uses the same receptive fields for all visual tasks. Thus, it makes sense to initialize the layers using a pretrained network.

Related to the implementation, we also learned that there are no easy “out-of-the-box” solutions for effectively implementing convolutional networks. Current software tools, such as Caffe and MatConvNet, require specialist skills. If it is possible to use such tools, creating a working implementation is not too difficult. However, the tools are quite finicky regarding interoperability of software versions and hardware, especially if implementation is attempted on a GPU.

8.3 Results

Regarding precision, the results were promising. We showed how a system trained on general image data can be used to detect objects in a specific task (traffic detection), thus demonstrating the adaptability of the methods. In many cases, Fast R-CNN detected more objects than the annotators of the original data had marked. These were labelled as false positives, despite clearly having the right object class by visual inspection.

On the other hand, the system did also make some actual mistakes. Water posts, trees and traffic cones were mistaken for humans. We deduced that this was most likely due to a lack of negative training examples. Further, we demonstrated the importance of postprocessing the results. Non-maximum suppression has a significant impact on the average precision. In general, overlapping objects and bounding boxes cause problems for object detection systems. We also showed that choice of the region generation method affects both speed and accuracy of the object detection system.

8.4 The future

Exact study of the speed of the method variants was left outside the scope of the thesis. More detailed execution-time evaluation on consumer-level computers would be an interesting topic for further research. Many methods that claim “real time performance” can achieve this only on hardware costing thousands of euros. Even though hardware cost is not a major issue for certain applications such as self-driving cars and computer servers, which use expensive hardware in any case, more applications become practical after CNNs can be evaluated on home computers and mobile devices. Yet, our implementation showed that the methods have improved enough to detect objects in a few seconds on a consumer laptop, even though we did not measure this exactly.

One of the strengths of convolutional networks is their inherent translation invariance. Yet, taking the context of the whole image into consideration could potentially create an even more precise system. We experimented with a geometry-based inference system, which alters the probabilities of object detections based on their geometric plausibility. However, the method did not improve the accuracy of Fast R-CNN. We provided an analysis of how the geometric detection method functioned and determined that, while the method eliminated some false detections, it also created many new ones. The geometric method was also impractically slow. Yet, lessons learned from the study can be used to explore further ways of implementing context-sensitive object detection.

As explained in the previous chapter, a trend can be perceived in the literature of making detection systems wholly neural or convolutional. Just like a deep neural network learns to automatically learn the inherent features of an object class, an even deeper and more smartly used neural network could learn the probabilities of finding an object from a certain part of the scene. This would make a separate geometric inference method irrelevant. However, time will show if this is the route future research takes. As a Danish proverb says, it is difficult to make predictions, especially about the future.

Bibliography

- [1] Imagenet database statistics. <http://image-net.org/about-stats>. Accessed: 2017-04-07.
- [2] Imagenet large scale visual recognition challenge 2016. <http://image-net.org/challenges/LSVRC/2016>. Accessed: 2017-03-18.
- [3] Infotrends - how long does it take to shoot 1 trillion photos? <http://blog.infotrends.com/?p=21573>. Accessed: 2017-06-20.
- [4] Kpcb internet trends report 2014. <http://www.kpcb.com/blog/2014-internet-trends>. Accessed: 2017-06-20.
- [5] Matconvnet: Cnns for matlab. <http://www.vlfeat.org/matconvnet/>. Accessed: 2017-06-21.
- [6] Nvidia cuda gpu listing. <https://developer.nvidia.com/cuda-gpus>. Accessed: 2017-07-28.
- [7] Piotr's computer vision matlab toolbox. <https://pdollar.github.io/toolbox/>. Accessed: 2017-04-10.
- [8] Software at the personal website of derek hoiem. <http://dhoiem.cs.illinois.edu/software/>. Accessed: 2017-04-21.
- [9] BENGIO, Y., ET AL. Deep learning of representations for unsupervised and transfer learning. *ICML Unsupervised and Transfer Learning 27* (2012), 17–36.
- [10] BILESCHEI, S. M. *StreetScenes: Towards scene understanding in still images*. PhD thesis, Citeseer, 2006.
- [11] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

- [12] BODLA, N., SINGH, B., CHELLAPPA, R., AND DAVIS, L. S. Improving object detection with one line of code. *arXiv preprint arXiv:1704.04503* (2017).
- [13] CHATFIELD, K., SIMONYAN, K., VEDALDI, A., AND ZISSERMAN, A. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531* (2014).
- [14] DALAL, N., AND TRIGGS, B. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* (2005), vol. 1, IEEE, pp. 886–893.
- [15] DOLLÁR, P., AND ZITNICK, C. L. Fast edge detection using structured forests. *IEEE transactions on pattern analysis and machine intelligence* 37, 8 (2015), 1558–1570.
- [16] ERHAN, D., SZEGEDY, C., TOSHEV, A., AND ANGUELOV, D. Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 2147–2154.
- [17] EVERINGHAM, M., VAN GOOL, L., WILLIAMS, C. K. I., WINN, J., AND ZISSERMAN, A. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision* 88, 2 (June 2010), 303–338.
- [18] FELZENZWALB, P. F., AND HUTTENLOCHER, D. P. Efficient graph-based image segmentation. *International journal of computer vision* 59, 2 (2004), 167–181.
- [19] FUKUSHIMA, K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks* 1, 2 (1988), 119–130.
- [20] GIRSHICK, R. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 1440–1448.
- [21] GIRSHICK, R., DONAHUE, J., DARRELL, T., AND MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2014), pp. 580–587.
- [22] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.

- [23] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.
- [24] HOIEM, D., EFROS, A. A., AND HEBERT, M. Automatic photo pop-up. *ACM transactions on graphics (TOG)* 24, 3 (2005), 577–584.
- [25] HOIEM, D., EFROS, A. A., AND HEBERT, M. Geometric context from a single image. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (2005), vol. 1, IEEE, pp. 654–661.
- [26] HOIEM, D., EFROS, A. A., AND HEBERT, M. Putting objects in perspective. *International Journal of Computer Vision* 80, 1 (2008), 3–15.
- [27] HORNIK, K. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4, 2 (1991), 251–257.
- [28] HUANG, T. Computer vision: Evolution and promise. *CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH-REPORTS-CERN* (1996), 21–26.
- [29] HUBEL, D. H., AND WIESEL, T. N. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology* 195, 1 (1968), 215–243.
- [30] IOFFE, S., AND SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR abs/1502.03167* (2015).
- [31] JIA, Y., SHELMAMER, E., DONAHUE, J., KARAYEV, S., LONG, J., GIRSHICK, R., GUADARRAMA, S., AND DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093* (2014).
- [32] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.
- [33] LECUN, Y., BOSER, B., DENKER, J. S., HENDERSON, D., HOWARD, R. E., HUBBARD, W., AND JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.

- [34] LI, Y., HE, K., SUN, J., ET AL. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems* (2016), pp. 379–387.
- [35] LIN, T.-Y., DOLLÁR, P., GIRSHICK, R., HE, K., HARIHARAN, B., AND BELONGIE, S. Feature pyramid networks for object detection. *arXiv preprint arXiv:1612.03144* (2016).
- [36] LIU, W., ANGUELOV, D., ERHAN, D., SZEGEDY, C., REED, S., FU, C.-Y., AND BERG, A. C. Ssd: Single shot multibox detector. In *European Conference on Computer Vision* (2016), Springer, pp. 21–37.
- [37] LONG, L. N., AND GUPTA, A. Scalable massively parallel artificial neural networks. *Journal of Aerospace Computing, Information, and Communication* 5, 1 (2008), 3–15.
- [38] LOWE, D. G. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on* (1999), vol. 2, Ieee, pp. 1150–1157.
- [39] MARR, D., AND HILDRETH, E. Theory of edge detection. *Proceedings of the Royal Society of London B: Biological Sciences* 207, 1167 (1980), 187–217.
- [40] REN, S., HE, K., GIRSHICK, R., AND SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (2015), pp. 91–99.
- [41] ROJAS, R. *Neural Networks - A Systematic Introduction*). Springer-Verlag, Berlin, New-York, 1996.
- [42] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J., ET AL. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 (1988), 1.
- [43] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., BERG, A. C., AND FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.
- [44] SEBE, N. *Machine learning in computer vision*, vol. 29. Springer Science & Business Media, 2005.

- [45] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
- [46] SPRINGENBERG, J. T., DOSOVITSKIY, A., BROX, T., AND RIEDMILLER, M. A. Striving for simplicity: The all convolutional net. *CoRR abs/1412.6806* (2014).
- [47] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15 (2014), 1929–1958.
- [48] STEINKRAUS, D., BUCK, I., AND SIMARD, P. Using gpus for machine learning algorithms. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on* (2005), IEEE, pp. 1115–1120.
- [49] SZELISKI, R. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.
- [50] TORRALBA, A., FREEMAN, W. T., AND MURPHY, K. P. Graphical model for recognizing scenes and objects. In *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. SchÃ¶lkopf, Eds. MIT Press, Cambridge, MA, 2003, p. None.
- [51] UIJLINGS, J. R. R., VAN DE SANDE, K. E. A., GEVERS, T., AND SMEULDERS, A. W. M. Selective search for object recognition. *International Journal of Computer Vision* 104, 2 (2013), 154–171.
- [52] VAN DE SANDE, K. E., UIJLINGS, J. R., GEVERS, T., AND SMEULDERS, A. W. Segmentation as selective search for object recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (2011), IEEE, pp. 1879–1886.
- [53] WALTHER, D., ITTI, L., RIESENHUBER, M., POGGIO, T., AND KOCH, C. Attentional selection for object recognition - a gentle way. In *International Workshop on Biologically Motivated Computer Vision* (2002), Springer, pp. 472–479.
- [54] WILSON, D. R., AND MARTINEZ, T. R. The general inefficiency of batch training for gradient descent learning. *Neural Networks* 16, 10 (2003), 1429–1451.

- [55] YANG, B., YAN, J., LEI, Z., AND LI, S. Z. Craft objects from images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 6043–6051.
- [56] ZENG, X., OUYANG, W., YAN, J., LI, H., XIAO, T., WANG, K., LIU, Y., ZHOU, Y., YANG, B., WANG, Z., ET AL. Crafting gbd-net for object detection. *arXiv preprint arXiv:1610.02579* (2016).
- [57] ZITNICK, C. L., AND DOLLÁR, P. Edge boxes: Locating object proposals from edges. In *European Conference on Computer Vision* (2014), Springer, pp. 391–405.