Karlsruhe Institute of Technology

Multi-task Network Cascades for Instance Segmentation using Bounding Boxes as Weak Supervision

Master thesis of

Andreas Schmidt

At the faculty of Computer Science
Institute for Anthropomatics and Robotics

Reviewer:	Prof. Dr.-Ing. Rainer Stiefelhagen
Second reviewer:	Prof. Dr.-Ing. Tamim Asfour
Advisor:	Monica Haurilet

Duration: 6. April 2017 – 29. September 2017

Computer Vision for Human-Computer Interaction Research Group
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology
Title: Multi-task Network Cascades for Instance Segmentation using Bounding Boxes as Weak Supervision
Author: Andreas Schmidt

Andreas Schmidt
Durlacher Allee 44
76131 Karlsruhe
ams.91@web.de

Statement of Authorship

I hereby declare that this thesis is my own original work which I created without illegitimate help by others, that I have not used any other sources or resources than the ones indicated and that due acknowledgement is given where reference is made to the work of others.

Karlsruhe, 29. September 2017

.....
(Andreas Schmidt)

Abstract

Instance segmentation, i.e. the precise localization of objects in an image, benefits from the tremendous progress in deep learning. Training deep neural networks for this task requires a large number of manually labeled pixel-wise segmentation masks. This annotation is time consuming and cost-intensive and, hence, weaker forms of annotation reduce the necessary annotation effort. Bounding boxes, for example, are ∼15 times faster to obtain than pixel-wise segmentation masks [LMB+14]. Therefore, our weakly-supervised segmentation method is built upon the fully-supervised Multi-task Network Cascade (MNC) model of Dai et al. [DHS16] enabling the use of bounding boxes as weak supervision. Thereby, we use foreground-background segmentation algorithms that generate segmentation masks within each bounding box. We use these masks to train our network instead of manually labeled pixel-wise masks.

This thesis focuses on instance segmentation using bounding boxes as weak supervision. First, we provide baselines that are an independent combination of the YOLO object detector and seven different foreground-background segmentation algorithms. These segmentation algorithms weakly generate segmentation masks within each bounding box. We use the baselines to investigate the performance capability of these foreground-background segmentation algorithms without training our neural network. Moreover, the baselines provide comparability to our neural network.

Our weakly-supervised segmentation method is built upon the fully-supervised MNC model of Dai et al. [DHS16]. We enable the use of weakly-supervised training data in form of bounding boxes and, thus, we do not need manually labeled pixel-wise segmentation masks. The MNC model consists of three tasks: object detection (bounding boxes), segmentation of objects within each bounding box and categorization (label assignment) of each segmentation mask. By using bounding boxes as weak supervision, we can train the first and third task of the MNC model unchanged. For the second task, we use foreground-background segmentation algorithms to segment objects within each bounding box and, thus, our neural network learns these weakly generated masks instead of manually labeled masks.

GrabCut and MCG achieve the best baseline results between all foreground-background segmentation algorithms. Therefore, we choose these two algorithms to train our weakly-supervised MNC model. Our best weakly-supervised MNC model improves the state-of-the-art in weakly-supervised instance segmentation with respect to the $mAP^r_{0.5}$ and ABO results. In addition, we achieve ∼10% less $mAP^r_{0.5}$ and ∼8% less mIoU compared to the fully supervised network, but save time and cost, since less annotation effort. Moreover, we achieve better results in instance segmentation than the corresponding baselines and decrease the inference time to ∼300 ms per image. In addition, the inference time per image is independent of the number of objects in contrast to the state-of-the-art of Khoreva et al. [KBH+17].

Zusammenfassung

Instanzsegmentierung, sprich die genaue Lokalisierung von Objekten in einem Bild, profitiert von dem enormen Fortschritt im Bereich Deep Learning. Um tiefe neuronale Netze zu trainieren, wird eine große Anzahl manuell gelabelter pixelgenauer Segmentierungsmasken benötigt. Allerdings ist die Annotation dieser Daten zeitaufwändig und kostenintensiv. Schwächere Annotationformen reduzieren daher den benötigten Annotationsaufwand. Die Annotation von Bounding Boxen zum Beispiel ist ∼15 mal schneller als die Annotation von pixelgenauen Segmentierungsmasken [LMB+14]. Deshalb baut unsere schwach-überwachte Segmentierungsmethode auf dem voll-überwachten Multi-task Network Cascade (MNC) Modell von Dai et al. [DHS16] auf und erlaubt die Nutzung von Bounding Boxen als schwache Überwachung. Dabei benutzen wir Vordergrund-Hintergrund-Segmentierungsalgorithmen, die Segmentierungsmasken innerhalb jeder Bounding Box generieren. Wir nutzen diese Masken anstatt manuell gelabelter pixelgenauer Masken, um unser Netzwerk zu trainieren

In dieser Arbeit konzentrieren wir uns auf Instanzsegmentierung mittels Bounding Boxen als schwache Überwachung. Als erstes stellen wir Baselines bereit, die eine unabhängige Kombination aus dem YOLO Objektdetektor und sieben verschiedenen Vordergrund-Hintergrund Segmentierungsalgorithmen sind. Diese Segmentierungsalgorithmen generieren Segmentierungsmasken innerhalb jeder Bounding Box. Wir nutzen diese Baselines, um die Leistungsfähigkeit der Vordergrund-Hintergrund Segmentierungsalgorithmen zu untersuchen, ohne jedes Mal ein neuronales Netz zu trainieren. Darüber hinaus bieten diese Baselines Vergleichbarkeit gegenüber unserem neuronalen Netz.

Unsere schwach-überwachte Segmentierungsmethode baut auf dem voll-überwachten MNC Modell von Dai et al. [DHS16] auf. Wir ermöglichen die Nutzung von schwach-gelabelten Trainingsdaten in Form von Bounding Boxen und brauchen daher keine manuell gelabelten pixelgenauen Segmentierungsmasken. Das MNC-Modell besteht aus drei verschiedenen Aufgaben: Objektdetektion (Bounding Boxen), Segmentierung der gefundenen Objekte innerhalb jeder Bounding Box und Kategorisierung (Labelzuweisung) jeder Segmentierungsmaske. Indem wir Bounding Boxen als schwache Überwachung benutzen, können wir die erste und dritte Aufgabe des MNC Modells unverändert trainieren. Für die zweite Aufgabe nutzen wir Vordergrund-Hintergrund Segmentierungsalgorithmen, um die Objekte innerhalb der Bounding Boxen zu segmentieren. Damit kann unsere neuronales Netz die schwach generierten Masken anstatt der manuell gelabelten Masken lernen.

GrabCut und MCG erreichen die besten Baseline-Ergebnisse unter allen Vordergrund-Hintergrund Segmentierungsalgorithmen. Deshalb wählen wir diese beiden Algorithmen, um unser schwach-überwachtes MNC Modell zu trainieren. Unser bestes schwach-überwachtes MNC Modell übertrifft den State-of-the-art in schwach-überwachter Instanzsegmentierung hinsichtlich der $mAP^r_{0.5}$ und ABO Ergebnisse. Zusätzlich erreichen wir ∼10% weniger $mAP^r_{0.5}$ und ∼8% weniger mIoU verglichen mit dem voll-überwachten neuronalen Netz, sparen uns allerdings Zeit und Kosten aufgrund des geringeren Annotationsaufwandes. Darüber hinaus erreichen wir bessere Ergebnisse in Instanzsegmentierung als die entsprechenden Baselines und verringern zusätzlich die Inferenzzeit auf ∼300 ms pro Bild. Außerdem ist die Inferenzzeit pro Bild unabhängig von der Anzahl von Objekten im Gegensatz zum State-of-the-art von Khoreva et al. [KBH+17].

Contents

Acronyms

ABO Average Best Overlap

ANN Artifical Neural Network

AP Average Precision

CE Cross Entropy

CNN Convolutional Neural Network

CRF Conditional Random Field

DenseCRF Dense Conditional Random Field

EM Expectation Maximization

FCN Fully Convolutional Network

FN False Negative

FP False Positive

GMM Gaussian Mixture Model

GT Ground Truth

HOG Histogram of Orientated Gradients

IBC Image Boundary Contrast

ILSVRC ImageNet Large Scale Visual Recognition Challenge

IoU Intersection over Union

mAP mean Average Precision

MBD Minimum Barrier Distance

MBOD Minimum Barrier Salient Object Detection

MCG Multiscale Combinatorial Grouping

MIL Multiple Instance Learning

mIoU mean Intersection over Union

MLP Multi-Layer Perceptron

MNC Multi-task Network Cascade

MS COCO Microsoft Common Objects in Context

MSE Mean Squared Error

PascalVOC PASCAL Visual Object Classes

RBD Robust Background Detection

ReLU Rectified Linear Unit

ResNet Residual Network

RoI Region of Interest

RPN Region Proposal Network

SBD Semantic Boundaries Dataset

SGD Stochastic Gradient Descent

SIFT Scale Invariant Feature Transform

SLIC Simple Linear Iterative Clustering

SLP Single-Layer Perceptron

SPL Self-Paced Learning

SPN Superpixel Pooling Network

SVM Support Vector Machine

tanh Tangens Hyperbolicus

TN True Negative

TP True Positive

UCM Ultrametric Contour Map

YOLO You Only Look Once

1. Introduction

1.1 Motivation

Computer vision made tremendous progress since the revival of neural networks. A wide range of computer vision tasks, e.g. object detection and image segmentation, benefit from this progress. Object detection models [RHGS15, RDGF16, LAE+16] provide object locations in form of bounding boxes. Image segmentation is the task of assigning to each pixel a label and, thus, segmentation models [NHH15, LSD15, HAGM15, DHS16] provide pixel-wise segmentation masks. Current Convolutional Neural Network (CNN) based models achieve high accuracy in both computer vision tasks. The segmentation task is further distinguished between semantic and instance segmentation depending on the label type. In semantic segmentation the labels are solely class-aware (e.g. person, dog). In addition, instance segmentation can also differentiate between two instances of the same class by using instance-aware labels (e.g. person_1, person_2). Thus, instance-aware segmentation masks provide more precise object locations than current object detectors. Precise object location is necessary for a variety of application like the safety enhancement in traffic by using driver assistance systems or autonomous driving cars. According to the report on road safety published by the World Health Organization in 2015 [Wor15], more than 3,000 people in Germany and about 1.25 million people in the world are dying due to traffic accidents. Driver assistance systems and autonomous driving cars have the potential to reduce the number of road traffic fatalities. Thereby, cars can benefit from precise object locations to prevent collisions with other road users or pedestrians by performing braking or evasive maneuvers. Precise object locations can not only increase the safety of car drivers and car passengers but also the safety of visually impaired people. A technical system can support them to navigate in known and in particular in unknown environments. Object locations can be used for preventing collisions and, thus, finding optimal routes. Furthermore, Service robots that operate in environments built for humans are another application. Perceiving objects in these environments is one essential skill of service robots, since they have to navigate through these environments or use objects for their tasks. For example, a household service robot has to navigate in the kitchen and grasp a knife to cook. Thus, precise object locations can help service robots to navigate and operate in these environments.

To achieve good results in instance segmentation, neural networks require a lot of annotated training data. According to Mercer's famous comment "There is no data like more data" [Jel05], the availability of large scale datasets increased in the last years and, thus, neural networks achieved state-of-the-art results in a lot of computer vision tasks. One example of large scale datasets is the ImageNet dataset [DDS+09] for image classification, consisting of more than one million labeled training images. Datasets for image segmentation are also available, since segmentation models also can benefit from large scale datasets. However, common datasets that provide pixel-wise segmentation masks are smaller in terms of number of images compared to the ImageNet dataset. The MS COCO dataset [LMB+14], the largest public available dataset for semantic segmentation, provides more than 150,000 training images, nearly eight times smaller than the ImageNet dataset. Another common dataset in segmentation, the PascalVOC dataset [EEVG+15] contains almost 1,500 training images, what is nearly 875 times smaller than the ImageNet dataset. The reason for this unevenness is that annotating/labeling images takes a different amount of time depending on the needed annotation. According to the annotation report of the large-scale Microsoft Common Objects in Context (MS COCO) dataset [LMB+14], obtaining object bounding box annotations, for example, is ∼15 times faster than obtaining pixel-wise segmentation masks. However, collecting large annotated datasets is not only time-consuming but also requires substantial financial investments. One possibility to keep the annotation time and in particular the financial investments within limits is to use easily obtainable annotations for more than one computer vision task. For example, use bounding box annotations for object detection and image segmentation. Several models [KBH+17, PCMY15, DHS15a, RLO+17, PC15, PCMY15, WLC+16, PKD15, KL16] based on CNNs use weaker forms of annotations (weak supervision) instead of using fully-supervised pixel-wise segmentation masks (full supervision). There are different types of weak supervision, e.g. image tags, bounding boxes, labeled points and scribbles. All types differ in annotation time and information content. Image tags for example include no knowledge about the object position in contrast to bounding boxes. However, obtaining bounding boxes is more time-consuming. Therefore, the choice of weak supervision is a trade-off between information content and annotation time and, thus, also the amount of annotated data.

1.2 Objectives

Our goal is to segment instances solely using bounding boxes as supervision during training. Thereby, we want to achieve competitive results on the PASCAL Visual Object Classes (PascalVOC) dataset compared to full supervision in consideration of the gain in annotation time and the corresponding financial investments. For this purpose, we combine an object detector with different foreground-background segmentation algorithms that do not need pixel-wise segmentation masks during training. These methods serve as baselines for our end-to-end trained weakly-supervised instance segmentation model that is based on the Multi-task Network Cascade (MNC) [DHS16]. For weakly-supervised instance segmentation, we generate object segmentation masks within each bounding box and, thus, our MNC model learns these generated masks instead of annotated pixel-wise segmentation masks. Since precise object detection is needed in time-critical applications, we also focus on fast and input-independent inference, i.e. the time that is needed to segment all instances in an image.

1.3 Outline

This thesis is structured as follows: In chapter 2, we introduce the fundamentals that are necessary for this thesis. Thereby, we cover the Expectation Maximization (EM) scheme, the basics in deep learning, the image segmentation task, common datasets and evaluation metrics for image segmentation An overview of related work in image segmentation is presented in chapter 3. We consider related work in both semantic segmentation and instance segmentation with different supervision types. In chapter 4, we present our baselines and our end-to-end trained model. In chapter 5, we evaluate the baselines and our end-to-end trained model in instance segmentation and as well in semantic segmentation. Furthermore, we compare our results with respect to segmentation results and inference time. In chapter 6, we summarize the results of our work and give an outlook on future work.

2. Fundamentals

This chapter covers fundamental topics that are necessary for the following chapters. First, we introduce to the Expectation Maximization (EM) scheme and K-Means as a specific EM algorithm (see section 2.1). Second, we focus on the basics of deep learning in section 2.2, including CNNs, a neural network architecture applied in computer vision. In section 2.3, we consider neural networks for image classification. Afterwards, we present the image segmentation task in section 2.4, subdivided into semantic and instance segmentation. In section 2.5, we give an overview of common datasets used in computer vision for different tasks to train machine learning approaches. Finally, we focus on evaluation metrics for image segmentation (see section 2.6).

We use a standardized nomenclature in this thesis. Scalars are depicted by lower case letters (e.g. θ). Bold lower case letters correspond to vectors (e.g. $\mathbf{x}$) and bold capital letters to matrices (e.g. $\mathbf{W}$). A set or a list are depicted by calligraphic capital letters (e.g. $\mathcal{X}$). In addition, we use capital letters (e.g. N) to define the number within a set or list.

2.1 Expectation Maximization

Expectation Maximization (EM) [DLR77] is a standard algorithm in statistics. By alternating between the expectation step and the maximization step the EM scheme attempts to improve a mathematical model (see figure 2.1):

- **Expectation:** Estimates the statistics of the complete data and, thus, assign the data samples to different parts of the model [DLR77].
- **Maximization:** Reestimates the parameter of the model using the assignment of the expectation step [DLR77].

The EM scheme can be used as the basis for many unsupervised clustering algorithms [Mit97, 191]. Clustering algorithms have been extensively studied since 1963 [Jai10] and are methods for grouping a set of patterns based on a distance function. We present K-Means as one common clustering algorithm based on the EM scheme.

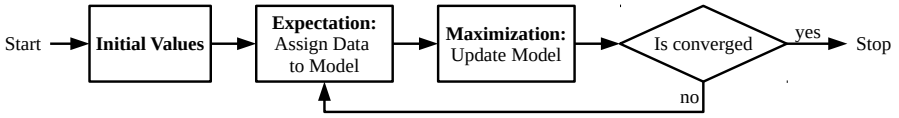


Figure 2.1: **EM Scheme:** Procedure of the Expectation Maximization (EM) scheme divided in the expectation step and the maximization step.

K-Means

We define the basics of K-Means similar to Jain [Jai10]. Let $\mathcal{X} = \{\mathbf{x}_0, ..., \mathbf{x}_{N-1}\}$ be a set of N p-dimensional patterns. The aim of K-Means is to divide the set $\mathcal{X}$ into K clusters $\mathcal{C} = \{\mathcal{C}_0, ..., \mathcal{C}_{K-1}\}$, where each cluster groups similar patterns. Each cluster is defined by its empirical mean μ_k and a subset $\mathcal{C}_k \subset \mathcal{X}$ of all patterns that are belonging to this cluster. Since, by definition, a point belongs to a single cluster, the subsets are pairwise disjoint: $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for $i \neq j$. Intuitively, the distance between patterns of the same cluster should be small after applying K-Means, while the distance between patterns of different clusters should be large. The accumulated distance within a cluster $\mathcal{C}_k$ can be mathematically expressed as an error [Jai10]:

$$E(\mathcal{C}_k) = \sum_{\mathbf{x} \in \mathcal{C}_k} ||\mathbf{x} - \mu_k||^2 \tag{2.1}$$

where $||\cdot||$ defines a distance metric, e.g. Euclidean distance. If all patterns of a cluster are similar, i.e. close together, the error should small. Otherwise, if all patterns of a cluster vary a lot, the error is large. Since each cluster should represent similar patterns, this error can be extended to [Jai10]:

$$E(\mathcal{C}) = \sum_{k=0}^{K-1} E(\mathcal{C}_k) = \sum_{k=0}^{K-1} \sum_{\mathbf{x} \in \mathcal{C}_k} ||\mathbf{x} - \mu_k||^2 \tag{2.2}$$

Thus, dividing a set of patterns in clusters corresponds to minimizing this error function $E(\mathcal{C})$. Although minimizing this objective function is NP-hard [DFK$^+$04], a suitable solution can be found iteratively using K-Means (see algorithm 1).

K-Means is an iterative algorithm that starts with initial clusters $\mathcal{C}$ defined by K initial means. Each pattern is assigned to the cluster with the smallest distance between the pattern and the corresponding initial cluster mean. Thus, this algorithm includes three user-specified parameters:

- number of clusters K
- initial means μ_k
- distance metric $||\cdot||$

The first parameter, the number of clusters K, is usually not obvious, although it has a huge impact to the result. The problem with the error formulation in equation 2.2 is that increasing K decreases the error $E(\mathcal{C})$. For example, if there are as many clusters as patterns ($K = N$), the error becomes zero ($E(\mathcal{C}) = 0$). Thus, the algorithm can not learn the optimal K and K has to be fix. Heuristics for K can circumvent this problem.

Algorithm 1 K-Means

Require: number of clusters K
Require: initial means μ_k
Require: distance metric $||\cdot||$
  while cluster assignment is not stabilized **do**
    # **Expectation:** Calculate new clusters
    for all $\mathbf{x} \in \mathcal{X}$ **do**
      $j = \underset{k \in \{0,...,K-1\}}{\arg\min} ||\mathbf{x} - \mu_k||^2$ # find closest mean
      assign $\mathbf{x}$ to $\mathcal{C}_j$
    end for
    # **Maximization:** Compute new cluster centers
    for $k = 0$ **to** $K - 1$ **do**
      $\mu_k = \frac{1}{|\mathcal{C}_k|} \sum_{\mathbf{x} \in \mathcal{C}_k} \mathbf{x}$ # empirical mean
    end for
  end while
  return $\mathcal{C} = \{\mathcal{C}_0, ..., \mathcal{C}_{K-1}\}$

Typically, K-Means is run several times for different values of K independently [Jai10] and afterwards the most promising clustering is used. The choice of different initial means, can lead to different local minima and, thus, to different clusters. The Euclidean distance is typically used for K-Means to find spherical clusters.

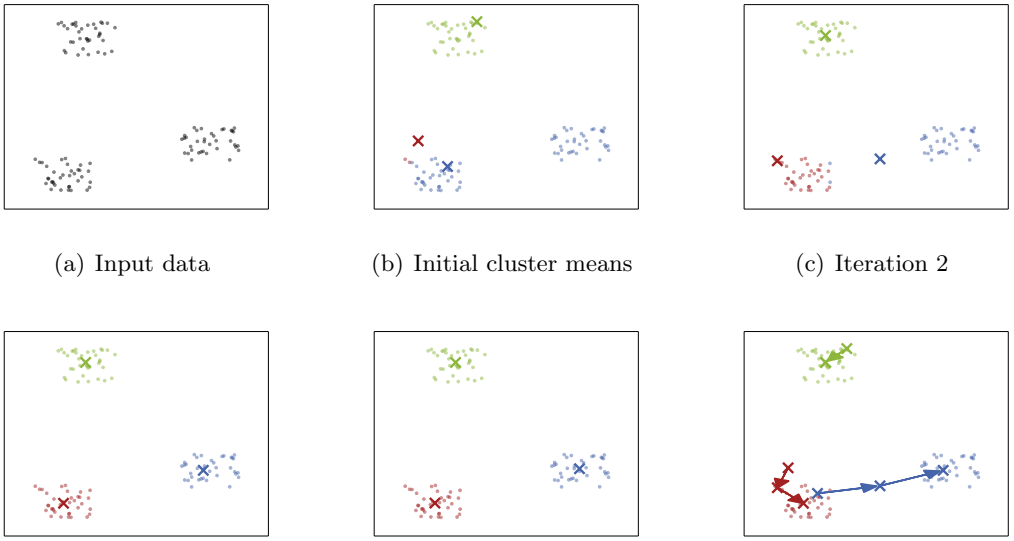
First, the three user-specified parameters have to be chosen and, afterwards, the error $E(\mathcal{C})$ can be minimized. According to the EM scheme, K-Means consists of these three steps [Jai10]:

1. **Initial Values:** Selecting an initial partition with K clusters (user-specified parameters).

2. **Expectation:** Generation of new partition by assigning each pattern to the closest cluster center.

3. **Maximization:** Computation of new cluster centers by using the new partition.

K-Means iteratively improves the clusters by applying step two and three alternately (EM scheme). This alternating process is illustrated in figure 2.2 for a two-dimensional dataset with three clusters. While the second step constantly reestimates the empirical means, the first step reassigns each pattern to the nearest cluster center. Thus, the clusters with its empirical means and their corresponding data points are updated in every iteration. Intuitively, this process moves ideally all empirical means μ_k in every iteration closer to the real cluster centers. For example, comparing the blue cluster in figure 2.2(c) and in figure 2.2(d), the empirical mean moves to the right cluster in one iteration. The iterative adaption of the empirical means and, thus, the adaption of the clusters is visualized in figure 2.2(f) by arrows.



(a) Input data (b) Initial cluster means (c) Iteration 2

(d) Iteration 3 (e) Final clustering (f) Final clustering and the changes of the cluster centers

Figure 2.2: **K-Means Example:** Clustering of a two-dimensional dataset and three clusters. This example shows the iterative changes of the cluster centers, converging to the real cluster centers.

2.2 Deep Learning

Humans can recognize their environment very fast and accurately due to the strong ability of our brains to recognize patterns. Therefore, the human brain and its neural activity inspired the structure and functionality of Artifical Neural Networks (ANNs). The fundamental cell of the brain is the neuron [DSSF+16, 9]. The human brain consists of nearly 10^{11} neurons, with each neuron connecting up to 100,000 other neurons [Du14, 1]. Neurons gained importance in computer science for the first time in 1943, when Warren McCulloch and Walter Pitts [MP43] described a first simplified mathematical model of a neuron. Thereby, they were inspired by the analysis of how biological neurons generate and propagate electrical impulses [DSSF+16, 11]. Finally, they modeled the artificial counterpart of the biological neuron as a simple threshold device to perform logic function. The next major step was in 1958, when Frank Rosenblatt developed the Perceptron. This neuron model can be used to classify linearly separable patterns. In section 2.2.1, we describe not only the structure of One-Neuron Perceptron but also how it can be trained. The next milestone in ANN's history is the backpropagation learning algorithm for Multi-Layer Perceptrons (MLPs) proposed in 1986 [Du14, 2]. In addition to the Perceptron, MLPs can also classify nonlinearly separable patterns. In section 2.2.2, we cover the structure of MLPs and the basics of the backpropagation learning algorithm. We obtain the basics of One-Neuron Perceptrons and MLPs primarily from the book "Neural Networks and Statistical Learning", proposed by Du and Swamy in 2014 [Du14]. Finally, section 2.2.3 addresses CNNs, a commonly applied neural network architecture in computer vision.

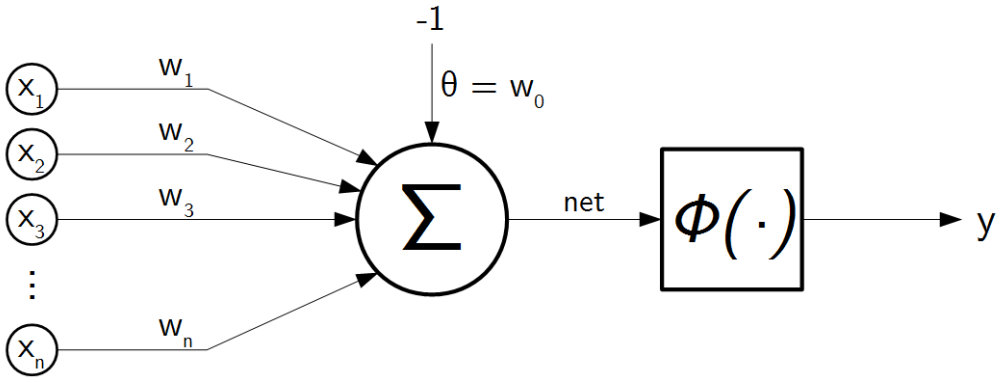


Figure 2.3: **Structure**: Structure of an One-Neuron Perceptron including the inputs x_i, the weights w_i, the bias θ and the activation function $\phi(\cdot)$.

2.2.1 One-Neuron Perceptron

In this section, we describe the structure and the training of an One-Neuron Perceptron that was developed by Frank Rosenblatt in 1958 [Ros58].

Structure

A biological neuron is connected up to 100,000 other neurons [Du14, 1]. Thus, its main function is the signal transmission of incoming stimuli. Thereby, the neuron transmit a stimulus if the incoming signals of all connected neurons exceed a certain threshold. Since Frank Rosenblatt was inspired by the biological neuron, the One-Neuron Perceptron models this behavior mathematically. The incoming signals are modeled by a vector $\mathbf{x}$. The input $\mathbf{x}$ is weighted with $\mathbf{w}$, to control the influence of specific inputs [Du14, 67]. The One-Neuron Perceptron fires, if the weighted sum is greater than a threshold θ. Thus, the artificial neurons input is defined by [Du14, 67]:

$$net = \sum_{i=1}^{n} w_i \cdot x_i - \theta = \mathbf{w}^\top \cdot \mathbf{x} - \theta \tag{2.3}$$

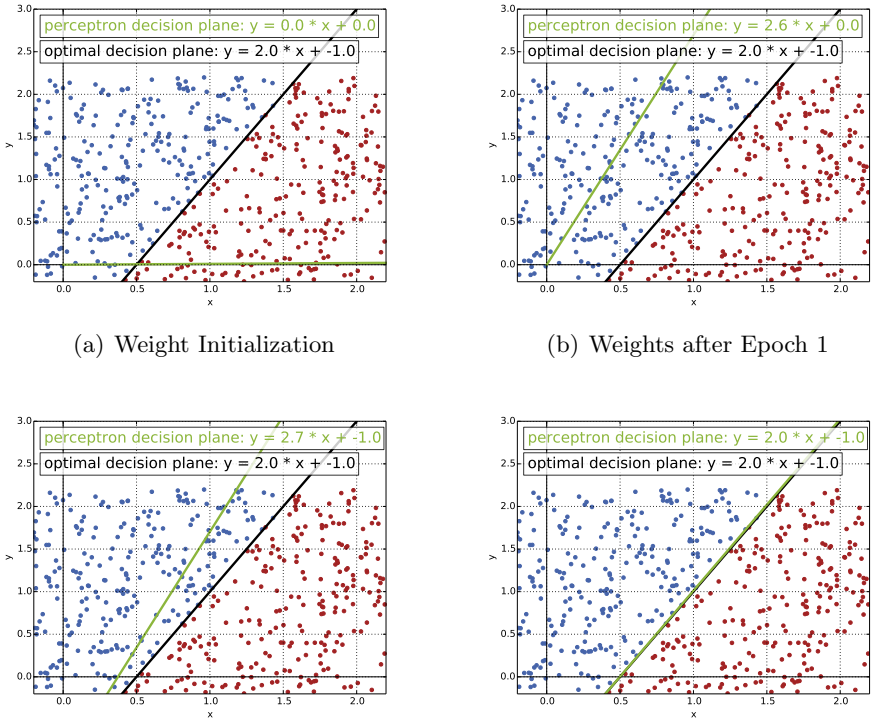
Furthermore, the basic structure of an One-Neuron Perceptron is illustrated in figure 2.3. As already stated, the One-Neuron Perceptron can be used to classify linearly separable patterns. Geometrically, the weights $\mathbf{w}$ define a hyperplane that separates the decision regions [Du14, 67]:

$$\mathbf{w}^\top \cdot \mathbf{x} - \theta = 0 \tag{2.4}$$

The threshold θ can shift the hyperplane away from the origin [Du14, 67]. For simplicity the threshold θ can be added to the weight vector $\mathbf{w}$ as w_0 and a constant input (-1). Hence, equation 2.3 can be written as a simple vector multiplication:

$$net = \sum_{i=0}^{n} w_i \cdot x_i = \mathbf{w}^\top \cdot \mathbf{x} \tag{2.5}$$

Since the One-Neuron Perceptron is motivated by the behavior of the biological neuron, the Perceptron fires if weighted input exceeds the threshold θ. According to equation 2.5



(a) Weight Initialization

(b) Weights after Epoch 1

(c) Weights after Epoch 6

(d) Weights after Epoch 13

Figure 2.4: **Weight Learning Example:** Learning the weights $\mathbf{w}$ of an One-Neuron Perceptron in a two-dimensional dataset. By using a learning rate η that decreases every epoch, the decision plane converges to the optimal decision plane.

the threshold is exceeded if $net > 0$. An activation function $\phi(\cdot)$ models the firing of an One-Neuron Perceptron mathematically. Thus, the final output y of an One-Neuron Perceptron is defined by:

$$y = \phi(net) \tag{2.6}$$

There are several different activation functions $\phi(\cdot)$, but the easiest one is the linear activation (identity function):

$$\phi(x) = x \tag{2.7}$$

In section 2.2.2, we discuss the problem of linear activation when using MLPs and present further activation functions. These activation functions can also be used for an One-Neuron Perceptron.

Training

To classify linearly separable patterns the One-Neuron Perceptron has to learn appropriate weights and, thus, learns an appropriate decision hyperplane. Unfortunately, finding optimal weights is nontrivial. Therefore, the One-Neuron Perceptron is trained by adapting the weights iteratively using training samples. Thus, the Perceptron decision hyperplane

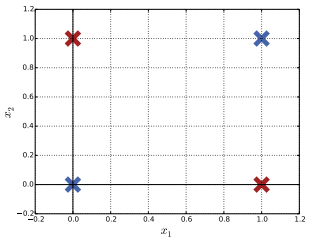


Figure 2.5: **XOR-Problem:** The two-dimensional XOR-Problem is not solvable by a linear decision plane.

plane should converge to the optimal decision hyperplane. The weights are initialized to small random values [GBC16, 176]. A training sample includes the corresponding pattern (e.g. $\mathbf{x} = (x_1, x_2)$) and its target value. Figure 2.4 illustrates two classes (represented by different colors) in a two-dimensional dataset. The green plane represent the decision hyperplane defined by the weights of the Perceptron. The black plane represent the optimal decision hyperplane.

We mention a simple method to train a Perceptron. During training of the Perceptron, the output $y(\mathbf{x})$ of each training pattern $\mathbf{x}$ is compared to its target value $t(\mathbf{x})$. The output is determined according equation 2.6. The weights $\mathbf{w}$ are updated only if the target value and the output (prediction) differ. If they differ, the decision hyperplane should be shifted towards the wrongly classified pattern. A learning rate η is usually used to influence the weight update. The update rule is defined by [Du14, 72]:

$$\begin{aligned} \mathbf{w} &= \mathbf{w} + \Delta\mathbf{w} \\ \Delta\mathbf{w} &= \eta \cdot (t(\mathbf{x}) - y(\mathbf{x})) \cdot \mathbf{x} \end{aligned} \tag{2.8}$$

where $(t(\mathbf{x}) - y(\mathbf{x}))$ defines the error between target and prediction. This rule updates the weights after each training sample. An epoch summarizes the update step for each training sample, i.e. each training sample is considered one time. We use this update rule for the example in figure 2.4 and decrease the learning rate after each epoch. The Perceptron converges iteratively to the optimal decision hyperplane.

2.2.2 Multi-Layer Perceptron

The One-Neuron Perceptron can be trained to classify linearly separable patterns, but it reaches its limit if the problem is nonlinear [Du14, 1]. The easiest nonlinear problem is the XOR-Problem [Du14, 7] ("exclusive or"), illustrated in figure 2.5. Although this is an easy two-dimensional problem, there exists no linear plane to distinguish between the two classes. In contrast to the One-Neuron Perceptron, Multi-Layer Perceptrons (MLPs) can also classify nonlinearly separable patterns.

Structure

While the One-Neuron Perceptron models one neuron, the MLP consists of multiple neurons. Each neuron is built like an One-Neuron Perceptron, including the calculation of the weighted sum of the inputs and an activation function. Neurons are organized in layers

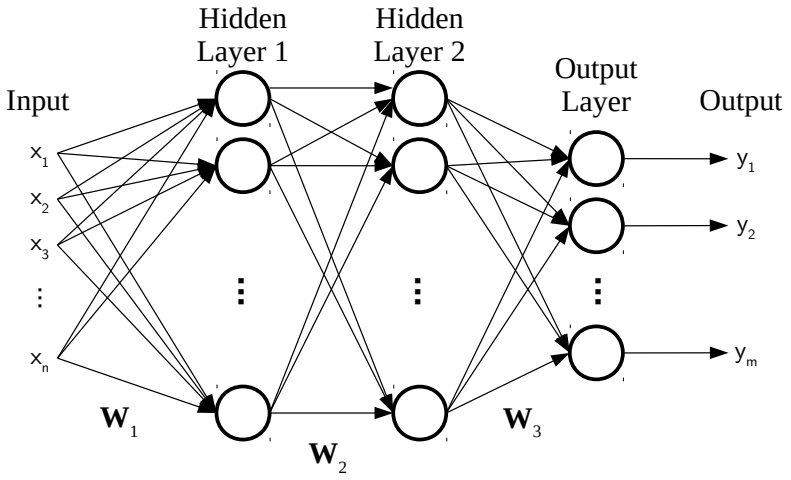


Figure 2.6: **Structure:** Structure of a MLP with input, hidden and output layers.

(see figure 2.6). Each neuron of a specific layer is connected to every neuron of the following layer, i.e there are no connections between neurons of nonadjacent layers. Therefore, this is called fully-connected. The MLP consists of an input layer, an output layer and at least one hidden layer.

If we consider a MLP with M neurons and L layers, each neuron consists of a weight vector $\mathbf{w}_i$ defining its connections. Then, w_{ji} defines the weight from neuron j to neuron i. The output o_i (for $i = 1, ..., M$) of a neuron is defined according to equation 2.5 and equation 2.6. All weight vectors of a layer k (for $k = 1, ..., L$) can be aggregated in one weight matrix $\mathbf{W}_k$ and, thus, defines the output $\mathbf{o}_k$ of a layer as [Du14, 83]:

$$\begin{aligned} \mathbf{net}_k &= \mathbf{W}_k^\top \cdot \mathbf{o_{k-1}} \\ \mathbf{o}_k &= \phi_\mathbf{k}\left(\mathbf{net}_k\right) \end{aligned} \tag{2.9}$$

where $\mathbf{o_0} = \mathbf{x}$ and $\phi_k(\cdot)$ applies an activation function on every value of $\mathbf{net}_k$. According to equation 2.9 the output of the whole MLP is equal to the output of the last layer: $\mathbf{y} = \mathbf{o}_L$ Furthermore, $\mathbf{W} = [w_{ji}]$ aggregates all weights in one weight matrix.

Activation Functions

An MLP with linear activation functions can be expressed as an One-Neuron Perceptron and, therefore, can not classify nonlinearly separable patterns. We illustrate this problem

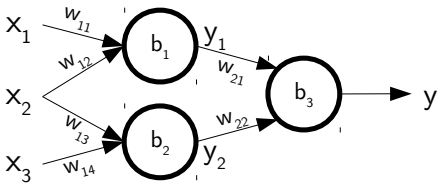
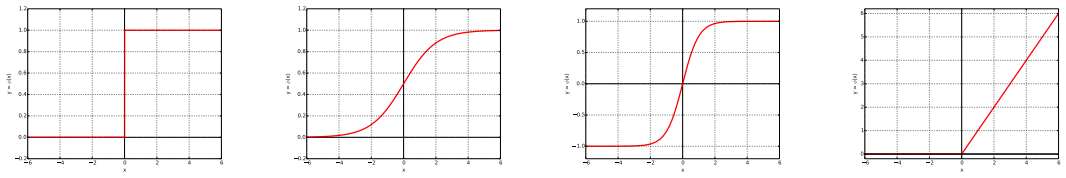


Figure 2.7: **Example:** Example of a MLP with linear activation.



(a) step (b) sigmoid (c) tanh (d) ReLU

Figure 2.8: **Activation Functions:** Comparison of four different activation functions.

by an easy example. Considering the neural network visualized in figure 2.7, the outputs of the hidden neurons are defined according equation 2.5:

$$y_1 = w_{11}x_1 + w_{12}x_2 + b_1$$
$$y_2 = w_{13}x_2 + w_{14}x_3 + b_2$$

Then, the output of the whole network is defined by:

$$y = w_{21}y_1 + w_{22}y_2 + b_3$$

This example shows that this specific MLP can be expressed as a weighted sum of the input variables and, therefore, can be expressed by an One-Neuron Perceptron. This is achieved by insertion of y_1 and y_2 into the formula of the network output:

$$\begin{aligned}
y &= w_{21}y_1 + w_{22}y_2 + b_3 \\
&= w_{21}(w_{11}x_1 + w_{12}x_2 + b_1) + w_{22}(w_{13}x_2 + w_{14}x_3 + b_2) + b_3 \\
&= w_{21}w_{11}x_1 + w_{21}w_{12}x_2 + w_{21}b_1 + w_{22}w_{13}x_2 + w_{22}w_{14}x_3 + w_{22}b_2 + b_3 \\
&= \underbrace{(w_{21}w_{11})}_{=\tilde{w}_1}x_1 + \underbrace{(w_{21}w_{12} + w_{22}w_{13})}_{=\tilde{w}_2}x_2 + \underbrace{(w_{22}w_{14})}_{=\tilde{w}_3}x_3 + \underbrace{(w_{21}b_1 + w_{22}b_2 + b_3)}_{=\tilde{b}} \\
&= \tilde{w}_1x_1 + \tilde{w}_2x_2 + \tilde{w}_3x_3 + \tilde{b}
\end{aligned}$$

The new weights, e.g. $\tilde{w}_1$, can be derived by the weights of all neurons of the MLP. This problem is caused by using linear activation functions. To prevent this problem nonlinear activation functions are used like the ones we present in the following. The first activation function is the step function $\sigma(\cdot)$ [Du14, 67] (see figure 2.8(a)), where firing corresponds to 1 and not firing corresponds to 0:

$$\phi(x) = \sigma(x) = \begin{cases} 1 & \text{for} \quad x \geq 0 \\ 0 & \text{for} \quad x < 0 \end{cases} \tag{2.10}$$

Second, the sigmoid function [Du14, 67] is similar to the step function, since firing also corresponds to 1 and not firing to 0. It is visualized in figure 2.8(b) and defined by:

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{2.11}$$

In contrast to the step function and the sigmoid function, the tanh [Du14, 68] express not firing as -1 (see figure 2.8(c)). The tanh activation function is defined by:

$$\phi(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.12}$$

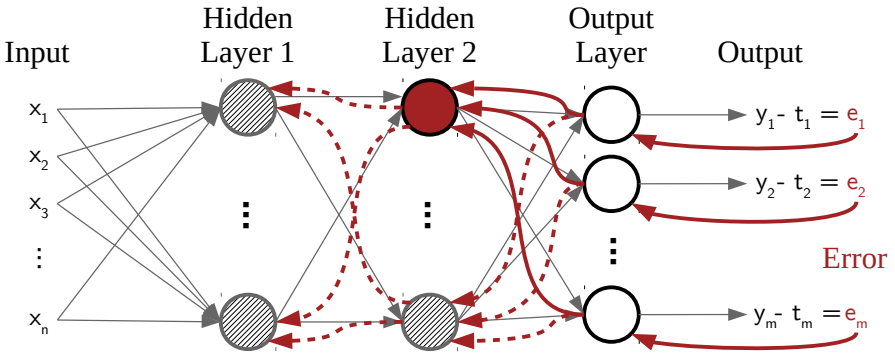


Figure 2.9: **Backpropagation:** Visualization of the backpropagation learning algorithm. First, the error is determined at the output layer and is backpropagated through the net until it arrives at hidden layers.

The last activation function, the ReLU [GBC16, 173], is not a hard limiter (threshold) function, but it is the most important activation function for CNNs. It is visualized in figure 2.8(d) and is defined by:

$$\phi(x) = \max(0, x) \tag{2.13}$$

Training - Backpropagation

Backpropagation [RHW88] is the most popular learning rule for training MLPs. Thereby, it minimizes a loss function using a gradient descent technique. One commonly used loss function is based on the Mean Squared Error (MSE) between the target output $\mathbf{t}(\mathbf{x})$ and the actual output $\mathbf{y}(\mathbf{x}|\mathbf{W})$ of a MLP with the network weights $\mathbf{W}$ (see section 2.2.3.2 for additional loss functions). The loss function is defined by MSE of each training samples of the whole training batch $\mathcal{B} = \{(\mathbf{x}_1,\ \mathbf{t}(\mathbf{x}_1)), ..., (\mathbf{x}_N,\ \mathbf{t}(\mathbf{x}_N))\}$ [Du14, 86]:

$$E_{MSE}(\mathcal{B}|\mathbf{W}) = \frac{1}{2N} \sum_{(\mathbf{x}_p,\ \mathbf{t}(\mathbf{x}_p))\in\mathcal{B}} \|\mathbf{y}(\mathbf{x}_p|\mathbf{W}) - \mathbf{t}(\mathbf{x}_p)\|^2 \tag{2.14}$$

The loss function can be minimized using gradient descent and, thus, the weight update is defined by the derivative of the loss function [Du14, 86]:

$$\begin{aligned} w_{ji} &= w_{ji} + \Delta w_{ji} \\ \Delta w_{ji} &= -\eta\ \frac{E_{MSE}(\mathcal{B}|\mathbf{W})}{\partial w_{ji}} \end{aligned} \tag{2.15}$$

Using the chain rule the derivative in equation 2.15 can be expressed as [Du14, 86]:

$$\frac{E_{MSE}(\mathcal{B}|\mathbf{W})}{\partial w_{ji}} = \frac{E_{MSE}(\mathcal{B}|\mathbf{W})}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}} \tag{2.16}$$

While the second factor can be derived easily [Du14, 86], the first factor is derived depending on whether the weight corresponds to an output or a hidden neuron. If the weight

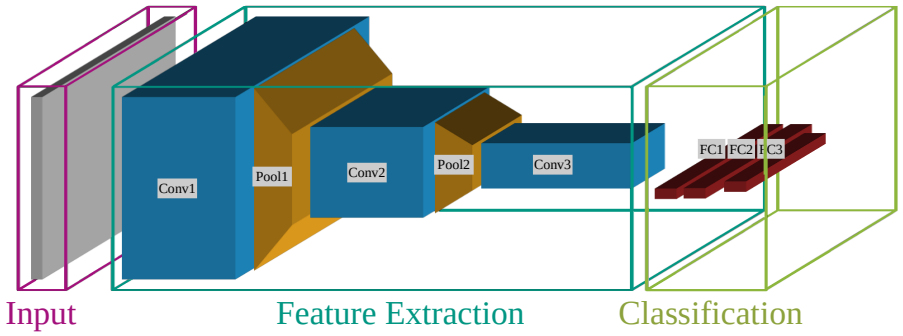


Figure 2.10: **Architecture:** Typical deep CNN architecture for classification consisting of the input, the feature extraction and the classification part.

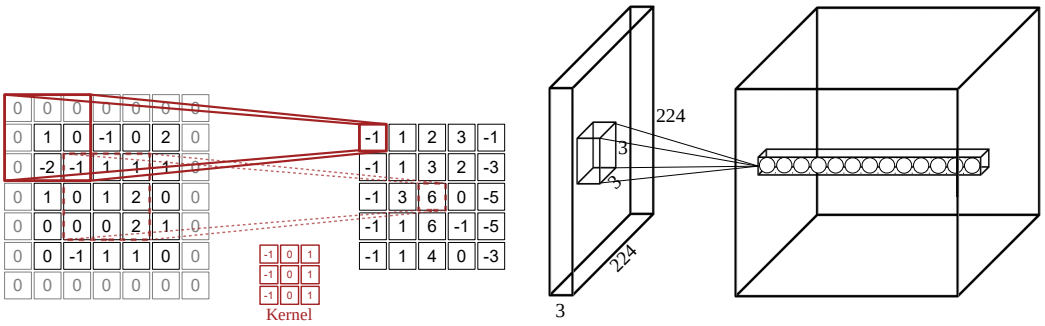
corresponds to an output neuron the first factor can also be derived easily [Du14, 87]. In case of a hidden neuron the first factor depends on the derivatives of the output neurons and later hidden neurons. Due to this dependency this training algorithm is called backpropagation. First, the error is determined at the output and this error is used to update the weights of the output neurons. Then, the error is backpropagated through the network and updates the weights of the hidden layers. Figure 2.9 illustrates the backpropagation of the error to a specific hidden neuron.

2.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special type of MLPs and are the commonly used network type in computer vision. They are based on the two-dimensional convolution operation and leverage three important ideas: *sparse connectivity*, *weight sharing* and *equivariant representations* [GBC16, 335].

In standard MLPs each hidden neuron is connected to every neuron of the previous layer. In terms of images each neuron in the first hidden layer is connected to all pixels in the image. Thus, the number of weights depends on the size of the input image. For an input image size of 28×28 each neuron in the first hidden layer has to learn $28 \cdot 28 = 784$ weights plus the bias. Although this number of weights per hidden neuron seems manageable, the number of weights explosively increase with a larger input size. First, images are normally larger than 28×28 and, second, have more input channels (e.g. colored images). Thus, the number of weights per neuron for a $256 \times 256 \times 3$ input image size increases to $256 \cdot 256 \cdot 3 = 196,608$. Moreover, the larger the input image size the more hidden neurons to represent the data and the more training samples are needed. CNNs avoid this problem by *sparse connectivity* [GBC16, 335]. Instead of fully-connected neurons, the neurons in a CNN are only connected to local regions in the previous layer and, thus, the amount of weights is reduced. The motivation behind *sparse connectivity* is that small meaningful features like edges can be computed using only tens or hundreds of pixels.

The second idea is *weight sharing* [GBC16, 335]. Thereby, several neurons in the same layer use the same weights and, therefore, apply the same operation. For example the CNN can learn edges as a feature for every local region in the image by sharing their weights. Thus, *weight sharing* reduces the number of weights as well as *sparse connectivity*. Because of this particular form of weight sharing, the convolution has a further property:



(a) Convolution Operation (b) Convolution Layer

Figure 2.11: **Convolutional Layer:** (a) illustrates the convolution operation with stride=1 and zero padding, (b) illustrates a convolution layer.

equivariance [GBC16, 338] to translation. If the input of a function changes and the output changes in the same way, this is called *equivariance*. Mathematically expressed this is $f(g(x)) = g(f(x))$. In case of the convolution operation, no matter whether a translation is applied before or after the convolution, the output is the same.

2.2.3.1 Layers

In the following, we present the different layers of a CNN in detail. Figure 2.10 illustrates a typical CNN architecture for classification and its layers. The architecture can be divided in three parts: input, feature extraction and classification. Previous approaches in computer vision used hand-crafted features like Scale Invariant Feature Transform (SIFT) [Low99] and Histogram of Orientated Gradients (HOG) [DT05] to represent the data and classify the features using a classifier like a MLP or a Support Vector Machine (SVM) [CV95]. In comparison, deep CNNs learn both the features and the classifier.

Convolutional Layer

The two-dimensional convolution between an image I and a kernel K is defined as [GBC16, 332]:

$$S(i,j) = (I*K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.17}$$

This operation is similar to the cross-correlation [GBC16, 333]. The cross-correlation and, thus, the convolution operation can be seen as a "comparison" between a kernel and the image. Since the kernel is normally smaller than the image, the kernel is pushed over the image and determines a "local similarity". The more similar the higher the convolution response (see figure 2.11(a)).

The convolutional layer is based on this operation and unifies the three previously mentioned ideas. A convolutional layer consists of a specific number of channels. Each channel learns one kernel (e.g. edge filter) by *weight sharing*. The size of the kernel is user specific, but is typically set to 3×3 (see figure 2.11(b)) and, thus, is smaller than the actually input (*sparse connectivity*). The stride defines the location offset between two convolution responses. Thus, a stride greater than one decreases the input size. In figure 2.11(a) the

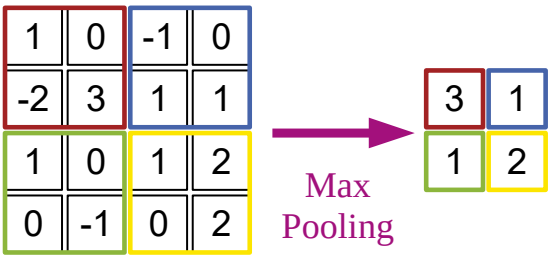


Figure 2.12: **Max Pooling:** Max Pooling Operation with size=2 and stride=2.

convolution is applied with stride one. Despite applying a convolution with stride one, the input size is slightly decreased depending on the kernel size. For a kernel size of 3×3 the input size decreases by one pixel at every boundary. To prevent decreasing of the input size, padding can be used, e.g. zero padding (see figure 2.11(a)).

Pooling Layer

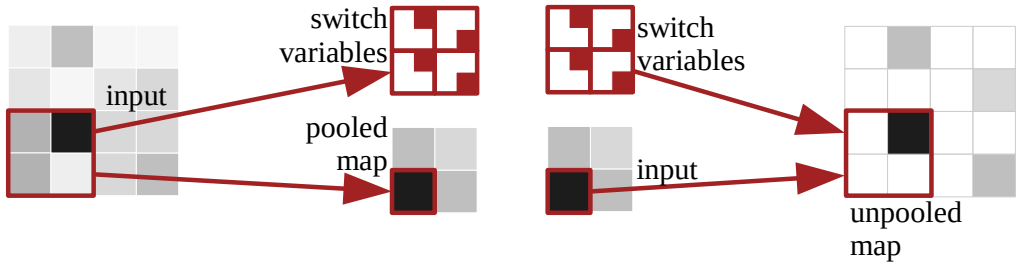
Since the kernel size of a convolution layer is typically set to 3×3, a CNN can not compute high level features, i.e. features with large receptive field (the region a kernel considers in the input image). Additionally to a convolutional layer with stride higher than one, a pooling layer can be used to increase the receptive field. A pooling layer modifies the output of the layer further [GBC16, 339] by a summary statistic of the nearby outputs. For example, the max pooling chooses the maximum within a rectangular neighborhood (see figure 2.12) and, therefore, shrinks the input size. Using max pooling approximately make the CNN invariant to small translations of the input [GBC16, 342].

In our case a Region of Interest (RoI) is bounding box proposal within a possible object. If the network learns whether this region is a good proposal or not, each RoI should be mapped to the same size. Since fully-connected layers always need the same input size, Girshick et al. proposed RoI pooling based on max pooling [Gir15]. Depending on the input size, the size of the max pooling has to be adapted to always convert the input into fix sized feature maps.

Fully-connected Layer

A combination of pooling layers and convolutional layers are used for feature extraction. At the beginning convolutional layers learn low level features like edges [ZF14]. Pooling layers increase the receptive field and, thus, convolutional layers can learn higher level features. At the end, fully-connected layers are typically used for classification (see figure 2.10). The first fully-connected layer converts the last feature map in a feature vector, i.e. a representation of the whole image. The last fully-connected layer converts this feature to a vector depending on the number of classes. In case of fully-connected layers for classification, a softmax layer is normally used. Classifying the input into C classes, leads to C output neurons, since the output of the last fully-connected layer depends on the number of classes. The final output $\hat{o}$ is defined by the softmax operation [Du14, 95]:

$$\hat{o}_i = \frac{e^{o_i}}{\sum_{j=1}^{C} e^{o_j}} \tag{2.18}$$



(a) Pooling (b) Unpooling

Figure 2.13: **Unpooling Layer:** (a) illustrates the pooling operation, (b) illustrates the unpooling operation [NHH15].

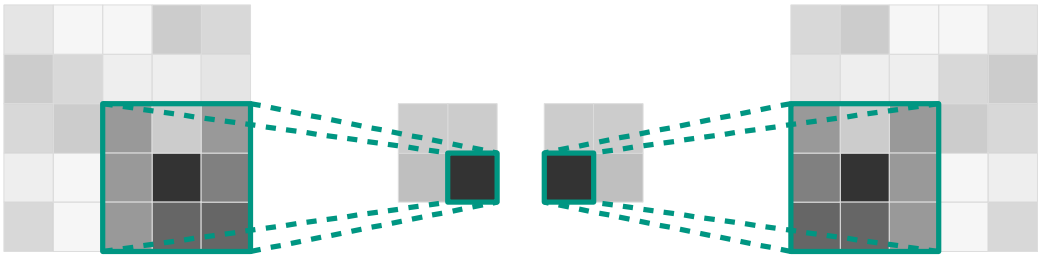
The softmax layer maps each output to a value between 0 and 1. Furthermore, the outputs sum up to 1. Therefore, each output can be interpret as an a-posteriori probability of the corresponding class.

Unpooling Layer

During pooling, spatial information within a receptive field is lost [NHH15]. This is a problem if CNNs are used for image segmentation since segmentation requires precise localization. Therefore, Noh et al. proposed unpooling layers [NHH15] to reconstruct the original size of activations as illustrated in figure 2.13. Noh et al. record the location of the maximum activation during the pooling operation in switch variables (see figure 2.13(a)). These locations are used to place each activation back to its original pooled location (see figure 2.13(b)). This strategy can be used to reconstruct the structure of the input object.

Deconvolution Layer

The output of unpooling layers are enlarged and sparse feature maps (see figure 2.13(b)). Noh et al. proposed deconvolution layers [NHH15] to densify the sparse activations obtained by unpooling through convolution-like operations with multiple learned filters. Convolution layers map multiple input values within a filter window to a single value.



(a) Convolution (b) Deconvolution

Figure 2.14: **Deconvolution Layer:** (a) illustrates the convolution operation, (b) illustrates the deconvolution operation [NHH15].

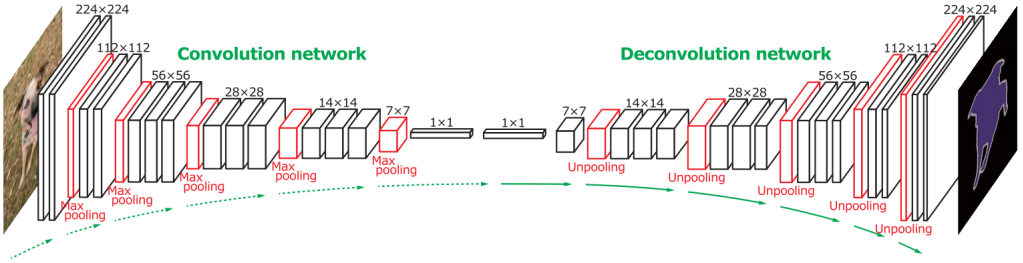


Figure 2.15: **Segmentation Architecture:** Architecture for semantic segmentation proposed by Noh et al. using deconvolution and unpooling [NHH15].

In contrast to convolution layers, deconvolution layers map single values to multiple values (see figure 2.14(b)). Thus, the output of deconvolution layers is an enlarged and dense feature map. Noh et al. crop the boundary to keep the size of the output map identical to the size of the input map. The learned filters can reconstruct the shape of the input image and are structured hierarchically to capture different levels of shape details [NHH15] (see figure 2.15). Filters in lower layers rather capture the overall shape of an object while higher layers capture class-specific fine-details.

2.2.3.2 Loss Functions

In this section we focus on three loss functions: MSE loss function, Cross Entropy (CE) loss function and L_1 loss function. We already introduced MSE loss function in equation 2.14 as the summed up squared error of all training samples. According to Du et al. [Du14, 38] the MSE function assumes independence and Gaussianity of the target data. Due to the discrete nature of classification problems, Gaussianity assumption of the target data is not valid [Du14, 38]. Moreover, MSE is a function of absolute errors and, therefore, tends to produce large relative errors for small output values [Du14, 38]. CE is a function of relative errors and is expected to estimate more accurately small probabilities [Du14, 38]. The CE loss function is defined by [Du14, 38]:

$$E_{CE}(\mathcal{B}|\mathbf{W}) = -\sum_{(\mathbf{x}_p,\ \mathbf{t}(\mathbf{x}_p))\in\mathcal{B}}\sum_{k=1}^{C} t_k(\mathbf{x}_p)\ln(y_k(\mathbf{x}_p|\mathbf{W})) \tag{2.19}$$

where $(\mathbf{x}_p, \mathbf{t}(\mathbf{x}_p))$ is one of N training samples, $t_k(\mathbf{x}_p) \in \{0, 1\}$ and C the number of classes.

The third loss proposed by Girshick et al. [Gir15] is specifically for bounding box regression. The loss is defined by position and size of a predicted bounding box $\mathbf{p} = (p_x, p_y, p_w, p_h)$ and the position and size of a Ground Truth (GT) bounding box $\mathbf{t}^k = (t_x^k, t_y^k, t_w^k, t_h^k)$ [Gir15]:

$$E_{loc}(\mathbf{p}, \mathbf{t}^k) = \sum_{i\in\{x,y,w,h\}} \text{smooth}_{L_1}(t_i^k - p_i) \tag{2.20}$$

The the L_1 smoothing function is defined by [Gir15]:

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5 \cdot x^2 & ,\ |x| < 1 \\ |x| - 0.5 & ,\ \text{otherwise} \end{cases} \tag{2.21}$$

This loss is a robust L_1 loss that is less sensitive to outliers compared to other regression losses.

Algorithm 2 Stochastic Gradient Descent (SGD) [GBC16, 294]

Require: learning rate η
Require: initial weights $\mathbf{W}$
  while stopping criterion not met **do**
    # Sample a minibatch of M training samples
    $\mathcal{B}_M = \{(\mathbf{x}_1, \mathbf{t}(\mathbf{x}_1)), ..., (\mathbf{x}_M, \mathbf{t}(\mathbf{x}_M))\}$
    # Compute gradient
    $\Delta w_{ji} = -\eta \cdot \frac{\partial E(\mathcal{B}_M|\mathbf{W})}{\partial w_{ji}}$
    # Apply weight update:
    $w_{ji} = w_{ji} + \Delta w_{ji}$
  end while

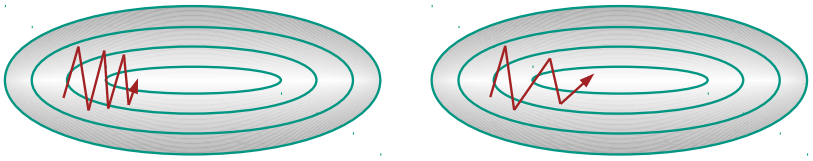
2.2.3.3 Optimizer

We previously presented the gradient descent algorithm (see section 2.2.2) for training neural networks. The gradient of the loss function is computed using the entire training set. In general, the training algorithm follows this gradient downhill to a (local) minimum in the loss function [GBC16, 294]. If the learning rate is to high, the training algorithm jumps around near the (local) minimum.

SGD is the most commonly used optimization algorithms for deep learning and accelerates the gradient descent algorithm by following the gradient of minibatches downhill [GBC16, 294]. A minibatch is a randomly selected subset of the entire training set. The error formulation in equation 2.14 can be adapted to SGD minibatch training by using only M samples of a minibatch instead of all N training samples ($M < N$).

Algorithm 2 shows the Stochastic Gradient Descent (SGD) in detail, where E could be the MSE error (see equation 2.14) or the CE error (see equation 2.19). According to Goodfellow et al. the learning rate is a crucial parameter [GBC16, 294]. In practice, it is necessary to gradually decrease the learning rate over time and is best chosen by monitoring learning curves that plot the loss function as a function of time [GBC16, 294]. The reason is that minibatches, i.e. random sets of training samples, introduces noise that does not vanish even if a minimum of the loss function is reached [GBC16, 294]. In contrast, the loss function oscillates around the minimum due to this introduced noise. This gets clear by comparing to training with the full training set (batch training). Thereby, the gradient of the total loss function becomes small and then zero when the minimum is approached [GBC16, 294]. Thus, a fixed learning rate can be used for batch training. Another important property of SGD is that the computation time per update does not increase with the number of training examples because of the fix size of a minibatch [GBC16, 294]. For very large datasets this property allows convergence before processing the entire training set [GBC16, 294].

Although SGD accelerates the training of neural networks, introducing a momentum to the learning algorithm can accelerate the training even more. The momentum algorithm also takes past gradients into account and accumulates them by an exponentially decaying moving average. Figure 2.16 shows the acceleration of SGD using a momentum term. It accelerates SGD in the relevant direction and suppresses oscillations. Figure 2.16 illustrates a two-dimensional projection of a loss function that has the form of a long shallow ravine



(a) SGD without momentum (b) SGD with momentum

Figure 2.16: **Momentum:** Acceleration of SGD by using a momentum term.

leading to the optimum with steep walls on two sides. Since the gradient points down the steep sides, the SGD starts oscillating. For SGD with momentum the weight update in algorithm 2 is redefined using a velocity v_{ji} [GBC16, 296]:

$$\begin{aligned} v &= \alpha \cdot v_{ji} - \eta \cdot \frac{1}{M} \cdot \sum_{k=1}^{N} \left(\frac{\partial E_k(\mathbf{W})}{\partial w_{ji}} \right) \\ w_{ji} &= w_{ji} + v_{ji} \end{aligned} \tag{2.22}$$

The larger α is relative to η, the more previous gradients affect the current direction.

2.2.3.4 Regularization

In the following, we present weight decay as a regularization method for neural networks [GBC16, 119]. The goal of regularization is to prevent overfitting and lead to better generalization. If a neural network overfits, then the neural network achieves good results on the training data but achieves worse results on the test data. For this purpose, weight decay penalizes large weights by adding an additional term in the loss function $\tilde{E}(\mathcal{B}|\mathbf{w})$ [GBC16, 119]:

$$\tilde{E}(\mathcal{B}|\mathbf{w}) = E(\mathcal{B}|\mathbf{w}) + \lambda \cdot \mathbf{w}^\top \cdot \mathbf{w} \tag{2.23}$$

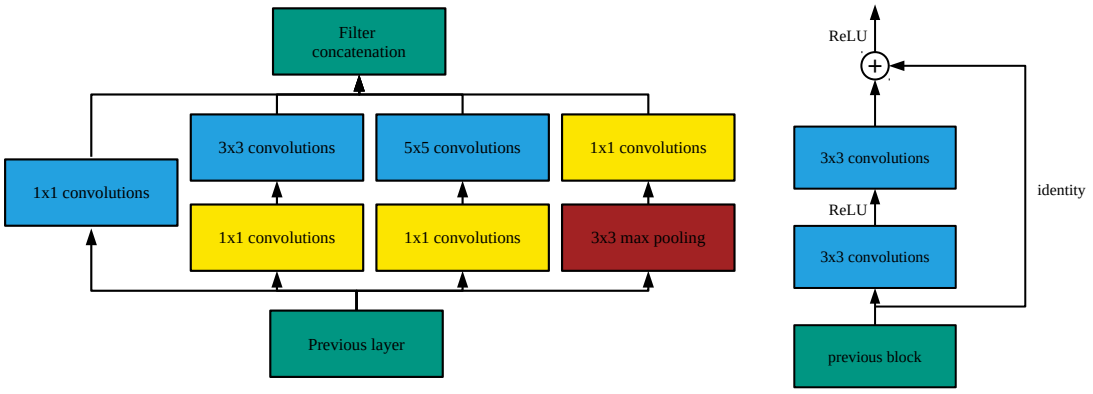
where w defines all parameters as a vector. The regularization parameter λ determines the trade-off between original loss function and large weights penalization. In practice, weight decay penalizes not only large weights but also limits the freedom in the model. Furthermore, the weight update rule changes to:

$$\begin{aligned} w_{ji} &= w_{ji} - \eta \frac{\tilde{E}(\mathcal{B}|\mathbf{w})}{\partial w_{ji}} \\ &= w_{ji} - \eta \frac{E(\mathcal{B}|\mathbf{w})}{\partial w_{ji}} - \eta\lambda w_{ji} \end{aligned} \tag{2.24}$$

where $-\eta\lambda w_{ji}$ results from the new regularization term in the loss function.

2.3 Image Classification

The image classification task is the determination of all objects that occur in an image. However, classification networks [KSH12, SZ14, SLJ+15, HZRS16] do not provide object locations within an image. The classification networks are typically divided in three parts



(a) Inception Module of GoogLeNet [SLJ+15].

(b) Exemplary Residual Block of ResNet [HZRS16].

Figure 2.17: **Innovations:** Innovations in CNN architecture.

(see figure 2.10): input, feature extraction (convolutional layers) and classification (fully-connected layers). Classification networks are not only used for image classification but also can serve as a feature extractor in other tasks (e.g. image segmentation) by dropping the classification part [LSD15, NHH15, KHH17, DHS15b, DHS16, KBH+17].

Krizhevsky et al. proposed a network, called AlexNet, in 2012 that outperformed other methods in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition from 2012 by a major of 10.9% [KSH12]. Since then deep learning and CNNs continue unabated achieving state-of-the-art results in computer vision tasks. The network consists of five convolutional layers each followed by pooling layers. For classification, fully-connected layers follow the convolutional layers.

A second architecture that wins ILSVRC-2014 competition in image classification is VGG proposed in 2014 by Simonyan et al. [SZ14]. The authors increased the number of layers with learnable weights to 16–19 compared to 8 weight layers of the AlexNet. VGG consists of stacked convolutional layers (2-3) with a kernel size of 3. Each stack is followed by a pooling layer. According to the basic architecture, the last three layers are fully-connected layers for classification.

Instead of improving the performance by increasing the network size, Szegedy et al. propose the GoogLeNet [SLJ+15] using an inception module. Typically, during the design of the network the researcher has to decide which kernel size at which stage is used. An inception module combines 1×1, 3×3 and 5×5 convolutional layers and, therefore, can learn which layer is best at which stage (see figure 2.17(a)). The authors use the 1×1 convolutions to reduce the dimensionality to keep the computations reasonable. Their Inception network, GoogLeNet, consists of this type of Inception module and additional pooling layers to halve the feature map size. In higher layers Szegedy et al. use traditional convolution layers and, finally, a fully-connected layer for classification.

In 2016 He et al. proposed a very deep classification network (up to 152 layers), called Residual Network (ResNet) [HZRS16]. It won the ILSVRC-2015 classification task with a 3.57% test error rate. The main part of the ResNet is the residual block (see figure 2.17(b)).

The network uses in each block the identity or a learned filter map using some convolution layers. This made inputs of a lower layers available to a nodes in a higher layers. Veit et al. showed that a ResNet can be seen as an ensemble of shallow networks [VWB16]. The whole network is built of these residual blocks and one fully-connected layer at the end of the network for classification.

2.4 Image Segmentation

In general, segmentation is defined as the subdivision into smaller parts, also called segments. Entities of the same segment exhibit similar structural properties, while entities of different segments should exhibit different structural properties. In summary, segmentation means assigning each entity to one of these segments. In image segmentation, this corresponds to assigning to each pixel a label. Each label represents one segment. Depending on the meaning of these labels, we distinguish between semantic segmentation (figure 2.18(b)) and instance segmentation (figure 2.18(c)). In semantic segmentation the labels corresponds to object classes (e.g. person, plane, ...) and, therefore, the assignment is class-aware. In instance segmentation the labels corresponds to instances of objects and, thus differentiates between two objects (e.g. person_1, person_2).

(a) Original Image (b) Semantic Segmentation (c) Instance Segmentation

Figure 2.18: **Image Segmentation:** Different types of image segmentation.

2.5 Datasets

In this section, we present four different datasets with different number of images and different annotations. Thereby, the SBD dataset is only an extension of the PascalVOC dataset. Table 2.1 compares these datasets regarding to their number of images and their classes.

ImageNet

The first dataset we present is ImageNet [DDS+09] that contains 14,197,122 images. The corresponding benchmark in object classification is called ILSVRC and includes 1,431,167 images grouped in 1,000 classes. Neural networks trained for object classification on ILSVRC are additionally very good feature extractors (see examples in section 2.3). Therefore, researchers often use these pretrained networks as a feature extracting part in their networks to solve other computer vision tasks like image segmentation.

Dataset	Task	Classes	Training images	Validation images	Test images
ImageNet [DDS+09]	image classification	1,000	1,281,167	50,000	100,000
Cityscapes [COR+16]	segmentation	30	2,975	500	1,525
MS COCO [LMB+14]	segmentation	91	165,482	81,208	81,434
PascalVOC [EEVG+15]	segmentation, object detection	20	1,464	1,449	1,456
SBD [HAB+11]	segmentation, object detection	20	5,623	5,695	1,456[1]

Table 2.1: **Dataset Comparison:** Comparison of datasets regarding the number of images and the number of classes.

Cityscapes

Cityscapes [COR+16] is a benchmark dataset for semantic and instance segmentation. It includes 5,000 fully labeled images recorded in streets from 50 different cities. Therefore, the 30 proposed object classes are strongly related to traffic situations. Figure 2.19 shows the 30 classes grouped in 8 different categories. Cityscapes provides instance-aware segmentation for vehicle and humans, while providing semantic segmentation for the other categories.

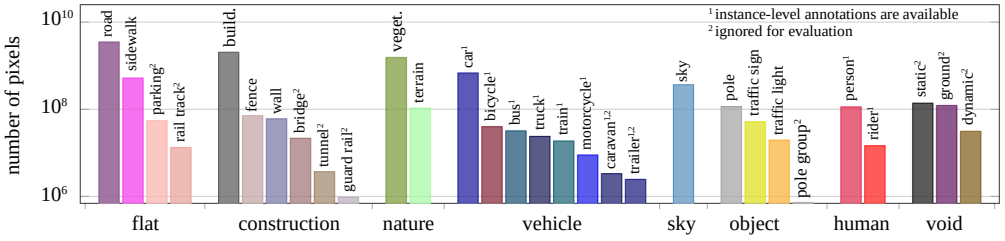


Figure 2.19: **Class Distribution:** Number annotated pixels (y-axis) per class and their associated categories (x-axis) [COR+16].

PascalVOC and SBD

PascalVOC [EEVG+15] is a benchmark dataset for both object detection and segmentation. It includes 4,369 images for both semantic and instance segmentation and, furthermore, provides object location in form of bounding boxes. These images are grouped into 20 different classes:

Vehicles aeroplane, bicycle, boat, bus, car, motorbike, train

Household bottle, chair, dining table, potted plant, sofa, tv/monitor

[1] The test set is similar to the test set of PascalVOC.

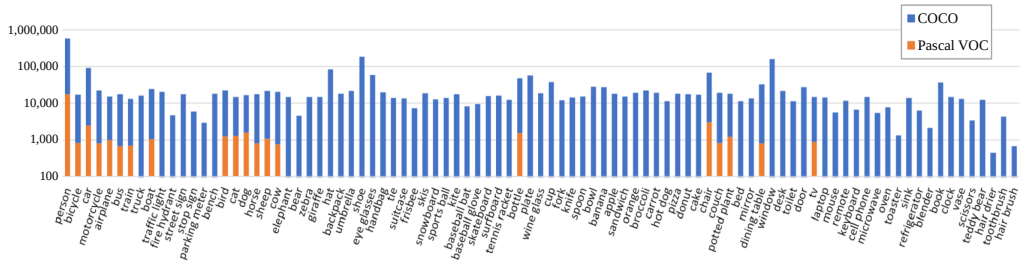


Figure 2.20: **Class Distribution:** Instances per class of PascalVOC and MS COCO [LMB+14].

Animals bird, cat, cow, dog, horse, sheep

Others person

In contrast to the Cityscape dataset that only focuses on traffic situations, the PascalVOC dataset focuses on object segmentation in any situation. Figure 2.20 compares the number of instances per class of PascalVOC and MS COCO. Moreover, PascalVOC provides bounding box annotations for their images.

PascalVOC includes more images than available segmentation mask annotations. The Semantic Boundaries Dataset (SBD) [HAB+11] is an extension that provides segmentation masks for each image in PascalVOC. SBD includes 11,318 training and validation images and has the same test set than PascalVOC.

MS COCO

MS COCO [LMB+14] is a dataset that provides instance-aware segmentation masks for 328,124 images. In contrast to the 20 classes of the PascalVOC dataset, the MS COCO dataset includes 91 object classes. Figure 2.20 shows the number of instances per class. Compared to PascalVOC, MS COCO provides significantly more segmentation annotations.

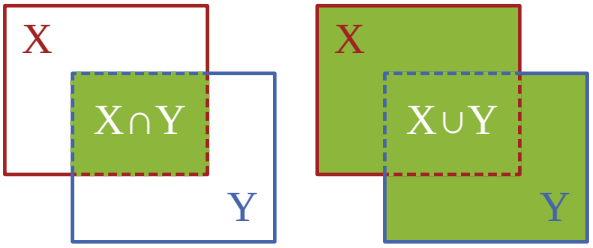
2.6 Evaluation Metrics

In this section, we present different metrics to evaluate methods in image segmentation. Thereby, we distinguish between metrics for semantic segmentation and instance segmentation.

Semantic Segmentation

The most common used metric for semantic segmentation is the Intersection over Union (IoU) [LSD15, NHH15, DHS15a, PCMY15, KBH+17, KL16]. Thereby, the intersection of the GT mask and the predicted mask (see figure 2.21(a)) is compared with the union of both (see figure 2.21(b)) and, thus, IoU is defined by [EEVG+15]:

$$IoU(X,Y) = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{area(X) \cap area(Y)}{area(X) \cup area(Y)} \tag{2.25}$$



(a) Intersection of X and Y (b) Union of X and Y

Figure 2.21: IoU using the intersection in (a) and the union in (b).

Since we want to evaluate the the IoU per class, we use the definition of the mean Intersection over Union (mIoU) metric presented by Long et al. [LSD15] for our evaluation:

$$\text{mIoU} = \frac{1}{C} \cdot \sum_i \left(\frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}} \right) \tag{2.26}$$

where n_{ij} define the number of pixels of class i predicted to belong to class j, C defines the number of classes and $t_i = \sum_j n_{ij}$ is the total number of pixels of class i.

Instance Segmentation

For instance segmentation we use the same metric as Khoreva et al. [KBH+17] and Dai et al. [DHS16], the mean Average Precision (mAP) per class [HAGM14]. This metric is derived from the mAP metric for object detection presented by Everingham et al. [EEVG+15] and is based on the precision. The precision P is defined by:

$$P = \frac{TP}{TP + FP} \tag{2.27}$$

where True Positive (TP) defines all correctly predicted masks and False Positive (FP) all falsely predicted masks. To decide whether a prediction mask is a TP or a FP the IoU defined in equation 2.25 is used [EEVG+15]. The predicted instance segmentation masks are sorted by their confidence and, afterwards, each segmentation mask is compared with each GT mask. If the IoU of an instance segmentation masks is higher than a certain threshold, this masks is a TP, otherwise a FP. Furthermore, GT masks with no matching and multiple matchings of the same GT mask are also treated as FP.

In addition, the recall R, also called sensitivity, is defined by:

$$R = \frac{TP}{TP + FN} \tag{2.28}$$

where True Negative (TN) defines all masks in the GT that are not found. For a given class, the precision–recall curve is computed from these ranked output by changing the threshold in the prediction confidence. The mAP is the area under this curve and can be approximated by a sum over the precisions at every possible threshold value, multiplied by the change in recall:

$$mAP = \sum_{k=1}^{N} P(k) \cdot \Delta r(k) \tag{2.29}$$

where N is the total number of images in the collection, $P(k)$ is the precision at a cutoff of k images and $\Delta r(k)$ is the change in recall that happened between cutoff $k-1$ and cutoff k. By using not every possible threshold value, but only a subset the area under the curve can be also approximated. Thereby, eleven values are commonly used and, hence, the corresponding metric is called "11-Point metric".

Moreover, we use the Average Best Overlap (ABO) metric [PTVG15] also used by Khoreva et al. [KBH+17]. Each GT object segmentation mask g_i is compared to the most promising (same class) predicted object segmentation mask l_j using the overlap (see equation 2.25). The ABO metric is then defined as [UVDSGS13]:

$$ABO = \frac{1}{|\mathcal{G}|} \sum_{g_i \in \mathcal{G}} \max_{l_j \in \mathcal{L}} IoU(g_i, l_j) \tag{2.30}$$

where $\mathcal{G}$ defines all GT masks.

3. Related Work

In this chapter, we present current research related to the topic of this thesis. First, we introduce to several foreground-background segmentation algorithms (see section 3.1). Then, we present current research in semantic segmentation, both supervised and weakly-supervised (see section 3.2). Finally, we present supervised and weakly-supervised methods for instance segmentation (see section 3.3).

3.1 Foreground-Background Segmentation

In this section, we present foreground-background segmentation algorithms [ZLWS14, ZSL+15, RKB04, APTB+14]. Since we use them for our baselines and our end-to-end trained model, we describe these algorithms more detailed in section 4.1.2.

Both Robust Background Detection (RBD) proposed by Zhu et al. [ZLWS14] and Minimum Barrier Salient Object Detection (MBOD) proposed by Zhang et al. [ZSL+15] detect salient objects within an image. Thereby, they assume that objects are located in the middle of an image and objects differ from the background. Therefore, Zhu et al. [ZLWS14] propose a robust background measure for superpixels, called boundary connectivity. Furthermore, the authors integrate the output of this measure, in addition to other low level cues, into an optimization framework to obtain clean and uniform saliency maps. Zhang et al. [ZSL+15] generate a saliency map by computing the Minimum Barrier Distance (MBD) from each pixel to the boundaries.

GrabCut is a segmentation algorithm proposed by Rother et al. [RKB04]. Thereby, the image is converted into a graph, where each node represents one specific pixel. If two pixels are adjacent, then the corresponding nodes are connected. The weight between these nodes correspond to their pixel similarity. Two further nodes are added to the graph: first, the foreground terminal and, second, the background terminal. By initial segmentation information (e.g. bounding box) a foreground and background model is learned. Each node is connected to these two terminals, while the weights are defined by the similarity the corresponding model. Finally, foreground-background segmentation is achieved by applying a minimal cut in this graph.

Arbeláez et al. [APTB+14] propose an object segmentation generator, called Multiscale Combinatorial Grouping (MCG). The authors develop a high-performance hierarchical segmenter that makes effective use of multi-scale information. Finally, a grouping strategy is applied that combines multiscale regions into highly-accurate object candidates.

3.2 Semantic Segmentation

In this section, we present research in semantic segmentation. We distinguish between supervised and weakly-supervised methods.

3.2.1 Fully-Supervised

The success of the classification network AlexNet [KSH12] in 2012 pushes the development in deep learning forward. The progress in deep learning enhances also the results in semantic segmentation. Thus, deep learning approaches achieve state-of-the-art results in this task. Typical deep learning networks decrease the feature maps using pooling layers (see section 2.2.3.1) and, thus, increase the receptive field. However, this contradicts the pixel accurate labeling in semantic segmentation. Therefore, deep learning networks in this form are not suitable for this task. In 2015, both Long et al. [LSD15] and Noh et al. [NHH15] proposed an end-to-end network for semantic segmentation and both circumvent the problem of decreasing feature map size.

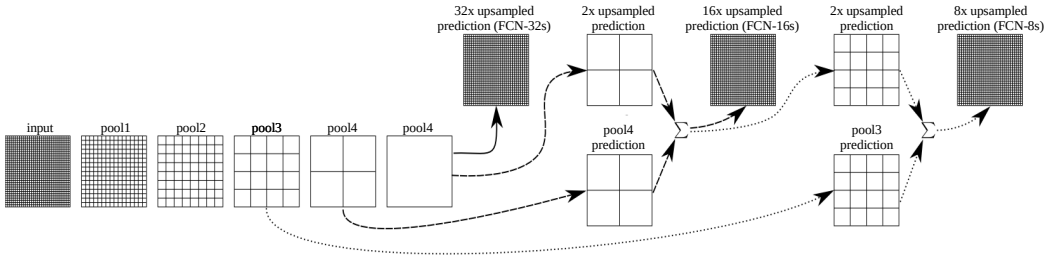


Figure 3.1: **FCN:** Combination of feature maps in a FCN [LSD15].

Long et al. [LSD15] proposed a network for semantic segmentation, called Fully Convolutional Network (FCN), which, as the name suggests, only contains convolutional layers. FCNs use the VGG classification network [SZ14] to represent the input image by feature maps and increase these feature maps using simple bilinear interpolation. Instead of increasing only the last feature map, the authors use this bilinear interpolation at several positions in the network and combine the increased feature with previous feature maps (see figure 3.1)

Noh et al. [NHH15] propose a network, also based on the VGG classification network, that is composed of deconvolution and unpooling layers. The unpooling layers and deconvolution layers are used to learn the increase of feature maps and, therefore, learn pixel-wise label assignment. The whole network is built of a convolutional network (VGG) and a deconvolution network (see figure 2.15). Lower layers in the deconvolution network tend to capture the overall coarse configuration of an object (e.g. location, shape and region), while higher layers discover more complex patterns.

3.2.2 Weakly-Supervised

To reduce the annotation effort, different researchers proposed weakly-supervised methods for semantic segmentation. Thereby, the researchers focus on different types of supervision (see figure 3.2).

Figure 3.2: **Supervision:** Different types of supervision and their annotation effort.

Image Tag Supervision

Several papers [PC15, PCMY15, WLC+16, PKD15, KL16] use image tags for training weakly-supervised networks. Pinheiro et al. [PC15] and Pathak et al. [PKD15] used a network based on Multiple Instance Learning (MIL). MIL is a variation of weakly-supervised learning [MLP98]. Instead of instances the learning algorithm receives positive and negative bags of instances [MLP98]. If a bag contains at least one positive instance, it is a positive bag. As a consequence, only if all instances in a bag are negative the bag is a negative one. MIL is learning a concept to label individual instances correctly given positive and negative bags [MLP98].

Pinheiro et al. [PC15] treat images as MIL bags. Images that contain at least one foreground pixel corresponding to the given image tag are positive bags. Images that contain not a single foreground pixel are negative bags. The authors propose a CNN that focuses during training on pixels that are important for classifying the image.

Pathak et al. [PKD15] propose a CNN and introduce constrained learning. They alternate between fulfilling the constrains by optimization and learning the optimized segmentation masks. They propose several constraints like background and foreground constraints. The foreground constraint, for example, corresponds to a minimal number of foreground pixels per image tag class, that is similar to the MIL paradigm.

Papandreou et al. [PCMY15] developed an Expectation Maximization (EM) method to train a CNN for semantic segmentation based on image tags. While the expectation step increases pixel values with correct class according to the image tags, the maximization step learns this new segmentation mask (with increased pixel values).

Kwak et al. [KHH17] propose a Superpixel Pooling Network (SPN) trained using only image tags to generate initial segmentation annotations. The authors use VGG as feature extractor and pool the last feature map using superpixels (caused by oversegmentation).

Thus, each superpixel is represented by a feature vector and is classified afterwards. The initial annotation masks are used to train another network that estimates pixel-wise semantic labels. The second network is based on the decoupled network proposed by Hong et al. [HNH15]. This network architecture decouples semantic segmentation task into classification and segmentation.

Wei et al. [WLC+16] used Self-Paced Learning (SPL) for their neural network. Thereby, SPL means first learning from simple and easy images and then learning from complex ones. The complexity of images strongly depend on the background and the number of objects. Therefore, they propose three networks for different image complexities. The authors use the output of earlier complexity stages for the later complexity stages to segment all images in the end.

Kolesnikov et al. [KL16] propose a segmentation CNN that segments objects incorporating object locations provided by the approach from Zhou et al. [ZKL+16]. Thereby, Zhou et al. learns object localization within an image without any bounding box supervision. Kolesnikov et al. combine three different losses to train their CNN using image tag supervision. First, the results of a segmentation CNN and the locations are used for their semi-supervised loss. Second, their classification loss examines consistent labeling with the given image tags. Finally, the boundary-aware loss penalizes segmentations that are discontinuous with respect to spatial and color information in the input image.

Point and Scribble Supervision

Bearman et al. investigates points and scribbles as supervision [BRFFF16] and demonstrate that pointing on objects adds negligible additional annotation cost. However, using point supervision improved their results compared to other annotation types. For segmentation, the authors use the FCN model and incorporate point supervision in its training loss function.

Bounding Box Supervision

Furthermore, several researchers [KBH+17, PCMY15, DHS15a, RLO+17] focus on bounding boxes as supervision. All of them used some kind of Expectation Maximization (EM) method (see section 2.1 for more details about EM).

Papandreou et al. [PCMY15] use not only image tags as supervision but also bounding boxes. The authors use bounding boxes to generate better initial segmentation masks for their EM training scheme. For the initial segmentation mask $\alpha\%$ of the center area within the box are set to foreground. Then, the authors apply a Conditional Random Field (CRF) to this mask to generate the initial segmentation mask. This method is a preprocessing step and provides the initial masks for the first maximization step.

Dai et al. [DHS15a] assign labels to object proposals from Selective Search [UVDSGS13] or MCG [APTB+14] using a neural network based on VGG. The bounding boxes are used for the initial assignment. For improving this assignment, the authors use an iterative training procedure according to the EM scheme. During the expectation step, the neural network is used to assign a label to each object proposal. In the maximization step, the neural network learns the new assigned labels.

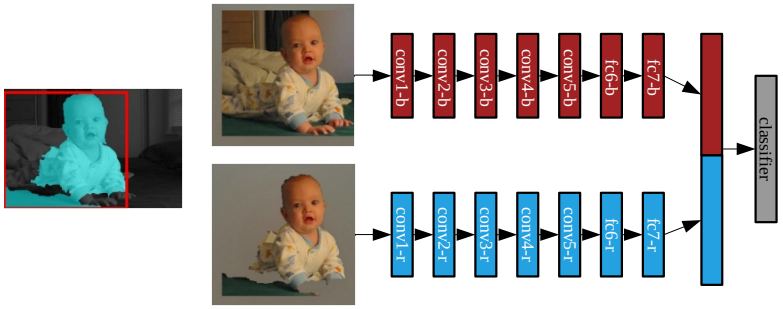


Figure 3.3: **Two Pathway Network:** Left: The region with its bounding box. Right: The network architecture with two pathways. The top pathway operates on cropped boxes and the bottom pathway operates on region foregrounds.

Khoreva et al. [KBH+17] follow the EM scheme and provide segmentation masks using bounding box supervision. Thereby, the authors use weakly generate segmentation masks within each bounding box using the intersection between GrabCut [RKB04] and MCG [APTB+14]. In the maximization step, these segmentation masks are learned from a neural network based on VGG. In the expectation step, the masks are refined using different cues: First, pixels outside boxes are reset to background. Second, if the segmentation is too small it is set to the initial box. Finally, a Dense Conditional Random Field (DenseCRF) is used to improve the segmentation masks.

In contrast to previous mentioned papers, Rajchl et al. [RLO+17] train a neural network weakly-supervised to segment organs like brains and lungs. The training is similar to the approach of Khoreva et al. [KBH+17]. In the expectation step, a DenseCRF is used to refine the segmentation.

3.3 Instance Segmentation

Finally, we present papers related to our work that segment instances. Similar to semantic segmentation, we distinguish between supervised and weakly-supervised methods.

3.3.1 Fully-Supervised

Researchers [HAGM14, HAGM15, PCD15, DHS15b, DHS16, LQD+17] propose methods for instance segmentation based on neural networks. Since this task requires both classification and segmentation of object instances, the subtasks are typically accomplished separately [HAGM14, DHS15b, HAGM15, DHS16, LQD+17]. Hariharan et al [HAGM14], Dai et al. [DHS15b] and Hariharan et al. [HAGM15] use object segmentation proposals provided by MCG [APTB+14] for instance segmentation.

Hariharan et al. [HAGM14] extract features using two separate neural networks (pathways) for classifying either object segmentation proposals provided by MCG or corresponding bounding boxes. This two pathway network is illustrated in figure 3.3. The first network learns to predict bounding boxes, while the second network learns the overlap of the proposal with the GT mask. Each network is finetuned for its specific task. In the end, the two

networks are trained jointly by adding a final layer, that combines features from both networks. The authors train a SVM to classify the feature vector and assign a class label per proposal. Moreover, the segmentation was refined by dividing the corresponding bounding box into a 10×10 grid. A classifier predicts for each grid the foreground probability. This coarse grid is fused with the segmentation generated by object proposals.

Hariharan et al. [HAGM15] build upon their previous work [HAGM14], but replace the refinement step. Thereby, the ranked hypotheses are refined by a method using "Hypercolumns". A "Hypercolumn" is a per-pixel feature vector using information of early and late convolutional layer. At several points in the CNN the feature maps are resized using bilinear filtering and summed up. Each pixel feature is then classified.

Dai et al. [DHS15b] classify object segmentation proposals using a neural network. A CNN extracts feature maps from the given image (regional features). These feature maps are masked using the object segmentation proposals. They investigate several proposal algorithms, e.g. MCG. The masked feature maps are fed into several fully-connected layers and, thus, are represented by a feature vector. According to the two pathway network of Hariharan et al. [HAGM14], segmentation features alone are insufficient. Therefore, the authors use a second pathway with the regional features and fed them in further fully-connected layers. The resulting feature vector is concatenated with the feature vector of the segmentation features and classified by a SVM inspired by the SVM used by Hariharan et al. [HAGM14].

Pinheiro et al. [PCD15] propose a neural network that predicts a segmentation mask within an image patch. By a given image patch the network first predicts the segmentation of the object in the middle of the image patch and second predicts an object score. To detect and segment all objects the neural network is applied densely at multiple locations and scales.

Dai et al. [DHS16] propose an end-to-end instance segmentation network (called Multi-task Network Cascade (MNC)) consisting of three stages: proposing bounding boxes, proposing segmentation masks within these bounding boxes, categorizing these segmentation masks. Thereby, all stages share the same feature extraction network (VGG). Moreover, later stages depend on the output of earlier stages Since we use their neural network for our weakly supervised segmentation model, we describe the paper of Dai et al. [DHS16] more detailed in section 4.2.

Li et al. [LQD+17] train an end-to-end instance segmentation network. For feature extraction the authors use a ResNet model. Then, the network propose Region of Interests (RoIs) using a Region Proposal Network (RPN). For each RoI the segmentation mask and the category is predicted. Therefore, the authors use position-senstive inside/outside score maps. Each score map is responsible for a specific region within the RoI. Thus, each score represents the likelihood of the pixel belongs to some object instance at a relative position. In contrast to RoI pooling, where a RoI is pooled to a specific size, each score map represents the whole image. To decide whether a pixel belonging to the RoI and to decide whether it is inside an object instance's boundary they use the inside/outside score map. These score maps are used to predict the final segmentation mask and the final category.

3.3.2 Weakly-Supervised

To the best of our knowledge, Khoreva et al. [KBH+17] are the only ones who address the task of weakly-supervised instance segmentation. Therefore, the authors segments objects in a two-task approach (see figure 3.4): First, their approach detects objects in form of bounding boxes using the Fast-RCNN object detector [Gir15]. Second, their approach segments objects by applying a segmentation network for each bounding box. Their segmentation network, called $\text{DeepLab}_{\text{BOX}}$, is inspired by the DeepMask network proposed by Pinheiro et al. [PCD15]. They add a fourth input channel that represents the bounding box by a binary map and guides the network to segment the object within the bounding box. As already stated, the authors use Fast-RCNN during inference but use GT bounding boxes during training. The authors generate segmentation masks within the GT bounding boxes using GrabCut [RKB04] and $\text{DeepLab}_{\text{BOX}}$ learns this segmentation masks instead of fully-supervised segmentation masks.

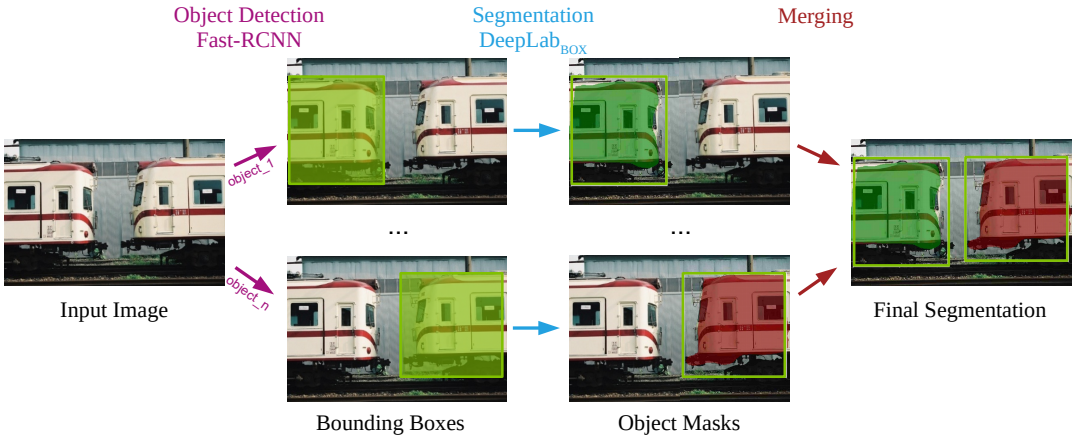


Figure 3.4: **Two-task Approach:** The two-task approach of Khoreva et al. [KBH+17] for weakly-supervised instance segmentation: (1) object detection using Fast-RCNN and (2) object segmentation using $\text{DeepLab}_{\text{BOX}}$.

4. Methods

In this chapter, we present our weakly-supervised end-to-end trained segmentation model (see section 4.2). For this purpose, we built upon the Multi-task Network Cascade (MNC) model proposed by Dai et al. [DHS16] and enhance it by enabling the use of bounding boxes as weak supervision. Instead of using pixel-wise annotated segmentation masks, we use foreground-background segmentation algorithms to weakly generate masks within each bounding box. In section 4.1, we present our baselines consisting of three independent tasks. First, the detection of objects in form of bounding boxes using YOLO [RDGF16]. Second, we use seven different foreground-background segmentation algorithms to weakly generate segmentation masks within each bounding box. Last, we use a postprocessing step to refine the segmentation masks. With the aid of these baselines we evaluate the foreground-background segmentation algorithms without training our neural network. Moreover, these baselines provide comparability to our weakly-supervised network.

4.1 Baselines Using Box-wise Segmentation

In this section, we present our three-task structured baselines (see figure 4.1). These baselines are a combination of independent algorithms. The first task is the detection of objects in form of bounding boxes with the You Only Look Once (YOLO) [RDGF16] object detector and is described in section 4.1.1. In the second task, we apply one of seven

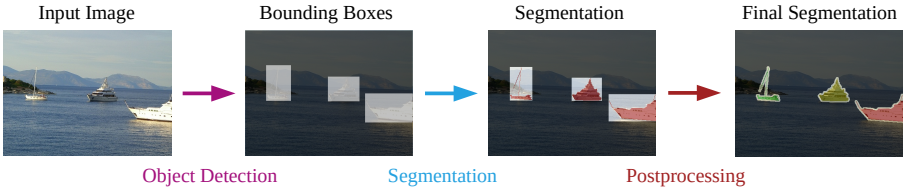


Figure 4.1: **Three-task Baseline Structure:** (1) object detection using axis aligned bounding boxes, (2) foreground-background segmentation within the bounding boxes, (3) potential postprocessing step

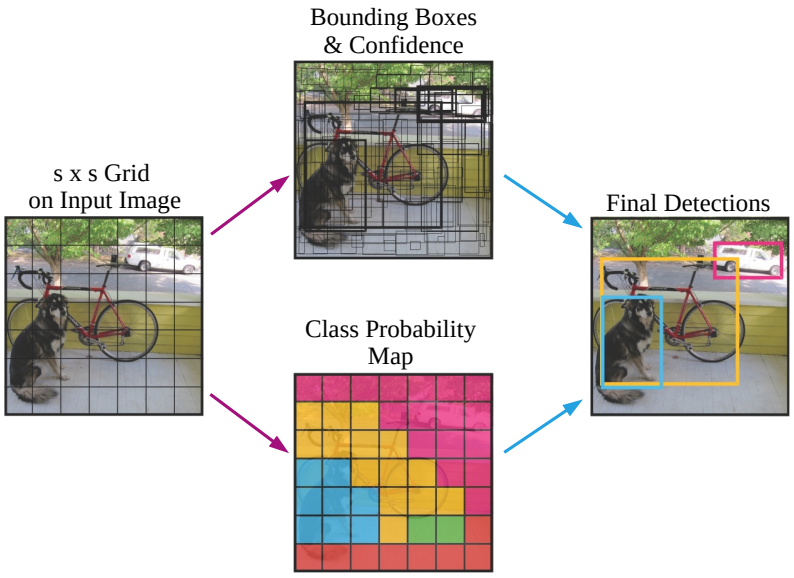


Figure 4.2: **YOLO Regression System:** The system for object detection divides the image into a $s \times s$ grid. For each grid cell the model predicts b bounding boxes, b confidence values and c class probabilities [RDGF16].

different foreground-background segmentation algorithms to segment the object within each bounding box (see section 4.1.2). Thereby, we describe the functionality of each segmentation algorithm and our modification that the algorithm works in our three-task structure. The last task is a postprocessing step to refine the segmentation masks of the second task (see section 4.1.3). Therein, all foreground-background segmentation masks are merged and, in addition, we use a DenseCRF [KK11].

4.1.1 Object Detection

In this section, we present YOLO, a neural network for object detection proposed by Redmon et al. [RDGF16] in 2016. It achieves not only high detection accuracy but also fast inference (45 frames per second [RDGF16]). Moreover, Redmon et al. proposed a less accurate model, called Fast YOLO, that processes images at 155 frames per second [RDGF16]. For our segmentation baselines we use the more accurate model.

Detection

YOLO unifies all necessary steps of object detection into a single end-to-end trained neural network. Thereby, one crucial step is dividing the image into a $s \times s$ grid (see left image in figure 4.2). Each grid cell is only responsible for detecting objects whose centers fall into it. Every cell in this grid predicts b bounding boxes and b corresponding confidence scores. First, a confidence score represents how confident the model is that the box contains an object. This is modeled by a probability $P(Object)$. Second, it represents how accurate the predicted box is according to the GT bounding box. This can be measured with the IoU between the predicted box and the GT bounding box (IoU_{truth}^{pred}). Thus, the confidence

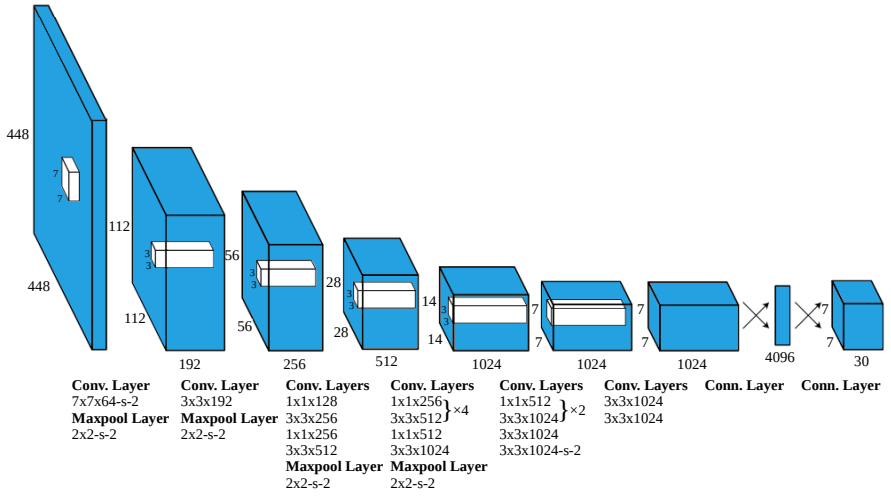


Figure 4.3: **YOLO Network Architecture:** The architecture consists of 24 convolutional layers followed by 2 fully-connected layers. The feature spaces of previous layers are reduced by using 1×1 convolutional layers.

score of a specific bounding box B is defined as:

$$Conf(B) = P(Object) \cdot IoU_{truth}^{pred} \tag{4.1}$$

In the top middle image in figure 4.2 we show an example of several bounding box predictions. The thickness of the bounding box lines represent the confidence score. Thereby, thick lines correspond to a high confidence score. Additionally, each grid cell predicts c conditional class probabilities $P(Class_i|Object)$, one for each object class. Therefore, the conditional class probability models how likely this cell contains an object with a specific class. In the bottom middle image in figure 4.2 we visualize a class probability map C. C corresponds to a per pixel maximum of all class probabilities. In the figure we visualize the class with the maximum predicted probability.

In summary, the proposed network learns $(5 \cdot b + c)$ values per grid cell using a regression model[1]. Therefore, the output of the network is encoded as a $s \times s \times (5 \cdot b + c)$ tensor. In case of PascalVOC: $c = 20$. Redmon et al. set the remaining parameters to: $s = 7$, $b = 2$. This results in a $7 \times 7 \times 30$ tensor.

Network Architecture and Training

To achieve fast inference, Redmon et al. propose a relative shallow neural network, corresponding to less parameters [RDGF16]. This network (see figure 4.3) consists of 24 convolutional layers followed by two fully-connected layers. Thereby, 9 convolutional layers of all 24 layers has a filter size of 1×1. These layers are used to reduce the feature map size and, therefore, the number of parameters. All convolutional layers are collectively responsible for extracting features from the image and the fully-connected layers then predict the output probabilities and the bounding box coordinates.

[1] A bounding box is represented by 4 values.

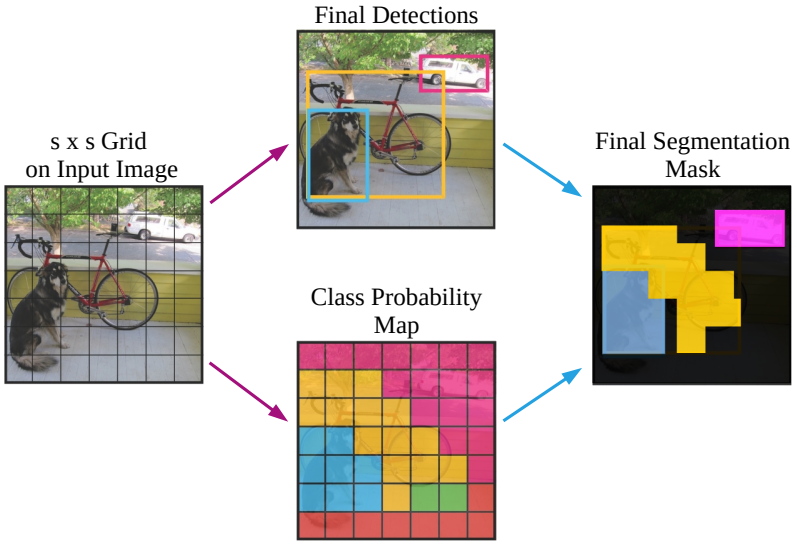


Figure 4.4: **YOLO based Segmentation:** A coarse segmentation mask is estimated by combining the resulting bounding boxes and the class probability map (7×7).

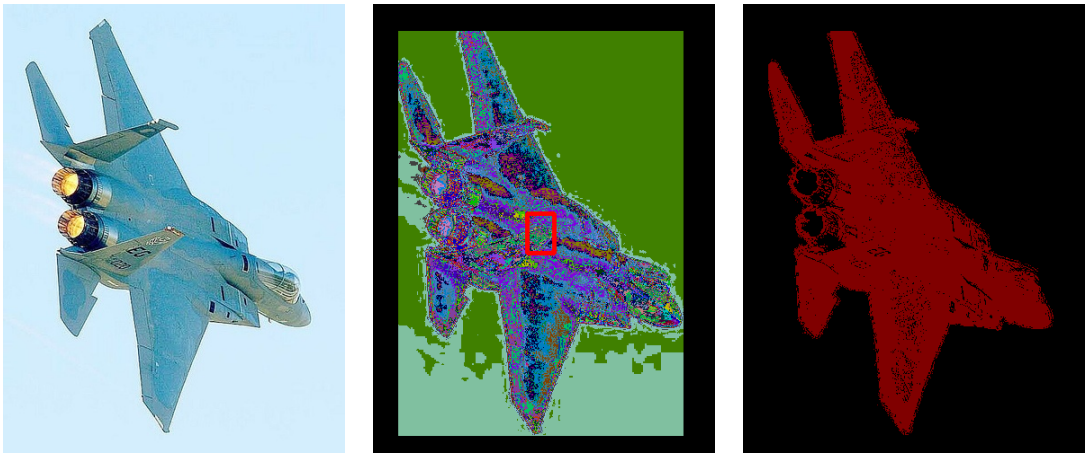
The network is pretrained to find expedient weights and is refined afterwards. The first 20 layers followed by an average-pooling layer and a fully-connected layer are pretrained on the ImageNet 1000-class competition dataset [DDS+09]. Then, both the missing convolutional layers and the missing fully-connected layers are added to the 20 pretrained layers . The added convolutional and fully-connected layers are initialized randomly. After initialization, the neural network is trained for the object detection task. During training Redmon et al. optimize a multi-part loss function. Each part is based on a sum-squared error and penalizes either errors in classification prediction or bounding box coordinates prediction. Moreover, this loss function only penalizes classification error if an object is present in that grid cell. It also penalizes bounding box coordinate error if the predictor is "responsible" for the GT box.

4.1.2 Foreground-Background Segmentation

In this section we present seven different segmentation methods. In case of the unsupervised approaches, we adapt them to use the predicted bounding boxes that we obtained from the YOLO model (see section 4.1.1) as additional segmentation cue.

4.1.2.1 YOLO

Although YOLO is a neural network for object detection, we adapt it to output segmentation masks. We interpret the class probability map C (see figure 4.4) as a coarse segmentation (7×7). Since segmentation is a pixel-wise assignment of labels, we rescale the class probability map to the resolution of the input image. This results in quadratic superpixels. YOLO models for each grid cell only the probability for all object classes and does not model the background. Therefore, we combine all object bounding boxes $\mathcal{B}$ with the rescaled class probability map and additionally threshold the probability. Every pixels



(a) Input data

(b) K clusters visualized by different gray values. Pixels in the center of the bounding box are marked by a red box.

(c) Final segmentation

Figure 4.5: **K-Means Mid:** Segment an image by assuming pixels in the center of the bounding box and their corresponding clusters as foreground.

that fall in a bounding box and in addition has the same class (exceeding the threshold) as the bounding box is set to this class. All remaining pixels are set to background. An example of the final mask is illustrated in figure 4.4.

4.1.2.2 K-Means

Foreground-background segmentation can be seen as a clustering problem, since we want to group foreground pixels and background pixels. In case of the RGB color space, each pixel $\mathbf{p}_i$ of an image I is represented by three values: red, green and blue. We use K-Means to cluster pixels with similar color. However, pixels with the same color do not necessarily belong to the same class. Therefore, we made further assumptions to achieve foreground-background segmentation within a bounding box and, in the following, we present two baseline segmentation algorithms based on these assumptions.

Object Location in the Middle of the Bounding Box

Our first assumption follows the idea of Papandreou et al., which assume that pixels in the middle of the bounding box are more likely foreground [PCMY15]. This assumption is based on characteristics of real world objects. Most objects that we want to detect are *"compact"*, i.e. do not contain holes. Counterexamples are mostly fine structured objects like a ladder or a bicycle. In the PascalVOC dataset almost all objects, except of the bicycle, have parts that exhibit this kind of compactness. Following this assumption, segmentation corresponds to finding these *"compact"* regions within the bounding box. Therefore, not the whole image I is used, but a subimage I_B defined by the bounding box. An exemplary subimage is illustrated in figure 4.5(a). We assume all pixels in box B_α, located in the middle of the subimage, as foreground. α defines the size of this box and correlates with the size of the subimage. More precisely, $\alpha = 0$ corresponds to the pixel in

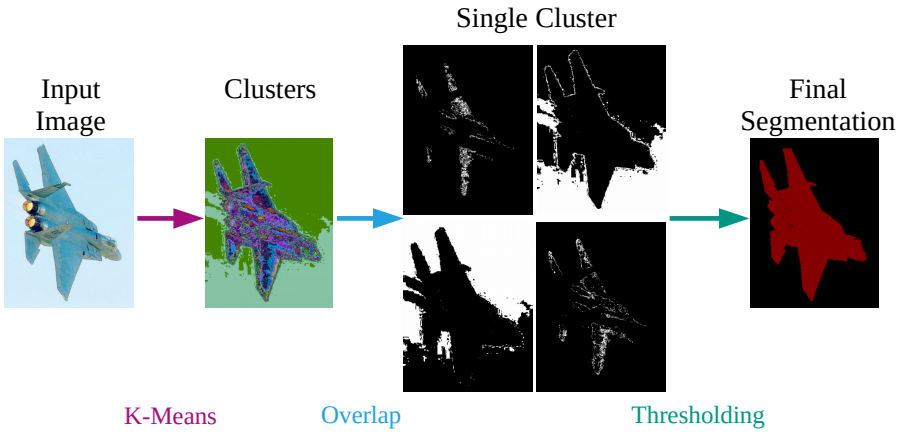


Figure 4.6: **K-Means Overlap**: First, K-Means is applied resulting in K different clusters, i.e. K segmentation proposals. For each proposal the overlap with the bounding box is computed. The overlap is thresholded, i.e. proposals with high overlap are set to foreground.

the middle, while $\alpha = 1$ corresponds to the whole subimage. But this assumption is not satisfied by all objects. Objects with no *"compact"* part in the middle (e.g. a floor lamp) do not satisfy this assumption. Nevertheless, many other objects (e.g. monitor, car, bus or plane) usually satisfy this assumption. If pixels are similar to pixels in the middle box B_α, then we assume these pixels also as foreground. Therefore, we use K-Means to cluster similar pixels. It turned out empirically that setting $K = 90$ and $\alpha = 0.1$ is appropriate for segmentation. Figure 4.5(b) visualizes all clusters and the middle box for an exemplary image. Each cluster is represented by a specific gray value. Figure 4.5(c) illustrates the resulting foreground-background segmentation.

Overlap Between a Cluster and a Bounding Box

Our second assumption uses the overlap between clusters and the bounding box and, therefore, considers not only pixels within the bounding box (see figure 4.6). We use the knowledge that pixels outside are background. Since pixels that are similar to background pixels are more likely to be background, we use K-Means to cluster pixels. As before, we use $K = 90$. Therefore, the overlap is an indication for the foreground probability, since large overlap corresponds to probable foreground. If otherwise a cluster contains many pixels outside the bounding box, i.e. the overlap is small, this cluster is more likely to be background. The overlap between a bounding box B and a cluster C is defined as

$$O(C, B) = \frac{|C \cap B|}{|C|} \tag{4.2}$$

To decide if a cluster is foreground or background we use a threshold θ. If the overlap is greater than the threshold, all pixels of this cluster within the bounding box are foreground, otherwise all pixels are background. We compare two different thresholds: a fixed threshold and an adaptive threshold. For the fixed threshold we empirically choose $\theta = 0.4$. We define the adaptive threshold based on the bounding box size. The absolute number of pixels within a bounding box depends on the bounding box size. The overlap decreases

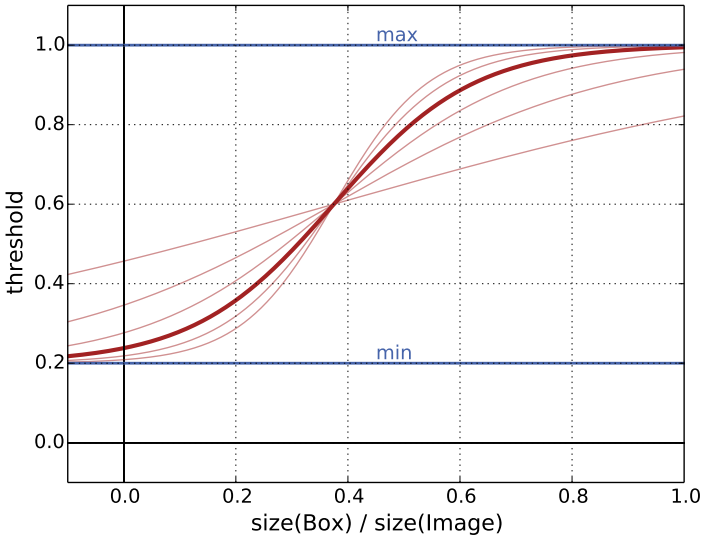


Figure 4.7: **Adaptive Threshold:** Example for the adaptive threshold with different temperature T ($4 \leq T \leq 12$, $min = 0.2$, $max = 0.8$, $b = 0.375$)

if pixels are outside. The problem is that the number of pixels outside the bounding box has more impact to the overlap in case of small objects. Therefore, we want a smaller threshold for small objects and a high threshold for large objects. We define $H(I)$ as the height and $W(I)$ as the width of a given image I. We use the size of a subimage I_B defined by a bounding box compared to the size of the whole image I as input P for our adaptive threshold. Thus, we define our adaptive threshold as a function on condition that $0 < \theta < 1$:

$$\begin{aligned} P &= \frac{H(I_B) \cdot W(I_B)}{H(I) \cdot W(I)} \\ \theta &= (max - min) \cdot \frac{1}{1 + e^{-T \cdot (P - b_{shift})}} + min \end{aligned} \tag{4.3}$$

θ is based on the sigmoid function and has parameters to adjust it. The parameters min and max restrict the range of θ. min corresponds to the minimal possible threshold and max to the maximal possible threshold. The parameter T corresponds to the slope of the sigmoid function and is called temperature. The parameter b_{shift} shifts the function along the x-axis (see figure 4.7)

4.1.2.3 GrabCut

GrabCut is a weakly-supervised algorithm for segmentation proposed by Rother et al. in 2004 [RKB04] and is based on the work of Boykov et al. [BJ01]. The main idea of GrabCut is to transform an image into a graph and achieve foreground-background segmentation by applying a minimal cut to this graph.

For a given image I consisting of pixels $\mathbf{p}_i \in I$, the GrabCut algorithm additionally requires a bounding box as input. Every pixel outside the bounding box is background and is assigned to a set $\mathcal{T}_B$. Pixels within the bounding box are treated as unknown and

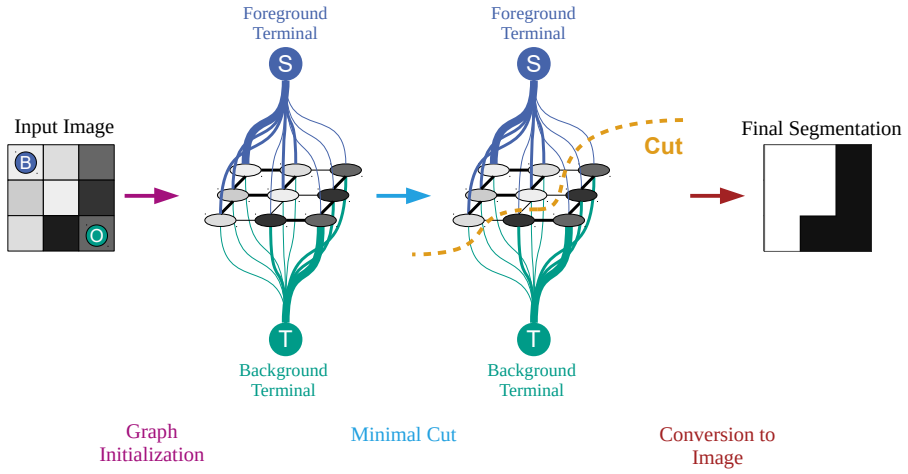


Figure 4.8: **GrabCut**: Segmentation of a simple 3×3 image using GrabCut by converting the image into a graph and applying a minimal cut. The foreground and background model is learned using the annotated pixels B (foreground) and O (background). The thickness of edges corresponds to the similarity of pixels or the similarity to the foreground or background model.

assigned to a set $\mathcal{T}_U$. The aim is then to assign the unknown pixels to the background set $\mathcal{T}_B$ or the foreground set $\mathcal{T}_F$. Gaussian Mixture Models (GMMs) are used to model both the foreground and the background color distribution.

This is achieved by minimizing the energy E, consisting of a data term U and a smoothness term V. Boykov et al. minimizes E by converting the image into a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ and applying a minimal cut in this graph [BJ01]. An exemplary graph is visualized in figure 4.8.

Every pixel p_i is treated as a node $v_i \in \mathcal{V}$ in the graph $\mathcal{G}$. Every node is connected to nodes of neighboring pixels by an edge. A weight w_e is assigned to every edge $e \in \mathcal{E}$. This weight corresponds to pixel similarity and is computed by the Euclidean distance. Moreover, two additional nodes are added to the graph: the source node S as *foreground/object* terminal and the sink node T as the *background* terminal. Every node is both connected to the source and the sink node. The weights between a (pixel) node v_i and the source node S or the sink node T depict how likely the pixel p_i is foreground or background. This probability is modeled by the previously mentioned GMMs. The weight $w_e > 0$ applies for every edge e because they model a probability. A minimal cut separates the sink node and the source node and is defined by

$$Cut = \arg\min_{\overline{Cut} \subset \mathcal{E}} |\overline{Cut}| = \arg\min_{\overline{Cut} \subset \mathcal{E}} \sum_{e \in \overline{Cut}} w_e \tag{4.4}$$

Moreover, this cut Cut minimizes the energy E. Applying Cut leads to two subsets of $\mathcal{E}$: the one connected to S and the one connected to T. These two subsets define a foreground-background segmentation, i.e. the assignment of pixels to $\mathcal{T}_F$ and $\mathcal{T}_B$.

Rother et al. extended this idea by an iterative approach [RKB04]. In every iteration the result is improved by reestimating the foreground and background model. For example, segmentation achieved by the first minimal cut leads to more knowledge of what is

Algorithm 3 GrabCut

Require: bounding box $\rightarrow \mathcal{T}_U$ and $\mathcal{T}_B$
  Initialize GMMs using bounding box
  repeat
    Assign GMM components to pixels # minimize V
    Learn GMM parameters # minimize U
    Estimate segmentation (minimal cut): $Cut = \arg\min_{\overline{Cut} \subset \mathcal{E}} |\overline{Cut}|$ # minimize E
  until convergence
  return segmentation defined by $Cut \rightarrow \mathcal{T}_F$

foreground and background. This information can be used to reestimate the GMM parameters. Then, the reestimated foreground and background models, in turn, can improve the segmentation in the next iteration. Algorithm 3 specifies the whole iterative GrabCut procedure. Thereby, assigning GMM components to pixels corresponds to minimizing the smoothness term V and learning GMM parameters corresponds to minimizing the data term U. Furthermore, the minimal cut minimizes the whole energy E.

4.1.2.4 Minimum Barrier Detection

Minimum Barrier Salient Object Detection (MBOD) is a fast unsupervised algorithm for salient object detection, proposed by Zhang et al. in 2015 [ZSL+15]. Zhang et al. made two assumptions for object detection:

- salient objects, i.e. large difference between object and background
- objects are located in the middle of the image

The second assumption holds for the most photos taken in everyday situations. When taking a picture, the interesting object is normally located in the middle. However, this assumption gets problematic if there are several interesting objects in the image, e.g. a traffic situation with several cars. Therefore, we use the bounding box as further information.

MBOD computes a background model by considering the boundary regions. Therefore, we use not the subimage I_B defined by a bounding box, but an enlarged subimage $\hat{I}_B$ with $|I_B| < |\hat{I}_B| < |I|$. Thus, we force the algorithm to learn the background model by using pixels outside the bounding box, i.e. using assured background pixels. Zhang et al. assume the object in the middle of the image (in our case in the middle of the enlarged subimage $\hat{I}_B$) and, thus, pixels in the middle are more likely foreground than pixels near the boundary. Therefore, Zhang et al. present a very fast, approximate Minimum Barrier Distance (MBD) transform algorithm, that computes the distance of a pixel to the image boundary. This distance directly correlates with the foreground probability of a pixel. If the distance is high, the pixel is more likely foreground. If the distance is low, the pixel is more likely background.

Distance Transform

The MBD transform algorithm determines a path from a pixel in the image $\hat{I}_B$ to the boundary. A path $\pi = \langle \pi(0), ..., \pi(k) \rangle$ is a sequence of adjacent pixels and has a specific

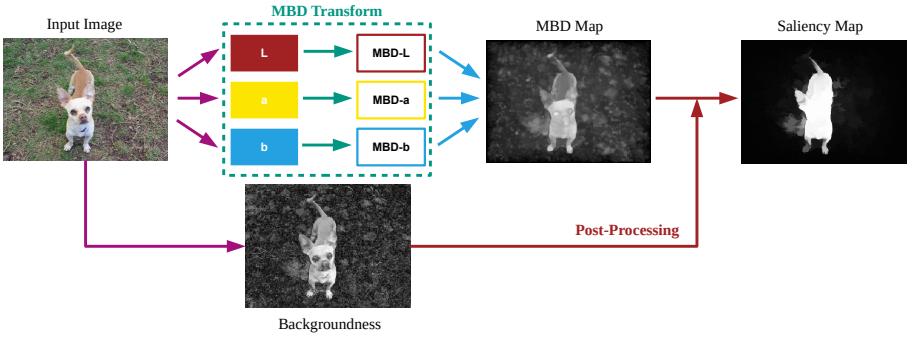


Figure 4.9: **MBOD Algorithm:** First, apply MBD transform to every color channel of the input image. The resulting MBD Map is combined with a backgroundness measure to receive the final saliency map.

cost, defined by a cost function. Zhang et al. uses MBD as cost function, a distance proposed by Strand et al. in 2013 [SCMS13]:

$$\beta_{\hat{I}_B}(\pi) = \max_{i=0}^{k} \hat{I}_B(\pi(i)) - \min_{i=0}^{k} \hat{I}_B(\pi(i)) \tag{4.5}$$

Computing the minimal path distance from each pixel to the boundary leads to a distance map D. Therefore, D can be computed for a specific pixel $\mathbf{p}_i$ as follows:

$$D(p_i) = \min_{\pi \in \Pi(\mathbf{p}_i)} \beta_{\hat{I}_B}(\pi) = \min_{\pi \in \Pi(\mathbf{p}_i)} \left(\max_{i=0}^{k} \hat{I}_B(\pi(i)) - \min_{i=0}^{k} \hat{I}_B(\pi(i)) \right) \tag{4.6}$$

where $\Pi(\mathbf{p}_i)$ are all possible paths to the image boundary. However, an exact algorithm for MBD transform is time consuming. Therefore, Zhang et al. present an approximate iterative algorithm, based on raster scanning techniques.

For the final segmentation of the object, the image $\hat{I}_B$ is converted into LAB color space. After that, MBD transform is applied to every color channel (see figure 4.9). Afterwards, the MBD transform results are added and, thus, leads to a final MBD map. If the object is salient, i.e. the first assumption holds, then the color difference between the object and the background is high. Since the cost of a path from any object pixel to the boundary is based on color difference, the cost should be also high. However, the cost of a path from background pixels to the boundary can also be high. In case of a satisfied assumption (existence of salient objects), high path cost for background pixels is less probable.

Background Model

The MBD map is improved by an additional backgroundness cue. Zhang et al. compute a further distance map U, called Image Boundary Contrast (IBC) map. Each pixel of this map is equivalent to the distance of this pixel in the original image to the background model. The background model is determined by four image boundary regions: lower, upper, left and right (see figure 4.10). According to the second assumption, regions near the boundary are more likely background. For each boundary region the mean color μ_k and the covariance $\mathbf{Q}_k$ are computed and, therefore, the background is modeled explicitly. For each boundary region Zhang et al. compute an individual IBC map U_k using the

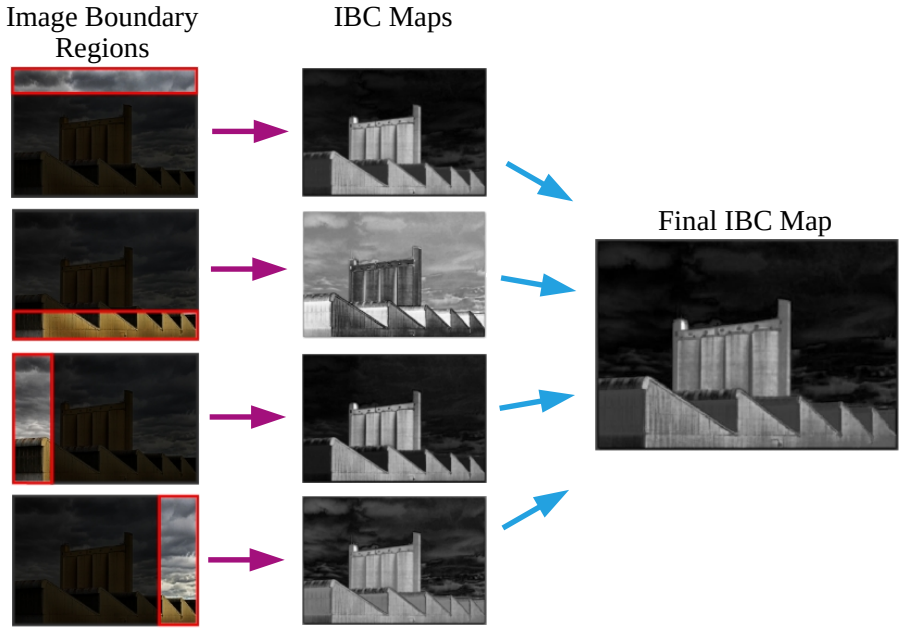


Figure 4.10: **IBC Map:** Image boundary regions and the resulting IBC map.

Mahalanobis distance. In contrast to the Euclidean distance the Mahalanobis distance uses the covariance matrix. If the covariance matrix is the identity matrix the Mahalanobis distance degenerates to the Euclidean distance. The Mahalanobis distance of a pixel $\mathbf{p}_i$ to the background model is defined by:

$$U_k(\mathbf{p}_i) = \sqrt{(\mathbf{p}_i - \mu_k)\ \mathbf{Q}_k^{-1}\ (\mathbf{p}_i - \mu_k)^\top} \tag{4.7}$$

U_k is normalized so that $U_k(p_i)$ lies in $[0, 1]$:

$$U_k(\mathbf{p}_i) = \frac{U_k(\mathbf{p}_i)}{\max_j U_k(\mathbf{p}_j)} \tag{4.8}$$

The final IBC map U is a modified sum of all individual IBC maps U_k, that is more robust in case the objects lies in one boundary region:

$$U(\mathbf{p}_i) = \left(\sum_{k=1}^{4} U_k(\mathbf{p}_i)\right) - \max_k\ U_k(\mathbf{p}_i) \tag{4.9}$$

This IBC map is also scaled such that all values lie in $[0, 1]$. The final step, also visualized in figure 4.9, is the combination of the previous MBD map and the IBC map by a simple addition.

4.1.2.5 Robust Background Detection

Robust Background Detection (RBD) is an unsupervised algorithm, proposed by Zhu et al. in 2014 [ZLWS14] for object detection. Similar to MBOD [ZSL+15], Zhu et al. assume that objects are located in the middle of an image and, therefore, only slightly touch the

image boundary. While Zhang et al. compute a distance from each pixel to the image boundary [ZSL+15], Zhu et al. propose a robust background measure based on superpixels and their boundary connectivity. A superpixel is a group of nearby, similar pixels caused by oversegmentation of an image (see left images in figure 4.11). Since we detect an object within its bounding box the object inevitably touches the image boundary. Therefore, we use an enlarged subimage $\hat{I}_B$ defined by the bounding box B (similar to section 4.1.2.4).

Boundary Connectivity

The proposed measure, called boundary connectivity, determines how heavy a region $\mathcal{R}$ is connected to the image boundary.

$$BndCon(R) = \frac{|\{\mathbf{p}|\mathbf{p} \in \mathcal{R}, \mathbf{p} \in \mathcal{B}\}|}{\sqrt{|\{\mathbf{p}|\mathbf{p} \in \mathcal{R}\}|}} \tag{4.10}$$

where $\mathcal{B}$ is the set of image boundary patches and $\mathbf{p}$ is an image patch. $BndCon(\cdot)$ determines the percentage of boundary patches compared to all patches of a region. The square root is used to achieve scale-invariance. However, this definition is hard to compute. Therefore, an alternative is used based on superpixels. Zhu et al. compute 200 superpixels by Simple Linear Iterative Clustering (SLIC) [ASS+12], an algorithm proposed by Achanta et al. in 2012 (see figure 4.11). Afterwards, they construct an undirected weighted graph by connecting all adjacent superpixels $(\mathbf{p}, \mathbf{q})$. The corresponding weight $d_{app}(\mathbf{p}, \mathbf{q})$ is calculated by the Euclidean distance between their average colors in the CIE-Lab color space. Using the geodesic distance[2] between any two superpixels leads to a definition of the "spanning area" for each superpixel $\mathbf{p}$:

$$Area(\mathbf{p}) = \sum_{i=1}^{N} S(\mathbf{p}, \mathbf{p}_i) \tag{4.11}$$

$S(\mathbf{p}, \mathbf{p}_i)$ shows how much a superpixel $\mathbf{p}_i$ contributes to the area $\mathbf{p}$. Similarly, the *"length along the boundary"* is defined by:

$$Len_{bnd}(p) = \sum_{i=1}^{N} S(\mathbf{p}, \mathbf{p}_i) \cdot \delta(\mathbf{p}_i) \tag{4.12}$$

where $\delta(\cdot)$ is 1 if the superpixel belongs to the boundary, 0 otherwise. Summarizing, the boundary connectivity is defined similar to equation 4.10 by:

$$BndCon(p) = \frac{Len_{bnd}(\mathbf{p})}{\sqrt{Area(\mathbf{p})}} \tag{4.13}$$

To compute equation 4.13, the shortest paths between all superpixel pairs are efficiently calculated using Johnson's algorithm [Joh77].

Saliency Map Pipeline

In figure 4.11 we show the four step pipeline for salient object detection. The first step is computing the contrast for each superpixel $\mathbf{p}$ against surrounding superpixels using the

[2] Zhu et al. define the geodesic distance as the accumulated edge weights along their shortest path on the graph [ZLWS14].

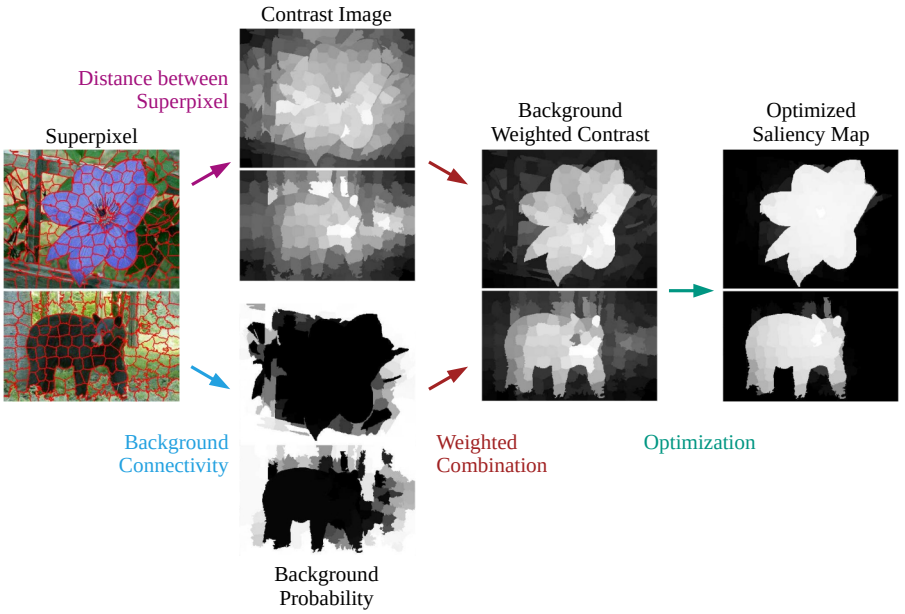


Figure 4.11: **Saliency Map Pipeline:** A contrast map and a background probability weight is computed using superpixels. Both are combined to a background weighted contrast map. This background weighted contrast map is optimized afterwards.

distance $d_{app}(\mathbf{p}, \mathbf{q})$ (see figure 4.11). Then, the superpixel's contrast Ctr is defined by:

$$Ctr(p) = \sum_{i}^{N} d_{app}(\mathbf{p}, \mathbf{p}_i)\ w_{spa}(\mathbf{p}, \mathbf{p}_i) \tag{4.14}$$

where $w_{spa}(\mathbf{p}, \mathbf{p}_i) = \exp(-\frac{d^2_{spa}(\mathbf{p},\mathbf{p}_i)}{2\sigma^2_{spa}})$. The distance between the centers of superpixel $\mathbf{p}$ and $\mathbf{p}_i$ is defined as $d_{spa}(\mathbf{p}, \mathbf{p}_i)$. Zhu et al. set $\sigma_{spa} = 0.25$ The second step is independent of the first step and computes a background probability w_i^{bg} per superpixel. This probability correlates with the boundary connectivity, i.e. the larger the connectivity to the image boundary the larger the probability w_i^{bg}. The background probability is defined by:

$$w_i^{bg} = 1 - \exp\left(-\frac{BndCon^2(\mathbf{p}_i)}{2\sigma^2_{bndCon}}\right) \tag{4.15}$$

Zhu et al. empirically set $\sigma_{bndCon} = 1$ [ZLWS14]. Afterwards, the contrast map Ctr is weighted with the background probability w_i^{bg}. This enhanced contrast map $wCtr$ is defined by:

$$wCtr(p) = \sum_{i}^{N} d_{app}(\mathbf{p}, \mathbf{p}_i)\ w_{spa}(\mathbf{p}, \mathbf{p}_i)\ w_i^{bg} \tag{4.16}$$

According to equation 4.16 and the weight w_i^{bg} the contrast increases for object regions and decreases for background regions and, therefore, the whole contrast difference increases between object and background.

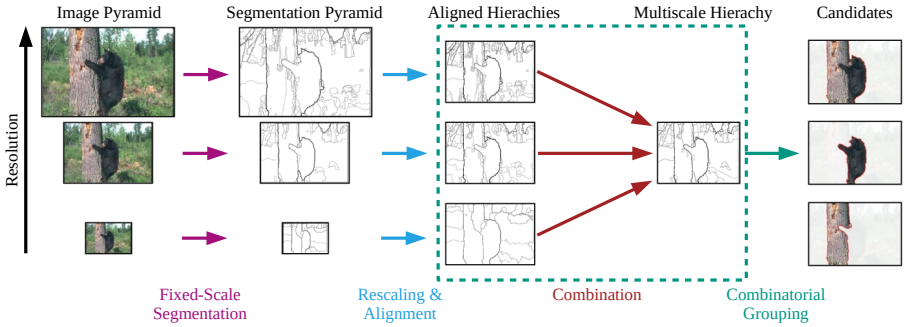


Figure 4.12: **Object Candidates Generation:** Applying a fixed-scale segmentation on every image of a multi-resolution pyramid. The resulting contour maps are rescaled and aligned. Afterwards, they are combined to a multi-scale hierarchy. This hierarchy is used to generate ranked object candidates.

Nevertheless, the background weighted contrast map is still noisy. Therefore, this map is smoothed by optimizing a cost function. Zhu et al. define the cost function C consisting of three different constraints: background constraint, foreground constraint, smoothness constraint. Minimizing C results in the final optimized saliency map. Consider s_i as the saliency map value of a corresponding superpixel $\mathbf{p}_i$. Intuitively, the background constraint encourages saliency map values of superpixels with high background probability to be low (close to 0). In contrast to the background probability, the foreground constraint encourages values to be high (close to 1) with high foreground probability. The smoothness constraint encourages continuous saliency values. Finally, C is defined by:

$$C = \underbrace{\sum_{i=1}^{N} w_i^{bg} s_i^2}_{\text{background}} + \underbrace{\sum_{i=1}^{N} w_i^{fg} (s_i - 1)^2}_{\text{foreground}} + \underbrace{\sum_{i,j} w_{i,j} (s_i - s_j)^2}_{\text{smoothness}} \tag{4.17}$$

All three terms are squared errors and, therefore, the optimal and final saliency map can be easily computed with a least-square algorithm.

4.1.2.6 Multiscale Combinatorial Grouping

Multiscale Combinatorial Grouping (MCG) is an algorithm proposed by Arbeláez et al. that generates ranked object candidates [APTB+14]. These object candidates can be used for segmentation within a bounding box. For generating object candidates a segmentation hierarchy $\mathcal{S} = \{\mathcal{S}^0, ..., \mathcal{S}^L\}$ is used. While $\mathcal{S}^0$ is the finest set of superpixels, $\mathcal{S}^L$ is the complete image. A specific segmentation $\mathcal{S}^i$ consists of different regions $\mathcal{S}^i = \{\mathcal{S}_0^i, ..., \mathcal{S}_M^i\}$. Moreover, every region from coarse levels are unions of regions from fine levels. For example, a segmentation $\mathcal{S}^{i+1}$ results from merging regions in segmentation $\mathcal{S}^i$. Such a hierarchy can be represented as an Ultrametric Contour Map (UCM) by assigning each level $\mathcal{S}^i$ a real-valued index λ_i. Weighting the boundary of each adjacent region pair by the index at which they are merged result into an UCM. Since a threshold at level λ_i in the UCM produces the segmentation $\mathcal{S}^i$, this representation unifies the problem of contour detection and hierarchical image segmentation. The UCM is then used to generate ranked object candidates. The whole system is illustrated in figure 4.12 and, in the following, we describe it in detail.

Fixed-scale Segmentation

The original input image is subsampled/supersampled to generate a multi-resolution pyramid with N scales. Afterwards, on every image in the pyramid a proposed single-scale segmentation is applied, leading to an UCM per image. In the single-scale segmentation Arbeláez et al. consider the following local contour cues:

1. brightness, color, texture, differences in half-disks of three sizes [MFM04]
2. sparse coding on patches [XB12]
3. structured forest contours [DZ13]

The contour cues are segmented independently treating segmentation as a graph partitioning problem and, thereby, using the normalized cuts criterion [SM00]. Arbeláez et al. proposed an efficient normalized cuts algorithm for contour detection based on Eigenvalue computation. After segmentation that uses the normalized cuts algorithm, global and local cues are combined linearly and construct an UCM based on the mean contour strength.

Hierarchy Alignment

Pixel errors in an UCM can have drastic effect on the final object proposals, since the topology and the strength of an UCM determine the underlying segmentation hierarchy. Thus, rescaling an UCM is nontrivial. In order to preserve thin structures and details, Arbeláez et al. use a rescaling and alignment step. They extract the finest superpixels in each hierarchy, rescale them to the original image resolution and declare them as possible boundary locations. Then, each UCM is transfered recursively using the rescaled sets of finest superpixels.

Multiscale Hierarchy

The alignment generates for each of the N scales a fixed set of boundary locations. Arbeláez et al. formulate this as a binary boundary classification problem and train a classifier that combines these N features into a single probability of boundary estimation using Platt's method [P$^+$99].

Combinatorial Grouping of Candidates

Arbeláez et al. consider singletons, pairs, triplets, and 4-tuples of regions from three individual scales and the multiscale hierarchy, resulting in 16 lists of candidates. First, these candidates are reduced by optimizing a learning problem with two conflicting objective functions: number of candidates and achievable quality. This step reduces the amount of candidates from millions to thousands. Second, Arbeláez et al. train a Random Forest using low-level features (size, location, shape and contours) to further reduce the number of candidates.

Box-wise Segmentation

We use the 500 best object proposals provided by MCG, to generate a foreground-background segmentation within a bounding box (see figure 4.13). Therefore, we compute the overlap between each object proposal and a specific bounding box. The higher the overlap with

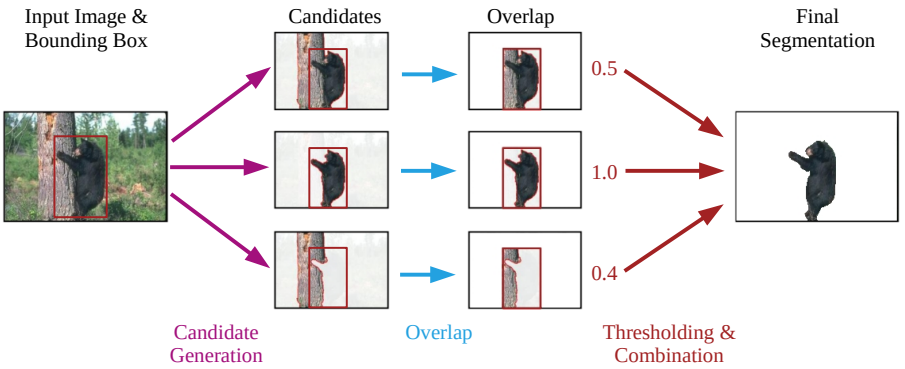


Figure 4.13: **Segmentation:** Segmentation using bounding boxes and MCG.

the bounding box, the more likely the object proposal is foreground. The overlap is defined equally as equation 4.2. Since objects normally are not single-colored, an object can consist of more than one object proposal. Therefore, we set all proposals to foreground if their corresponding overlap is higher than a defined threshold. Object proposals are not mandatorily distinct, i.e. several small proposals can cover the area of one large proposal. Moreover, all 500 proposals together cover the whole image. Thus, using small thresholds provides the whole bounding box as a foreground segmentation. A threshold near 1.0 empirically turned out as an appropriate threshold. Therefore, we set our threshold to 0.99.

4.1.3 Postprocessing

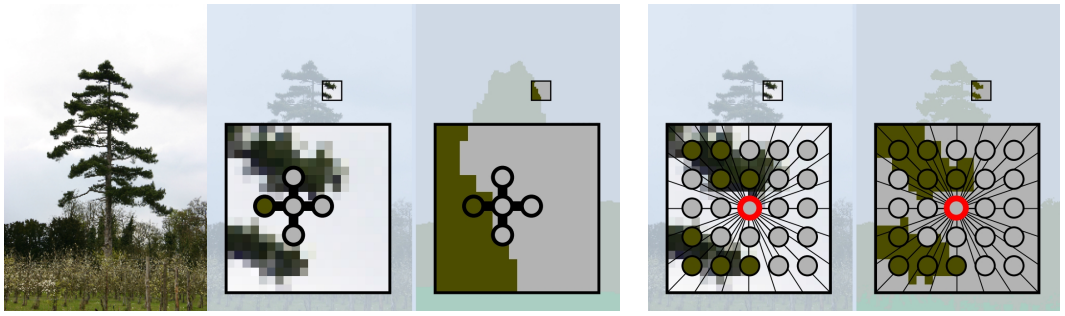
In this section we present our postprocessing step, that we apply after the foreground-background segmentation. Bounding boxes can overlap and, thus, the resulting segmentations can overlap as well. Thereby, we have to decide which pixel is treated as foreground. We follow Khoreva et al. [KBH+17] and Papandreou et al. [PCMY15] and resolve ambiguities by selecting the segmentation with smaller area. Thus, smaller segmentations can not disappear more infrequently in the final segmentation mask.

Dense Conditional Random Fields

Moreover, we present a common postprocessing algorithm proposed by Krähenbühl et al.: Dense Conditional Random Fields (DenseCRFs) [KK11]. A CRF is a graphical model, where each pixel is a node in that model with dependencies to other nodes. CRFs include unary potentials on individual pixels and pairwise potentials on neighboring pixels. Since the resulting adjacency CRF structure is limited in its ability to model long-range connections (see figure 4.14), Krähenbühl et al. use a fully-connected CRF and propose an efficient approximate inference algorithm.

In our case a CRF consists of two random fields $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and $\mathcal{I} = \{\mathbf{i}_1, ..., \mathbf{i}_N\}$. $\mathcal{I}$ ranges over all possible input images of size N and $\mathcal{X}$ ranges over all possible pixel-level labels. In addition, $\mathbf{i}_j$ is the color and $\mathbf{x}_j$ the assigned label of pixel j. The energy of the CRF is defined as [KK11]:

$$E(x) = \sum_i \underbrace{\psi_u(\mathbf{x}_i)}_{\text{unary term}} + \sum_i \sum_{j>i} \underbrace{\psi_p(\mathbf{x}_i, \mathbf{x}_j)}_{\text{pairwise term}} \tag{4.18}$$



(a) Input Image (b) Connections in a CRF (c) Connections in a DenseCRF

Figure 4.14: **Connection Comparison:** Comparison of short-range connections of a CRF and long-range connections of a DenseCRF.

where i and j define a pixel and ranges from 1 to N. The unary potential $\psi_u(\mathbf{x}_i)$ is given by any segmentation algorithm (see section 4.1.2). The pairwise potential is responsible for a consistent labeling and is a linear combinations of Gaussians. The efficient inference in fully-connected CRFs is based on a mean field approximation. This approximation yields an iterative message passing algorithm for approximate inference.

4.2 End-to-End Segmentation Model

In this section, we present our end-to-end trained segmentation model built upon the work of Dai et al. [DHS16]. For this purpose, we use their provided code[3]. This code is a reimplementation in python using the Caffe framework [JSD+14] of their original Matlab code. We modify both their code and the training data, since we use bounding boxes as supervision for training. We use our best foreground-background segmentation algorithms presented in section 4.1.2 to weakly generate object segmentation masks within each GT bounding box. Our neural network learns these masks instead of manually labeled segmentation masks. First, we present the Multi-task Network Cascade (MNC) model in detail (see section 4.2.1). In section 4.2.2, we mention all training parameters (e.g. learning rate). Finally, we describe the modification in the network learning code to deal with bounding boxes as supervision in section 4.2.3.

4.2.1 Multi-task Network Cascades

In this section, we describe the structure of the neural network proposed by Dai et al. [DHS16] in 2016, since we modify this neural network, also called Multi-task Network Cascade (MNC), to utilize it for weakly-supervised instance segmentation. The proposed architecture is visualized in figure 4.15.

According to the "multi-task" in their network name, the authors decompose instance segmentation into multiple sub-task, similar to the structure of our baselines (see section 4.1). Therefore, Dai et al. divide their network into three stages, where each stage is responsible for one sub-task: [DHS16]:

[3] https://github.com/daijifeng001/MNC

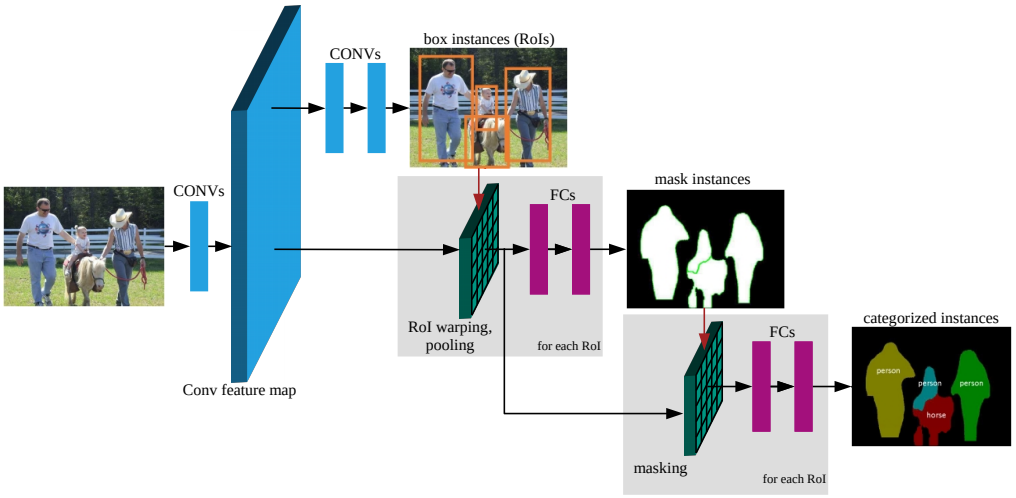


Figure 4.15: **MNC:** Multi-task Network Cascades for instance segmentation [DHS16] subdivided in three sub-task: bounding box regression, segmentation mask generation within each bounding box and categorizing each mask.

1. **Regressing Box-Level Instances:** The instances are represented by class-agnostic bounding boxes.
2. **Regressing Mask-Level Instances:** For each instance a pixel-level mask is predicted.
3. **Categorizing Instances:** For each mask-level instance the category-wise label is predicted.

Each stage depends on the output of earlier stages, e.g. the second stage requires the bounding boxes of the first stage for predicting the pixel-level mask. The authors designed their stages to share convolutional features. These convolutional features are obtained by the VGG model [NHH15], more precisely, VGGs first 13 convolutional layers. Moreover, each stage involves a loss term L. Since later stages depend on earlier stages, their loss terms also rely on earlier losses. Thus, the loss terms are not independent. Figure 4.16 illustrates a simplification of the architecture with the loss terms L_1, L_2 and L_3. In the following we describe each stage in detail.

Regressing Box-level Instances

In the first stage, the network proposes object instances in form of bounding boxes using the shared convolutional features [DHS16]. The proposed bounding boxes are class-aware. Moreover, the network predicts an objectness score for each bounding box. Therefore, the authors follow the work of Ren et al. [RHGS15] and use a Region Proposal Network (RPN). An RPN outputs a set of rectangular object proposals to a given input image, each proposal with an objectness score. According to Ren et al. [RHGS15], the authors model this process with a fully-convolutional network. For proposal generation a small network is slided over the shared convolutional features [RHGS15] (see figure 4.17). A 3×3 convolutional layer is used for reducing dimensions of each sliding window. This layer is followed by two sibling 1×1 convolutional layers. One for regressing box locations and the other for classifying

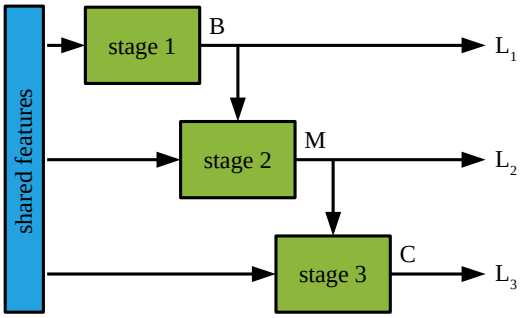


Figure 4.16: **Simplified MNC:** Simplified illustration of the three staged architecture [DHS16].

object/non-object. For regression the box location, "anchors" are used for each location. "Anchors" are predefined boxes with different rectangular form (see figure 4.17). Dai et al. [DHS16] use the RPN loss from Ren et al. [RHGS15]. With Θ defining all network parameters, the loss term L_1 of the first stage is defined as [DHS16, RHGS15]:

$$\begin{aligned} L_1 &= L_1(\mathcal{B}(\Theta)) \\ &= \frac{1}{N_{cls}} \cdot \sum_i L_{cls}(p_i, p_i^*) + \lambda \cdot \frac{1}{N_{reg}} \cdot \sum_i p_i^* \cdot L_{reg}(\mathbf{t}_i, \mathbf{t}_i^*) \end{aligned} \tag{4.19}$$

where $\mathcal{B}$ is the output of this stage, representing a list of boxes including the location and their objectness probability. Furthermore, i is the index of an anchor in a mini-batch and p_i represent the probability of anchor i being an object. The GT label p_i^* is determined depending on the IoU of the anchor and the GT bounding boxes. $\mathbf{t}_i$ represent the location on the bounding box, and $\mathbf{t}_i^*$ the location of the associated positive anchor. The classification loss L_{cls} is a log loss over two classes (object/non-object). According to equation 2.20, the regression loss is defined by [RHGS15]:

$$L_{reg}(t_i, t_i^*) = smooth_{L_1}(\mathbf{t}_i - \mathbf{t}_i^*) \tag{4.20}$$

Since the regression loss is multiplied with the GT label p_i^*, the regression loss is only activated for positive anchors. The two terms are normalized with the two numbers N_{cls} and N_{reg}. Moreover, λ is a weighting parameter between the two losses, whereby the authors set it to 10.

Regressing Mask-level Instances

The input to the second stage are the shared convolutional features and the boxes from the first stage. The output of the second stage is a pixel-level segmentation mask for each given bounding box. In this stage, a mask-level instance is still class-aware. The authors extract a feature for each given bounding box using RoI pooling (see section 2.2.3.1 for more details). RoI pooling produces a fixed size feature from different bounding box sizes. This size is set to 14×14 from Dai et al. [DHS16]. Then, two fully-connected layers are added to represent the feature map of each box. The first fully-connected layer reduces the dimension to 256, while the second layer regresses a pixel-wise mask. Since the resulting pixel-wise mask has a spatial resolution of $m \times m$, the second fully-connected layer consists

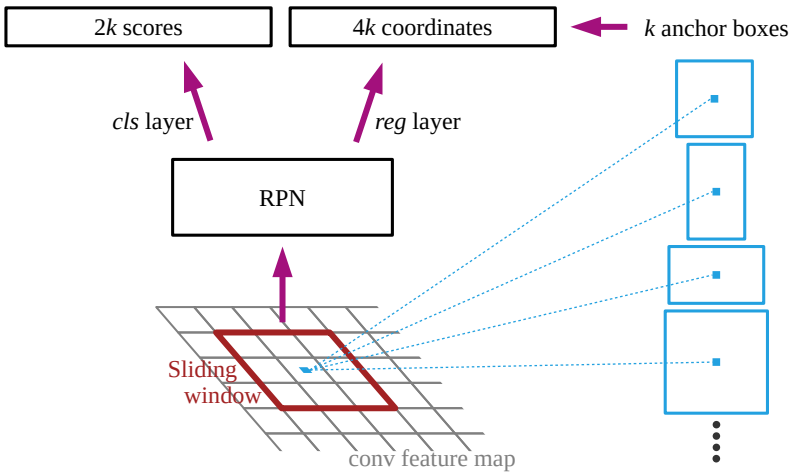


Figure 4.17: **RPN:** Region Proposal Network (RPN) [RHGS15] using "anchors" to determine the position of the bounding boxes.

of m^2 output neurons, each performing binary logistic regression to the GT mask. The authors use $m = 28$. Thus, the second loss term L_2 for regressing masks is defined by [DHS16]:

$$L_2 = L_2(\mathcal{M}(\Theta) \mid \mathcal{B}(\Theta)) \tag{4.21}$$

where M is the network output of the second stage, representing a list of $m \times m$ sized masks. Each output of the m^2 dimensional logistic regression outputs takes values between $[0, 1]$, since the authors use sigmoid activation functions. According to equation 4.21, the mask regression loss L_2 depends on both the masks $\mathcal{M}$ and the bounding boxes $\mathcal{B}$. To reduce computational cost, only few proposed boxes are used to regress segmentation masks.

Categorizing Instances

The inputs to the third stage are the shared convolutional features, the boxes from the first stage and the segmentation masks of the second stage. Since the pixel-level segmentation masks are still class-aware, the third stage outputs category scores for each instance. Similar to the second stage, the authors use the bounding boxes of the first stage to extract a feature map by RoI pooling. This feature map is then "masked" by the pixel-level segmentation mask of the second stage. This "masking" is inspired by the feature masking strategy of Dai et al. [DHS15b] (see section 3.3.1). Thus, the masked feature map focuses on the foreground of the prediction mask and is defined as [DHS16]:

$$\mathcal{F}_i^{Mask}(\Theta) = \mathcal{F}_i^{RoI}(\Theta) \cdot M_i(\Theta) \tag{4.22}$$

where $\mathcal{F}_i^{RoI}(\Theta)$ is the feature map after RoI pooling and $M_i(\Theta)$ is a mask prediction from the second stage (resized to the RoI resolution). The authors followed the idea of Hariharan et al. [HAGM14] (see section 3.3.1) and use a two pathway network for categorizing instances (similar to figure 3.3). The mask-based pathway consists of two 4096-dimensional fully-connected layers that are applied to the masked feature map $\mathcal{F}_i^{Mask}$. Furthermore, the authors use another box-based pathway, where the RoI pooled features are directly

fed into two 4096-dimensional fully-connected layers (this pathway is not illustrated in figure 4.15). Then, the feature vector of the mask-based pathway is concatenated with the feature vector of the box-based pathway. This concatenated feature vector is classified by an $N+1$ ways classifier, where N is the number of categories (plus background category). The box-level pathway can address the cases where the feature is mostly masked out by the mask-level pathway. Since the category predicting depends on both bounding boxes $\mathcal{B}$ (they are used for generating the RoI features) and segmentation masks $\mathcal{M}$, Dai et al. formulate the loss term L_3 as [DHS16]:

$$L_3 = L_3(\mathcal{C}(\Theta) \mid \mathcal{B}(\Theta), \mathcal{M}(\Theta)) \tag{4.23}$$

where $\mathcal{C}$ is the network output representing a list of category predictions for each instance.

End-to-End Training

Finally, the loss of the entire cascade is the weighted sum of all tree previous defined loss terms:

$$L(\Theta) = \underbrace{L_1(\mathcal{B}(\Theta))}_{\text{box loss}} + \underbrace{L_2(\mathcal{M}(\Theta) \mid \mathcal{B}(\Theta))}_{\text{mask loss}} + \underbrace{L_3(\mathcal{C}(\Theta) \mid \mathcal{B}(\Theta), \mathcal{M}(\Theta))}_{\text{category loss}} \tag{4.24}$$

The loss function $L(\Theta)$ is minimized with respect to the network parameter Θ. Furthermore, Dai et al. develop a differentiable RoI warping layer to achieve theoretically valid backpropagation [DHS16] and, thus, achieve end-to-end training including dependencies of earlier stages. While the RoI pooling layer uses max pooling, the RoI warping layer crops a feature map region and warps it into a target size by interpolation.

During training of the proposed neural network, the first stage regresses $\sim 10^4$ bounding boxes. To reduce redundant candidates the authors use non-maximum suppression, where the IoU between candidates is used. The threshold for the IoU of the non-maximum suppression is set to 0.7. The top-ranked 300 boxes serve as input for the second stage. These 300 boxes determine the "pathways" the forward/backward propagated signals are going through during training.

Different stages and different corresponding losses require different stage input labeling. Therefore Dai et al. define positive or negative samples for each stage separately [DHS16]: (1) If the IoU between a regressed bounding box (depending on the "anchors") and a GT bounding box is higher than a threshold, then, this sample is a positive sample. (2) In the second stage, the authors find the highest IoU between the proposed box and the GT mask. If the IoU is greater than 0.5, this proposed box is considered as positive and contributes to the mask regression loss. Otherwise, the box is ignored in the regression loss. The target of the mask regression is the intersection between the proposed box and the GT mask, resized to m×m pixels. (3) In the third stage, the authors considers two different sets of positive/negative samples. In the first set, each mask that has an IoU with the GT bounding boxes greater than 0.5 is positive, otherwise, negative. In the second set, masks where the IoU with GT bounding boxes and the IoU with GT segmentation masks is greater than 0.5, are positive. The rest are negative samples. Therefore, the loss function of the third stage involves two $(N+1)$-way classifiers. The first one is responsible for classifying mask-level instances and the second one is responsible classifying box-level instances (whose scores are not used for inference). The authors use these two classifiers

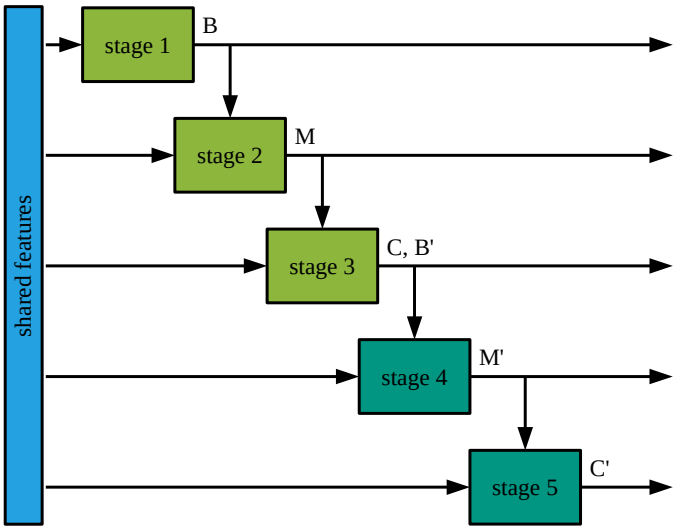


Figure 4.18: **5-stage cascade:** In addition to the 3-stage cascade, the bounding boxes updated by the box regression layer in stage 3 are used as the input to stage 4 [DHS16].

since box-level IoU is more reliable if the the proposed box is not a real instance (e.g., on the background or poorly overlapping with GT).

The shared features are based on the VGG model (see section 2.3). In detail, the authors use the VGG model pretrained on the ImageNet dataset to initialize the shared convolutional layers. During training the first seven convolutional layer, that are responsible for lower level features, are fixed, while the remaining VGG layers, that are responsible for higher level features, are learnable. The additional layers in the MNC model are initialized randomly. During training, Dai et al. use a mini-batch that involves 1 image, 256 sampled anchors for the first stage and 64 sampled RoIs for the second and third stage. The images are resized such that the shorter side has 600 pixels.

Cascades with More Stages

The authors extend their network cascade to more stages (see figure 4.18). At the third stage, a $4(N+1)$-dimensional fully-connected layer for regressing class-wise bounding boxes is added, additional to the classifier layer. The entire 3-stage network cascade is trained as described before. Thus, the third stage outputs not only the category per segmentation mask but also newly regressed bounding boxes. These bounding boxes are used for the fourth and fifth stage. These stages share the same structure as the second and third stage and output classified segmentation masks by using the new boxes. During inference, the 3-stage network is run first and the newly regressed boxes are considered as proposals for the fourth and fifth stage. This is a 5-stage inference and is illustrated in figure 4.18. This inference process can be iterated, but the authors have observed negligible gains. By adapting the training, the 5-stage cascade is also trained end-to-end.

4.2.2 Implementation Details

In this section, we give more details about the used python+caffe reimplementation from Dai et al. [DHS16], in particular in the training parameters and the weight initialization.

Weight Initialization

The feature extractor of MNC is initialized with the VGG-16 parameters. The weights of all remaining layers are either initialized to zero or initialized using a Gaussian distribution. All biases of the remaining layers are initialized with zero.

In stage 1, the weights of the RPN are initialized using Gaussian distribution with a standard deviation of 0.01. In stage 2 and stage 4, the weights of the mask regression layers are initialized with a standard deviation of 0.001. In stage 3 and stage 5, the weights of the first mask categorizing layers are initialized with zero. The fully-connected layers in stage 3 and stage 5, that are responsible to propose new bounding boxes, are initialized with a standard deviation of 0.001. The last fully-connected layers in stage 3 and stage 5 are initialized with a standard deviation of 0.01.

Training Parameters

One crucial training parameter is the learning rate [GBC16, 294]. The learning rate influences the weight update during training (see equation 2.15). Our base learning rate η_b is 0.001 and we use the step learning rate policy from caffe to decrease the learning rate after a certain number of steps. Thereby, the learning rate drops by a factor of γ every s iterations:

$$\eta = \eta_b \cdot \gamma^{\lfloor \frac{i_c}{s} \rfloor} \tag{4.25}$$

where we set $\gamma = 0.1$ and $s = 20,000$. The current iteration is defined by i_c and $\lfloor \cdot \rfloor$ rounds the value down. In total, we train our network $25,000$ iterations. Thus, our learning rate stays 0.001 for the first $20,000$ iterations and decreases according to equation 4.25 to 0.0001 for the last $5,000$ iterations.

Moreover, we use a small weight decay value of 0.0005 (see section 2.2.3.4) to slightly penalize huge weights. To accelerate the training, we use momentum with $\alpha = 0.9$ (see section 2.2.3.3)

4.2.3 Weakly-Supervised Training

Our weakly-supervised approach is based on the 5-staged network of Dai et al. [DHS16] (see section 4.2.1) Summarizing the architecture briefly, this network is divided into five stages to segment instances:

Stage 1 proposes bounding boxes

Stage 2 proposes a segmentation mask per bounding box

Stage 3 categorizes each mask and proposes new bounding boxes

Stage 4 proposes a segmentation mask per bounding box

Stage 5 categorizes each new mask

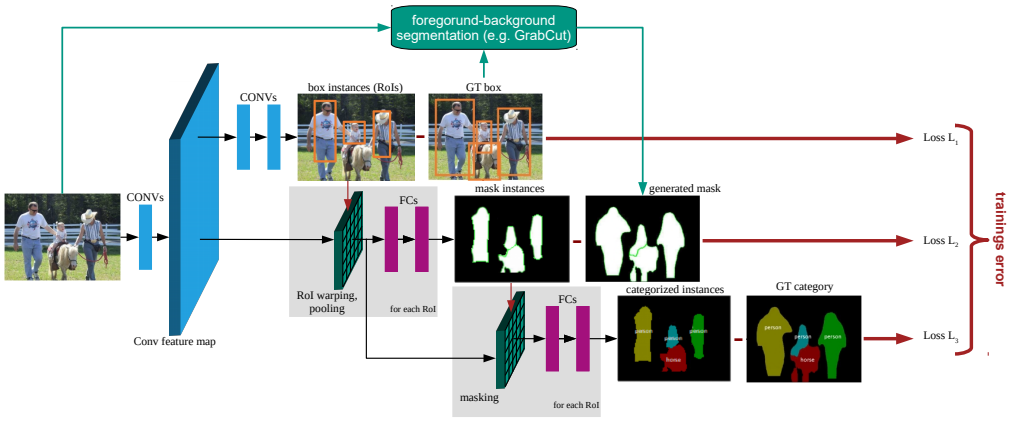


Figure 4.19: **Weakly-supervised 3-staged MNC Training:** The MNC model is end-to-end trained using a dedicated loss for each stage. A foreground-background segmentation algorithm is used to generate masks that are learned instead of GT masks.

Each stage consists of a dedicated loss and is trained using the outputs of previous stages. For example, stage 2 needs bounding boxes from stage 1 to segment instances within these boxes. Our goal is to train this network cascade without pixel-wise segmentation masks, but only with bounding box information. By using bounding boxes as weak supervision, we can train stage 1, stage 3 and stage 5 unchanged as in the original code. Since pixel-wise segmentation masks are missing, we use the foreground-background segmentation algorithms we presented in section 4.1.2 to weakly generate object segmentation masks within each bounding box. We use these weakly generated segmentation masks instead to train stage 3 and stage 5. Thus, the MNC learns manually labeled bounding boxes and weakly generated segmentation masks. We visualize the weakly-supervised training for the 3-stage cascade in figure 4.19.

Since the network learns both bounding boxes and segmentation masks, the training data is stored separately. For each training image a list of all bounding boxes is stored, including a list of all box positions within the image and a list of all corresponding classes. We use the information in this file to determine the mask segmentation training data. Therefore, we use our segmentation algorithms. For each training image a list of segmentation masks is stored. Each mask has the size of the corresponding bounding box. In addition, the class of each mask is stored and is equal to the class of the corresponding bounding box. Furthermore, we use data augmentation for training our segmentation model. We double our training set by flipping each box and the corresponding segmentation mask. These new boxes and masks are stored in addition to the original training data.

5. Evaluation

In this chapter we evaluate our baselines and our end-to-end trained segmentation model with respect to semantic segmentation, instance segmentation and runtime. We evaluate both on the same evaluation machine. This machine contains an Intel® Core™ i5-750 with four cores. This CPU has a basic frequency of 2.66 GHz and a maximal frequency of 3.20 GHz. Moreover, it contains an NVIDIA® GeForce GTX 1070 with 8 GB VRAM and 16 GB RAM working memory. Moreover, we compare our results with other segmentation models we described in chapter 3. We use the evaluation metrics presented in section 2.6.

5.1 Baselines

In this section, we evaluate our baselines in the image segmentation task. Thus, we can test our foreground-background segmentation algorithms without training our neural network. In addition, these evaluation results are the basis for our decision which algorithm we use to train our network. Our baselines are built up according to section 4.1 and consist of three independent steps. We investigate the effects of each individual step on the final segmentation results. First, we examine the effects of YOLO bounding boxes on the final segmentation results (see section 5.1.1). Second, we present the results in image segmentation for different foreground-background segmentation algorithms (see section 5.1.2). Finally, we investigate the effects of DenseCRF in the postprocessing step (see section 5.1.3). Thereby, we present results in both semantic segmentation and instance segmentation by evaluating on the PascalVOC validation dataset from 2012. Furthermore, we analyze the runtime of our baselines (see section 5.1.4).

5.1.1 Effects of YOLO Object Detection on Results

We use YOLO as object detector in our baseline structure trained on the PascalVOC dataset. According to Redmon et al., YOLO struggles with small objects that appear in groups and, therefore, makes more localization errors compared to other object detectors [RDGF16]. However, YOLO is less likely to predict False Positives (FPs) on background [RDGF16]. An indicator for the performance capability of YOLO is that it detects ∼3000 objects compared to ∼3500 Ground Truth (GT) boxes in the PascalVOC evaluation set.

Method	GT Bounding Boxes		YOLO Bounding Boxes	
	w/o DenseCRF	DenseCRF	w/o DenseCRF	DenseCRF
Bounding Boxes	60.6	-	54.2	-
GrabCut	69.8	64.4	62.0	57.1
K-Means Mid	45.5	56.1	42.9	51.3
K-Means Overlap (ada.)	56.0	61.7	47.1	54.2
K-Means Overlap (fix)	59.2	55.4	53.9	52.1
MBOD	32.9	27.8	31.4	26.5
MCG	73.4	65.9	58.4	57.6
RBD	32.4	27.7	31.3	26.8
YOLO Segmentation	48.0	50.0	46.2	48.0

Table 5.1: **Semantic Segmentation:** Evaluation results on PascalVOC validation dataset of our baselines using mIoU.

In the following, we examine the influence of the errors made by the YOLO object detector on the final segmentation results. For this purpose, we compare the semantic segmentation results of our baselines using YOLO bounding boxes to the results of our baselines using GT bounding boxes. According to table 5.1, using YOLO bounding boxes decreases the mIoU independent of the specific segmentation algorithm or the postprocessing. Averaged over all segmentation algorithms the mIoU decreases by 5.59% with no DenseCRF (4.43% with DenseCRF). Furthermore, the influence of the errors made by YOLO on the final results severely varies and ranges from 0.9 (RBD) up to 15.0 (MCG).

Moreover, the results in table 5.1 using GT bounding boxes indicate the maximum achievable benefit by enhancing the capabilities of the object detection step. Depending on the segmentation algorithm in the second step this enhancement has great effects on the final segmentation results.

5.1.2 Effects of Segmentation Algorithms on Results

In the following, we evaluate the influence of specific foreground-background segmentation algorithms on the final segmentation results. To maintain comparability, we obtain our results using YOLO bounding boxes and without DenseCRF. Furthermore, we focus on the comparison with YOLO bounding boxes as segmentation masks. These bounding boxes achieve good results, since a lot of objects have a rectangular form (e.g. bus, car, train and tv/monitor) and, thus, a box is good segmentation hypothesis.

Segmentation Results of RBD and MBD

First, we present the results of RBD and MBOD, the salient object detectors. These segmentation algorithms achieve the worst results in mIoU (see table 5.1), $mAP^r_{0.3}$ (see table 5.2) and ABO (see table 5.2) Furthermore, they achieve ∼28% worse results in mIoU compared to YOLO bounding boxes.

Both algorithms detect salient regions in the center of slightly enlarged bounding boxes and assume that regions near the boundary of these boxes are more likely background. In addition, salient regions exhibit a large color difference compared to the remaining regions.

Method	$mAP^r_{0.3}$	$mAP^r_{0.5}$	$mAP^r_{0.75}$	ABO
YOLO Bounding Boxes	<u>54.1</u>	21.2	2.0	38.1
GrabCut	51.4	<u>40.0</u>	<u>12.7</u>	<u>42.7</u>
K-Means Mid	33.6	7.7	0.2	29.7
K-Means Overlap (ada.)	38.3	15.9	1.3	33.0
K-Means Overlap (fix)	39.0	17.9	1.4	32.7
MBOD	26.7	3.4	0.1	26.6
MCG	46.6	27.6	7.3	39.9
RBD	26.1	9.2	1.0	25.6
YOLO Segmentation	39.3	12.7	0.5	30.6

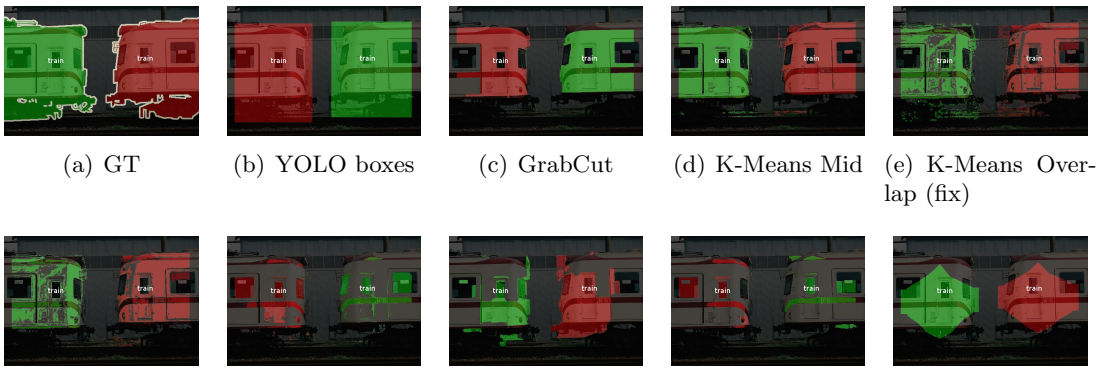
Table 5.2: **Instance Segmentation:** Evaluation results on PascalVOC validation dataset of our baselines using mAP and ABO.

Therefore, the salient object detection is prone to false foreground-background segmentation if the color difference between the actual object and the background is small. Figures 5.1(i) and 5.1(g) show examples where this problem occurs and where the segmentation of RBD and MBOD fail. The color difference between the white trains (objects) and the light background is too small. Therefore, the algorithms segment only salient parts of the train like the red paintings or the dark windows.

Although RBD and MBOD share the same main problem due to the saliency assumption of objects, RBD achieves better $mAP^r_{0.5}$ and $mAP^r_{0.75}$ results. Due to better results with respect to higher IoU thresholds, RBD achieves more accurate object segmentations compared to MBOD. Since the ABO and $mAP^r_{0.3}$ results are nearly the same, the segmentation of RBD fails in some cases where MBOD segments the object. RBD uses superpixels to segment the salient object. According to Zhu et al. [ZLWS14], superpixels near the boundary of the bounding box are more likely background. In case of small objects, the bounding box contains fewer superpixels in contrast to large objects and, thus, the ratio between one superpixel compared to the whole object is higher. The effect on IoU, when an object superpixel is falsely set to background, is accordingly higher for small objects. Furthermore, the ratio of superpixels that are connected to the boundary is higher for small superpixels. Therefore, RBD achieves worse results than MBOD for small objects.

Segmentation Results of YOLO Based Segmentation

According to table 5.1 and table 5.2, the YOLO based segmentation baseline (described in section 4.1.2.1) does not gain any improvements compared to YOLO bounding boxes. Due to the resizing and thresholding of the YOLO grid, the segmentation masks are almost always polygons near the size of the bounding boxes (see figure 5.1(j) and figure 5.2(j)). Therefore, the $mAP^r_{0.3}$ metric and the $mAP^r_{0.5}$ metric are much higher than the corresponding values of both saliency object segmentation algorithms. However, the segmentation masks of our YOLO baseline are rarely accurate, since the $mAP^r_{0.75}$ is lower than the $mAP^r_{0.75}$ of RBD. In total, the object segmentation masks are indeed inaccurate but also the objects are more often detected within bounding boxes than using the two saliency algorithms. Since we use YOLO as object detector, it already knows that there is an object. Our YOLO based segmentation baseline can detect two objects per grid cell and,



(a) GT (b) YOLO boxes (c) GrabCut (d) K-Means Mid (e) K-Means Overlap (fix)

(f) K-Means Overlap (adaptive) (g) MBOD (h) MCG (i) RBD (j) YOLO Seg.

Figure 5.1: **Segmentation Results:** Image segmentation results of all baselines for an image that shows two trains. We use bounding boxes provided by YOLO without any postprocessing

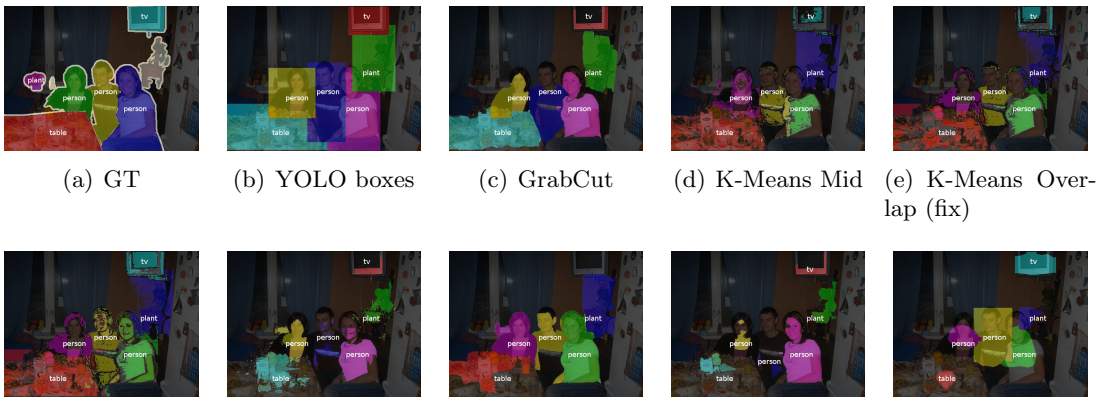
thus, it is possible that no pixel is treated as foreground pixel. However, in most cases, a grid cell detects only one object and, thus, at least some pixels are foreground.

Segmentation Results of K-Means Based Segmentation

According to table 5.1 and table 5.2, our baselines based on K-Means achieve worse results than YOLO bounding boxes. Thereby, the two different baselines (Mid and Overlap) make different assumptions that cause different problems.

Our K-Means Mid baseline assumes that the object is in the center of the bounding box. There are three cases where this assumption leads to problems during the segmentation. First, the center of the bounding box does not always cover the whole color variety of an object. Figure 5.2(d) shows an example where the centers cover only the shirts of the persons. Thus, our K-Means Mid baseline does not segment the heads of the persons. Second, the assumption is incorrect if the center of the bounding box contains also background pixels. In figure 5.2(d), the assumption for the potted plant is incorrect and, thus, our baseline additionally sets the wall as foreground. Last, our clustering approach only takes the pixel color into account. In case of similar foreground and background color, our baseline can not differentiate between them. In figure 5.1(d) a person wearing a black tie stands in front of a blackish background. Since the tie is in the center of the bounding box, also parts of the background with similar color are segmented as object.

Our second K-Means baseline shares the last problem of our K-Means Mid baseline, since both baselines are based on clustering similar pixels. If the background exhibits a similar color as the object, our second baseline also fails. However, this baseline has no further problems caused by its assumptions and, therefore, achieves better results than our K-Means Mid baseline. Thereby, the fix threshold baseline achieves better results in semantic segmentation compared to the adaptive threshold baseline (see table 5.1) while the threshold type has barely influence on the instance segmentation results.



(a) GT (b) YOLO boxes (c) GrabCut (d) K-Means Mid (e) K-Means Overlap (fix)

(f) K-Means Overlap (adaptive) (g) MBOD (h) MCG (i) RBD (j) YOLO Seg.

Figure 5.2: **Segmentation Results:** Image segmentation results of all baselines for an image that shows three persons sitting at a table. We use bounding boxes provided by YOLO without any postprocessing

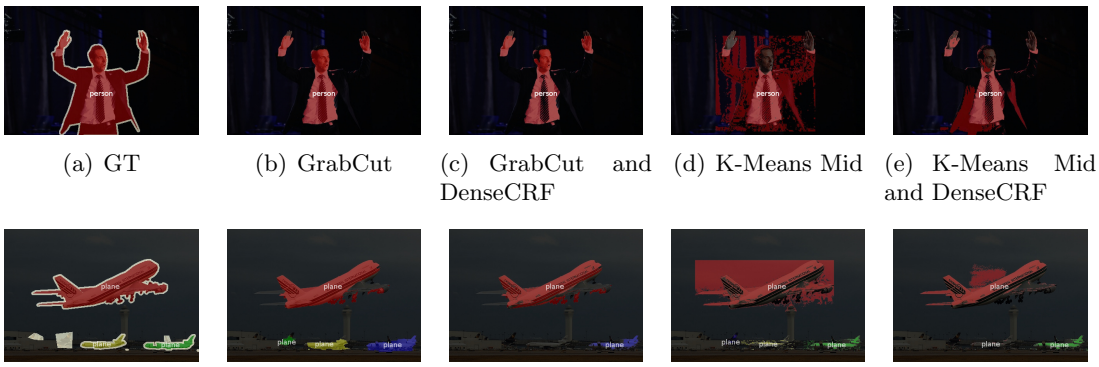
Segmentation Results of MCG and GrabCut

MCG and GrabCut are the only baselines that perform better than YOLO bounding boxes in both semantic and instance segmentation.

GrabCut achieves the best results amongst all baselines in semantic and instance segmentation. However, if we use a small IoU threshold for mAP, YOLO bounding boxes still achieve the best results. The benefit of YOLO bounding boxes is that there is always an IoU greater zero with the GT mask. The IoU only depends on the size of the object within the bounding box. Thus, YOLO bounding boxes can achieve good results using small thresholds. Furthermore, table 5.2 shows that the results decrease really fast with increasing threshold compared to GrabCut and MCG.

So far, we compare our results without any postprocessing and YOLO bounding boxes. Thereby, MCG achieves the second best results. However, MCG achieves the best results in semantic segmentation using GT bounding boxes. Since we use GT bounding boxes to generate segmentation masks for training our neural network, these results are also considerable. Comparing the results of MCG using GT boxes to the result using YOLO, the mIoU decreases the most between all foreground-background segmentation algorithms. The reason for this is that an object proposal generated by MCG must have an overlap threshold of 0.99. If the YOLO bounding boxes are not accurate enough, this threshold is not always achievable. In case of a slightly smaller bounding box, a perfect segmentation would be rejected since the overlap is smaller than 1.0. However, we also evaluated MCG with smaller thresholds but we achieved worse results. Therefore, we also use a threshold of 1.0 for segmentation with YOLO bounding boxes.

All in all, only these two approaches achieve better results than simple YOLO bounding boxes. Therefore, we choose these two algorithms to train our neural network.



(a) GT (b) GrabCut (c) GrabCut and DenseCRF (d) K-Means Mid (e) K-Means Mid and DenseCRF

(f) GT (g) GrabCut (h) GrabCut and DenseCRF (i) K-Means Mid (j) K-Means Mid and DenseCRF

Figure 5.3: **DenseCRF Results:** Semantic segmentation results of GrabCut and K-Means Mid with and without postprocessing.

5.1.3 Effects of DenseCRF on Results

In the following, we investigate the influence of DenseCRF on the final semantic segmentation results for each baseline separately. The main property of DenseCRF is to facilitate consistent labeling. However, according to table 5.1, only K-Means Mid, K-Means with adaptive threshold and the YOLO based segmentation benefit from this property.

Although DenseCRF serves to improve the segmentation results, it also has problems that worsen the segmentation results instead. The main problem of DenseCRF is the removal of details in the object or the removal of whole objects at worst due to the property of consistent labeling. Figures 5.3(b) and 5.3(c) illustrate the problem of detail removal. In figure 5.3(b), GrabCut primarily segments the tie (black) and the shirt (white). Both strongly vary in their color and, thus, the tie as the smaller area is removed by the DenseCRF. At worst, not only details of the object are removed but also whole objects due to their color difference to the background, e.g. the two planes in figure 5.3(h). Thereby, mostly small objects are affected by this problem.

Even though DenseCRF causes problems, some baselines benefit of this postprocessing step, e.g. the result of K-Means Mid increases by 8.4% (see table 5.1). Figures 5.3(d) and 5.3(i) show the segmentation results of K-Means Mid. Due to incorrect assumptions, K-Means Mid sets a large background area to foreground. For comparison, figures 5.3(e) and 5.3(j) show the results after applying the DenseCRF. Especially figure 5.3(j) illustrates the positive effect of DenseCRF. K-Means Mid sets the sky within the bounding box as foreground, but the larger part of the sky is outside the bounding box and, thus, set to background. The DenseCRF facilitates consistent labeling and, hence, sets almost the whole sky within the bounding box to background.

All in all, some baselines benefit of DenseCRF, but our best baselines (GrabCut and MCG) achieve worse results with DenseCRF. Therefore, an universal statement with respect to the use of DenseCRF is not possible and, thus, the use of DenseCRF has to be evaluated for each foreground-background segmentation algorithm separately.

5.1.4 Runtime

The runtime is crucial for security-critical applications, e.g. autonomous driving. Therefore, we compare the runtime of our baselines compared to their results in semantic segmentation (see table 5.3). The object detection uses the GPU of our evaluation machine while segmentation and postprocessing use the CPU.

Real-time Capability

In practice, real-time capable segmentation should exhibit predictable and fast inference time per image, i.e. small standard deviation and low average inference time. However, the inference time per image of our baselines strongly varies and ranges from ∼100 ms up to ∼23,000 ms. The only real-time capable baselines are the YOLO bounding boxes and the YOLO based segmentation, since they both primarily depend on the inference time of the fast YOLO model (see table 5.3).

All other baselines achieve less than 1 frame per second and exhibit a standard deviation greater than 600 ms. Thus, they are not real-time capable. According to our baseline structure, each baseline depends on the inference time of the YOLO model to determine bounding boxes. In addition, all not real-time capable baselines segment objects within these bounding boxes. Therefore, the final inference time additionally depends on runtime of the foreground-background segmentation and on the number of detected objects. GrabCut, for example, takes only hundreds of milliseconds for a foreground-background segmentation. However, this segmentation is applied for each detected bounding box and, thus, increases the inference time for the whole image. Since the segmentation algorithm is applied several times depending on the number of bounding boxes, the standard deviation correlates with the average inference time.

Segmentation-Runtime-Trade-off

Although most baselines are not real-time capable, we also compare the runtime with respect to the segmentation results. MBOD and RBD, for example, achieve the worst segmentation results and are significantly slower, in particular RBD.

Moreover, our clustering baselines exhibit different inference times. The K-Means Mid baseline clusters pixels within each bounding box in contrast to our K-Means Overlap baselines where pixels are clustered over the whole image. In most cases, the number of pixels in all bounding boxes is smaller than the number of pixels in the whole image and, thus, our K-Means Mid baseline takes less time for segmentation. However, the standard deviation of the K-Means Mid baseline is higher due to the different number and sizes of objects in the image. In contrast, our K-Means Overlap baselines are independent of the number of bounding boxes during the clustering and only depend on the input image size. Due to the computation of the overlap between the bounding boxes and the clusters, our K-Means Overlap baselines are not completely independent. However, all clustering baselines add neglectable segmentation gain compared to YOLO bounding boxes taking their significantly longer inference time into account.

Although MCG achieves better segmentation results than YOLO bounding boxes, the inference time is significantly higher. MCG also computes its object proposals only once per image and the remaining time is taken for the computation of the overlap similar to

Method	w/o DenseCRF		w/ DenseCRF	
	Runtime [ms]	mIoU	Runtime [ms]	mIoU
YOLO Bounding Boxes	90 (±82)	52.2	-	-
GrabCut	2,583 (±1,801)	62.0	3,250 (±2,127)	57.1
K-Means Mid	1,516 (±949)	42.9	2,618 (±1,492)	51.3
K-Means Overlap (ada.)	3,212 (±610)	47.1	4,236 (±1,294)	54.2
K-Means Overlap (fix)	3,266 (±607)	53.9	4,397 (±1,289)	52.1
MBOD	1,426 (±893)	31.4	2,295 (±1,374)	26.5
MCG	20,089 (±8,965)	58.4	22,970 (±9,464)	57.6
RBD	13,426 (±10,527)	31.3	14,249 (±11,100)	26.8
YOLO Segmentation	101 (±80)	46.2	623 (±550)	48.0

Table 5.3: **Runtime:** Comparison of baselines in runtime (in milliseconds) and their mIoU. All baselines using YOLO bounding boxes.

the K-Means Overlap baseline. However, the proposal generation has a huge standard deviation according to Arbeláez et al. [APTB+14]. Therefore, MCG exhibits a slow inference time with huge standard deviation.

GrabCut achieves the best segmentation results and is reasonably fast compared to other baselines. Therefore, GrabCut is the best choice for image segmentation, since it provides the best trade-off between runtime and mIoU.

Effects of DenseCRF on the Runtime

Finally, using a DenseCRF as postprocessing has not only impact on the segmentation results but also on the runtime. Averaged over all baselines, using a DenseCRF increases the runtime by 1,127 ms. DenseCRF yields neglectable benefits with respect to the small averaged increase in segmentation result (+0.05 mIoU) and the comparably huge increase in runtime. We examine the usage of DenseCRF for each baseline separately (see table 5.1). As we see, the benefit varies a lot. A gain of ∼6% mIoU, for example, entirely justifies the use of DenseCRF.

5.2 End-to-End Model

In this section, we evaluate our weakly-supervised end-to-end trained MNC model (described in section 4.2) in the image segmentation task (see section 5.2.1). We train this network with our two best foreground-background segmentation algorithms (GrabCut and MCG) according to the results in section 5.1. Furthermore, we analyze the runtime of our model (see section 5.2.2).

5.2.1 Image Segmentation

In this section, we present the results of our weakly-supervised MNC model (called Weak-MNC). We build upon the MNC model from Dai et al. [DHS16] and learn weakly generated segmentation masks instead of manually labeled segmentation masks. To weakly generate masks, we choose the two foreground-background segmentation algorithms that achieve the best baseline results (see section 5.1): GrabCut and MCG. In addition, we train our

	$mAP^r_{0.3}$	$mAP^r_{0.5}$	$mAP^r_{0.75}$	ABO	mIoU
Bounding Boxes	<u>54.0</u>	21.2	1.9	39.0	54.2
GrabCut	51.4	<u>39.9</u>	<u>12.7</u>	<u>50.8</u>	<u>62.0</u>
MCG	46.6	27.6	7.2	44.9	58.4
Full-MNC (trained on SBD)	68.0	58.3	24.4	60.2	61.2
Weak-$MNC_{GrabCut}$ (trained on SBD)	57.9	<u>45.3</u>	<u>14.1</u>	49.5	52.7
Weak-$MNC_{GrabCut}$ (trained on PascalVOC)	54.0	41.5	12.6	49.3	50.0
Weak-MNC_{MCG} (trained on SBD)	<u>58.1</u>	43.8	11.6	48.2	<u>53.9</u>
Weak-MNC_{MCG} (trained on PascalVOC)	56.0	41.0	11.0	<u>49.8</u>	52.0
Weak-$MNC_{GrabCut \cap MCG}$ (trained on SBD)	52.5	41.2	12.4	44.6	50.8
Weak-$MNC_{GrabCut \cap MCG}$ (trained on PascalVOC)	51.3	39.6	12.9	46.7	50.1

Table 5.4: **Image Segmentation:** Evaluation on PascalVOC validation set of our Weak-MNC models and the self-trained Full-MNC model using mAP and ABO for instance segmentation and mIoU for semantic segmentation.

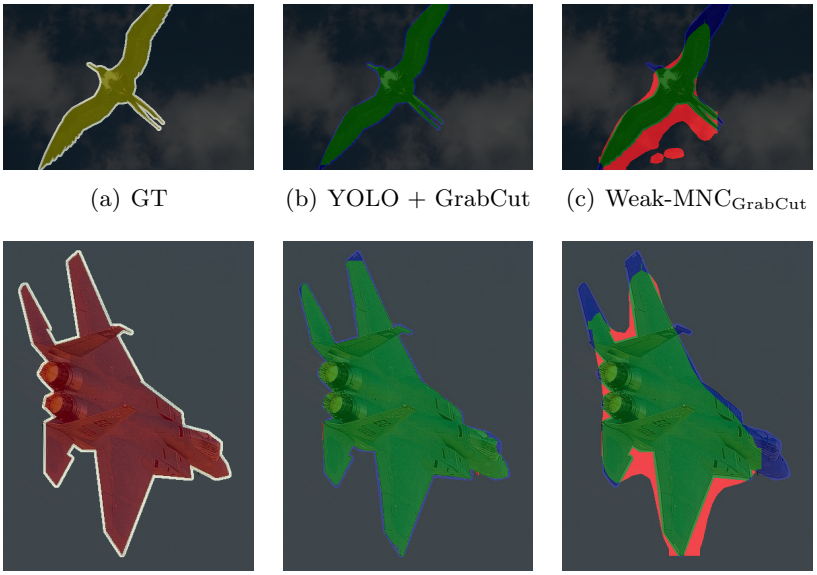
neural network using the intersection between these two algorithms (GrabCut∩MCG). We train and evaluate our models on both the PascalVOC dataset and the SBD dataset. Furthermore, we train the MNC framework with the fully-supervised SBD training data (called Full-MNC). Since we use a random initialization (see section 4.2.2) we achieve a different result compared to the paper from Dai et al. [DHS16].

We evaluate our Weak-MNC model and the Full-MNC model in both semantic and instance segmentation. Thereby, we focus on the comparison between our Weak-MNC models and the corresponding baselines and between our Weak-MNC models and the Full-MNC model. In addition, we present the effects of different training sets on the final segmentation results. Moreover, we evaluate our models class-wise to detect difficult classes and more frequent class confusion.

Comparison of Weak-MNC and Baselines - Semantic Segmentation

First, we present the results in semantic segmentation with respect to the corresponding baseline results. In comparison, our weakly-supervised models achieve worse results. In the following, we discuss the results in detail and present the reason for the worse neural network results in semantic segmentation.

According to table 5.4, our Weak-MNC_{MCG} model achieves the best results in semantic segmentation compared to slightly worse results of our Weak-$MNC_{GrabCut}$ model. However, the results are worse than the corresponding baseline results, although the results in instance segmentation are better (see table 5.4). The reason for this is that the baseline generates more precise segmentation masks compared to our Weak-$MNC_{GrabCut}$ model



(a) GT (b) YOLO + GrabCut (c) Weak-MNC$_{\text{GrabCut}}$

(d) GT (e) YOLO + GrabCut (f) Weak-MNC$_{\text{GrabCut}}$

Figure 5.4: **Semantic Segmentation Results:** Comparison of our Weak-MNC$_{\text{GrabCut}}$ model and the corresponding baseline in semantic segmentation (green pixels are TP, red pixels are FP, blue pixels are FN and black pixels are TN).

and the metric computation. In general, MNC learns only a 28×28 segmentation mask per object and, hence, can not generate edge-precise masks. Figure 5.4 shows two examples, where our baseline generates more edge-precise masks. In instance segmentation, the IoU between a single GT mask and a single predicted mask is computed. In contrast, the mIoU for semantic segmentation is computed according to equation 2.26 over all pixels in the validation set. Thus, pixel errors have different effects on the two evaluation metrics, in particular in case of small objects. In semantic segmentation, there is no difference between small objects and one similar large object when computing the mIoU. Thus, the mIoU only depends on the absolute number of pixel errors and is independent of the object sizes. In instance segmentation, the mAP depends on the number and the size of objects in contrast to the mIoU. The same number of pixel error has a different influence to the mAP depending on the object size. In case of smaller objects, it is more likely that the IoU falls below the threshold compared to large objects and, thus, the object is treated as FP. As a consequence, our weakly-supervised neural networks achieve worse results compared to the corresponding baselines due to less precise segmentation masks.

Comparison of Weak-MNC and Baselines - Instance Segmentation

Second, we discuss the results in instance segmentation with respect to the corresponding baseline results. In comparison, our weakly-supervised models achieve better results. Since our models achieve worse results in semantic segmentation, we discuss the results in detail. Furthermore, we present the reason for the better neural network results in instance segmentation and we illustrate the reason on basis of our GrabCut baseline and our Weak-MNC$_{\text{GrabCut}}$ model.

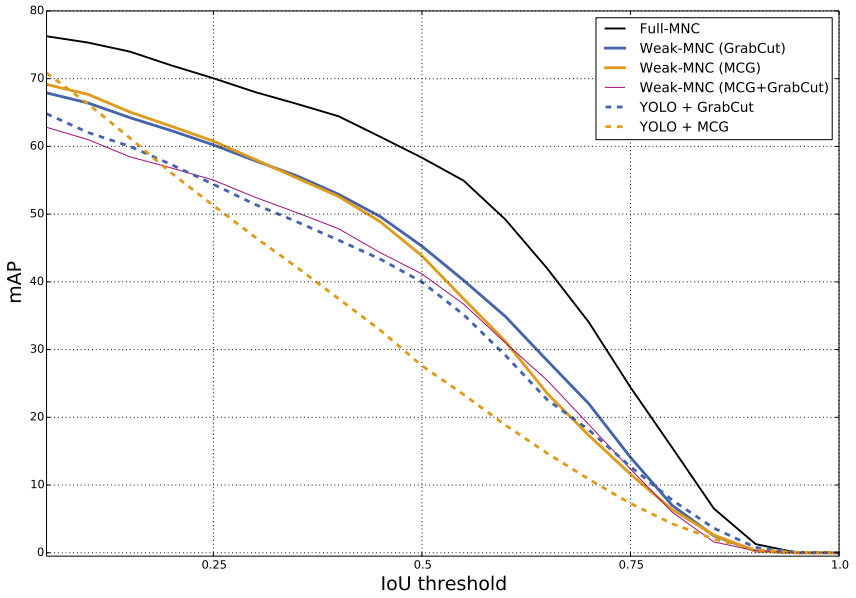
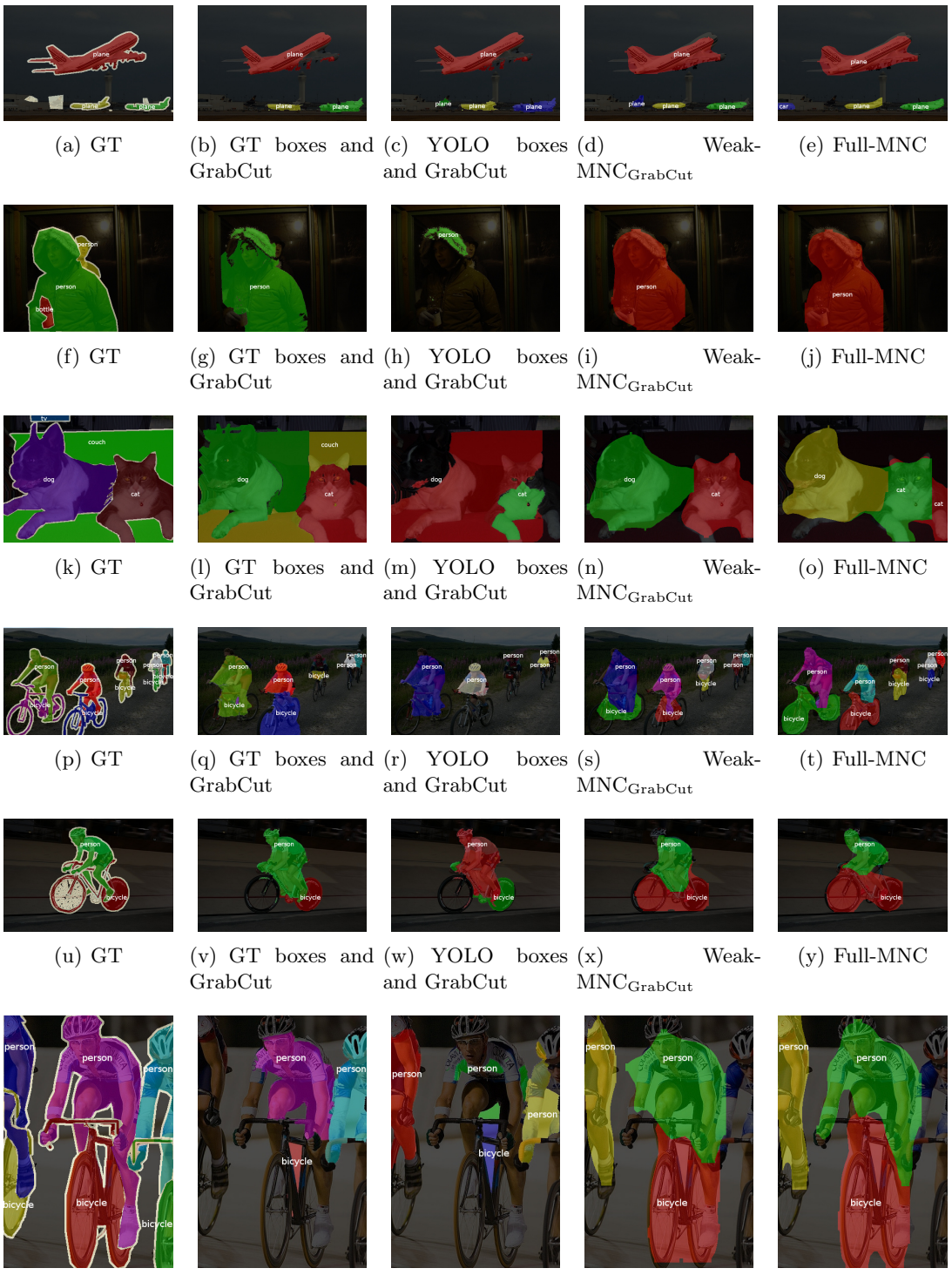


Figure 5.5: **Instance Segmentation Results:** Results in instance segmentation (mAP metric) with different IoU thresholds. We compare the Weak-MNC with the corresponding baselines.

According to table 5.4 and in particular to figure 5.5, our Weak-MNC$_{\text{MCG}}$ model achieves the best weakly-supervised results for small IoU thresholds in instance segmentation. For thresholds bigger than ∼0.35 our Weak-MNC$_{\text{GrabCut}}$ model outperforms our Weak-MNC$_{\text{MCG}}$ model and achieves the best weakly-supervised results. Thus, our Weak-MNC$_{\text{MCG}}$ model detects more objects with less IoU compared to our Weak-MNC$_{\text{GrabCut}}$ model that detects slightly less objects but with higher IoU. We show results of our Weak-MNC$_{\text{GrabCut}}$ model in figures 5.9 and 5.10. The combination of both segmentation algorithms, i.e. the intersection between GrabCut and MCG achieves worse results than the models trained on the segmentation masks of the individual algorithm. Since GrabCut and MCG often generate contradictory segmentation masks and, thus, the intersection causes small masks, our Weak-MNC$_{\text{GrabCut}\cap\text{MCG}}$ model performs worse.

In the following we focus on the better results in instance segmentation according to figure 5.5. The main problem of our baseline algorithms is that detection and segmentation are independent. Although YOLO predicts not only bounding boxes but also provides the corresponding classes, the segmentation algorithms do not use the class information of the bounding boxes to segment the object. These algorithms segment the object according to low level features like the pixel probability distribution, the pixel locations within the bounding boxes or edges defined by pixels. Neural networks are able to learn the underlying structure of an object based on high level features. Therefore, neural networks can learn objects independently to some low level feature (e.g. color and edges) if the object exhibits different values in this feature and focuses on the more relevant low level features. This holds in particular for the MNC model, since it is trained end-to-end, shares the same



(a) GT (b) GT boxes and GrabCut (c) YOLO boxes and GrabCut (d) Weak-MNC$_{\text{GrabCut}}$ (e) Full-MNC

(f) GT (g) GT boxes and GrabCut (h) YOLO boxes and GrabCut (i) Weak-MNC$_{\text{GrabCut}}$ (j) Full-MNC

(k) GT (l) GT boxes and GrabCut (m) YOLO boxes and GrabCut (n) Weak-MNC$_{\text{GrabCut}}$ (o) Full-MNC

(p) GT (q) GT boxes and GrabCut (r) YOLO boxes and GrabCut (s) Weak-MNC$_{\text{GrabCut}}$ (t) Full-MNC

(u) GT (v) GT boxes and GrabCut (w) YOLO boxes and GrabCut (x) Weak-MNC$_{\text{GrabCut}}$ (y) Full-MNC

(z) GT (aa) GT boxes and GrabCut (ab) YOLO boxes and GrabCut (ac) Weak-MNC$_{\text{GrabCut}}$ (ad) Full-MNC

Figure 5.6: **Instance Segmentation Results:** PascalVOC validation images segmented by our best weakly-supervised neural network (Weak-MNC$_{\text{GrabCut}}$ trained on SBD) compared to our GrabCut baseline (GT or YOLO bounding boxes) and the self-trained Full-MNC model.

weights of the feature extractor and, moreover, learns the task depending on the output of earlier tasks. To illustrate this, compare the results of our Weak-MNC$_{\text{GrabCut}}$ model and our GrabCut baseline. In figure 5.6(w) and 5.6(x) a person rides a bicycle. Our GrabCut baseline segments only parts of the bicycle, while our Weak-MNC$_{\text{GrabCut}}$ model uses the knowledge of the class and segments the whole bicycle. Since the bicycle is not segmented in figure 5.6(v), that shows the result using GrabCut with GT bounding boxes, the problem is not caused by the object detection but the segmentation. This difference is more significant in figures 5.6(r) and 5.6(s), where five persons ride a bicycle. Our GrabCut baseline totally fails to segment the bicycles, although YOLO detects them. Moreover, our baseline segments only three of five persons reasonably good. The remaining two persons are also found by YOLO, but their segmentation failed. GrabCut only uses the bounding box information but this information is too little to segment the persons. GrabCut segments only small parts of the persons instead. In contrast, our Weak-MNC$_{\text{GrabCut}}$ model segments not only all five persons reasonably good but also segments three of five bicycles. Since the first task in the network learns features commonly used of all tasks, the network can segment the person better, since it "knows" not only that there is a person but also the structure of a person. We show more examples in figure 5.6, where our Weak-MNC$_{\text{GrabCut}}$ model performs better in instance segmentation than the corresponding baseline.

Comparison of Weak-MNC and Full-MNC

In the following, we compare our Weak-MNC using different foreground-background segmentation algorithms to the self-trained Full-MNC evaluating on the PascalVOC validation set (see table 5.4).

Of course, the self-trained Full-MNC model using fully-supervised training data achieves better results compared to our Weak-MNC models. However, our best Weak-MNC model achieves only maximum ∼13% less mAP depending on the IoU threshold and only ∼8% less mIoU while allowing to reduce the annotation effort significantly.

Effects of Different Training Sets on Results

Since SBD is an extension of PascalVOC, we present the effect of different training sets on the image segmentation results (see table 5.4). Thereby, we evaluate all models ain the PascalVOC validation set.

We train our models on both the PascalVOC and SBD training sets, whereby SBD includes 3.5-times more images than PascalVOC. Thereby, we achieve different results. Although in some cases the models trained on the PascalVOC training set perform better, we achieve in general better results in both semantic and instance segmentation if we use the larger SBD training set. Depending on the foreground-background segmentation, the gain in mAP is up to ∼11% and the gain in mIoU is up to ∼5%.

Class Confusion in Semantic Segmentation

In the following, we evaluate our Weak-MNC$_{\text{GrabCut}}$ model class-wise in semantic segmentation. By the corresponding confusion matrix (see figure 5.7) of the Weak-MNC$_{\text{GrabCut}}$ model, we can detect more frequent class confusions due to similar structure or imprecise segmentation. In the confusion matrix, the classes are grouped in their overlying categories (e.g. vehicles and animals).

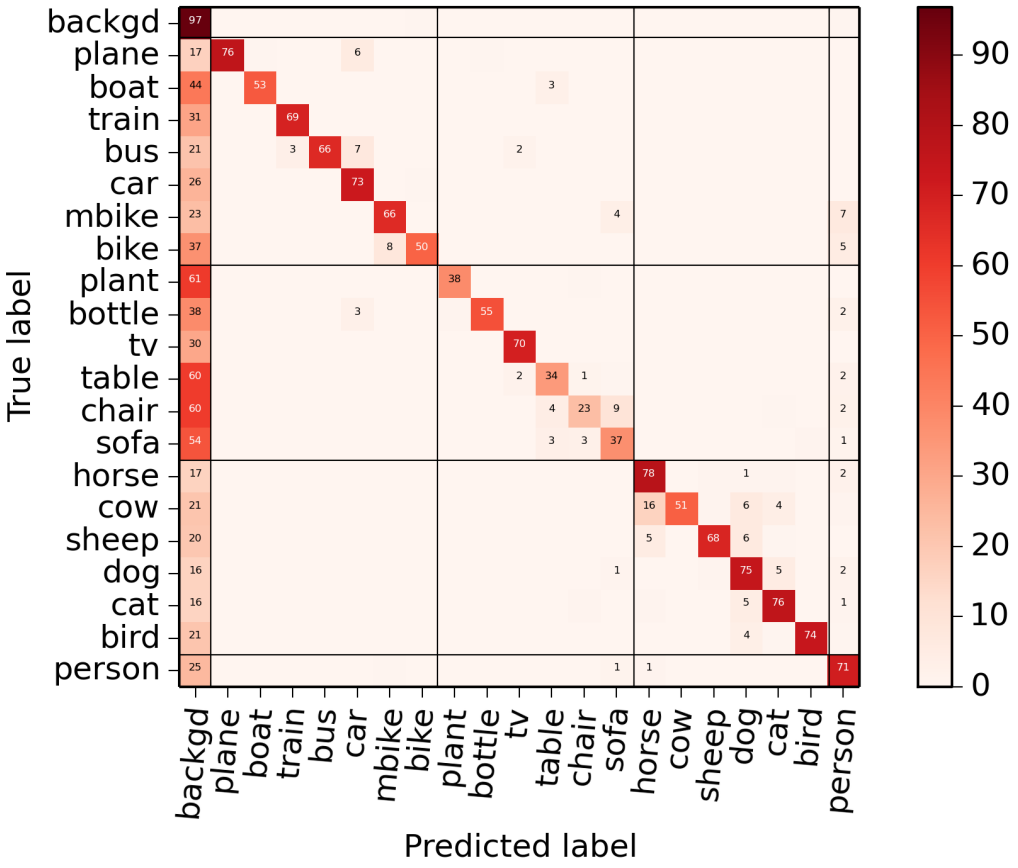


Figure 5.7: **Confusion Matrix:** Confusion matrix of our Weak-MNC$_{\text{GrabCut}}$ model in semantic segmentation. The classes (e.g. boat) are grouped in their overlying categories (e.g. vehicles).

First, objects within the same category own more similar structures than objects of different categories and, thus, expectable confusion occurs within the overlying categories. Between all categories, animals are confused the most. Except of birds, the appearance and, thus, the structure, of all animals is considerable similar, since all are four-legged animals mainly depicted in similar poses, e.g. standing on the ground. They primarily differ in their size and, therefore, animals are confused with similar sized animals (e.g. cow and horse). We show examples of falsely classified animals in figures 5.9(b), 5.9(d), 5.9(f) and 5.9(h). Furthermore, confusion also occurs in the household category due to similar structure. Sofas and chairs are often confused with each other, since both are sitting accommodations. Figures 5.9(r) and 5.9(t) show both a falsely classified sofa and chair. In addition, these figures illustrate the difficulties of the classification. The depicted object in figure 5.9(r) is rather an armchair than a chair and, thus, its structure is comparable to the structure of a sofa. In addition, the sofa in figure 5.9(t) looks similar to a wide chair or a bench. Finally, in the vehicles category, bicycles are often confused with motorbikes, since both are two-wheeled means of transport.

Second, confusion occurs due to imprecise segmentation and close objects of different classes. In the household category, for example, chairs are confused with tables. Due to

classes	Weak-MNC$_{GrabCut}$	Full-MNC	Difference
bicycle	0.2	0.2	0.0
sofa	42.7	46.7	4.0
chair	9.6	14.3	4.6
boat	36.9	41.5	4.6
tv/monitor	61.5	66.6	5.1
cat	81.5	89.1	7.6
bus	69.3	79.0	9.7
train	74.6	85.6	11.0
cow	57.1	68.8	11.7
dog	71.2	85.0	13.8
aeroplane	59.6	73.8	14.2
bottle	25.5	40.7	15.2
potted plant	13.8	29.6	15.8
bird	63.8	80.1	16.3
dining table	25.3	41.8	16.5
horse	51.4	68.8	17.4
sheep	39.9	58.0	18.1
person	43.6	64.8	21.2
motorbike	48.0	73.8	25.8
car	29.9	58.6	28.7
mean	45.3	58.3	13.0

Table 5.5: **Class-wise Instance Segmentation:** Class-wise evaluation on PascalVOC validation dataset of our Weak-MNC$_{GrabCut}$ and Full-MNC model by using mAP$^r_{0.5}$. The last column shows the difference between both models.

object usage of chairs and tables (to work on a table, we usually sit on a chair), both are often depicted close to each other. Imprecise segmentation can cause overlapping of segmentation masks and leads to confusion between classes. In addition, a lot of objects are mistaken as persons. Since persons interact with a lot of household objects and vehicles, these objects are often depicted close to persons. For example, persons ride motorbikes (see figures 5.10(j), 5.10(l) and 5.10(v)), ride bicycles (see figures 5.6(s), 5.6(x) and 5.6(ac)), ride horses (see figure 5.10(n)) or sit at a table (see figure 5.9(p)).

Class-wise Comparison

Finally, we compare the results in instance segmentation of the self-trained Full-MNC model to our Weak-MNC$_{\text{GrabCut}}$ model class-wise (see table 5.5). We both emphasize the difficulties of specific classes and compare the two models class-wise to point out in which class supervised training data would be useful.

Since MNC learns only a 28×28 segmentation mask per object, MNC struggles with fine-grained structures. According to table 5.5 the our Weak-MNC$_{\text{GrabCut}}$ and the Full-MNC model achieve similar instance segmentation results in fine-grained classes like bicycles and chairs. Figures 5.6(s), 5.6(x), 5.6(ac) and 5.9(x) show bicycles. Figure 5.9(j) shows chairs that are only partly fine-grained due to their thin legs. Potted plants also consist of fine-

grained structures like the stem (see figure 5.9(n)). Summarizing, the more fine-grained the harder to detect.

The detection of tables is also problematic (25.3% mAP). One reason for this is that tables are often only partly visible. The mostly visible part is the table surface, although a table also consists of legs. Moreover, there are often other objects standing on the table. This makes it hard to learn the structure of tables, since an empty table surface looks different to a table where plates, glasses and cutlery are lying on.

Furthermore, the detection of cars is worse than the average (29.9% mAP compared to 45.3% mAP). Figure 5.9(n) shows an example of a small car in the background. Small objects in the background are in general problematic. As we already stated when we explained the results in semantic segmentation, the effects on the IoU become significantly larger if the segmentation mask of smaller objects is too imprecise compared to larger objects. But not only the basic structure and the size of objects but also the imprecise segmentation of close objects is responsible for the segmentation results. The detection of chairs can be hard due to their structure but also due to close objects. Persons are often sitting on chairs and, thus, only parts of the chair are visible. In general, nearby objects can cause occlusions and these can reduce the detection accuracy.

If we compare the results class-wise between our self trained Full-MNC model and our Weak-MNC$_{\text{GrabCut}}$, we can conclude which classes are hard for GrabCut to segment. The difference per class is in average 13.0% and ranges from 0.0% to 28.7%. As we already stated, the detection of bicycles and chairs is difficult due to the 28×28 segmentation masks of MNC. If we compare our models in these two classes we achieve no or little improvement. However, there are classes like person, motorbike and car where we achieve more than 20% in mAP. Therefore, it would be reasonable to support the detection of these classes by some fully-supervised training data. The larger the difference between the fully-supervised model and the weakly-supervised model, the more fully-supervised training data can improve the segmentation results. Therefore, it is reasonable to invest more time and effort in the annotation of the harder classes.

5.2.2 Runtime

In figure 5.8, we compare the runtime of our end-to-end trained segmentation models with our baselines with respect to their instance segmentation results. We run both our models and our baselines on the previously described evaluation machine. Thereby the segmentation models run on the GPU. Our segmentation models not only perform better than the corresponding baselines but also the runtime per image is significantly faster. The inference of the MNC model takes ∼300 ms, no matter if we train fully-supervised or weakly-supervised. Furthermore, the standard deviation of ∼8 ms is comparably small. Our MCG baseline, for example, takes ∼20,000 ms by a standard deviation of ∼9,000 ms. Therefore, our Weak-MNC$_{\text{MCG}}$ is ∼67 times faster than the corresponding baseline and achieves ∼1.5 times better results. Our Weak-MNC$_{\text{GrabCut}}$ also achieves ∼1.1 times better results by a ∼8 times speedup compared to the corresponding baseline. As a consequence, we can train our Weak-MNC model with a precise but very slow foreground-background segmentation algorithm and, however, obtain fast inference time per image.

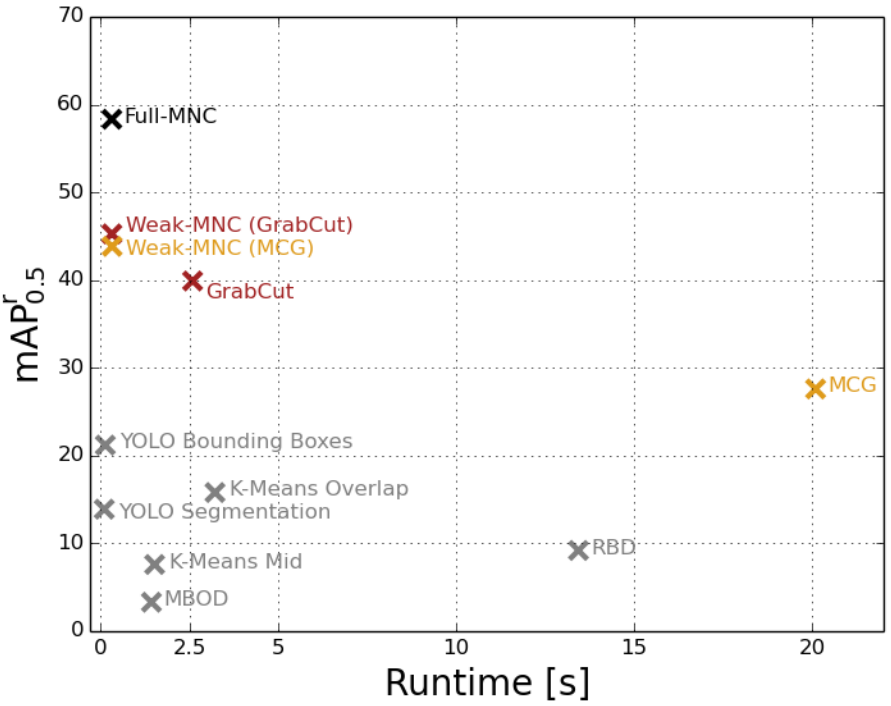


Figure 5.8: **Runtime:** Comparison of all baselines and all self trained MNC models in their runtime in contrast to their results in instance segmentation.

5.3 Comparison to Related Work

In this section, we compare our results to the results of related work in both semantic and instance segmentation. We compare not only our weakly-supervised model to the related work, but also to the self-trained fully-supervised model. This model is trained by ourself and achieves slightly different results compared to the paper of Dai et al. [DHS16] due to the random initialization. Furthermore, we compare our model to the state-of-the-art model from Khoreva et al. [KBH+17] with respect to the inference time per image.

5.3.1 Image Segmentation

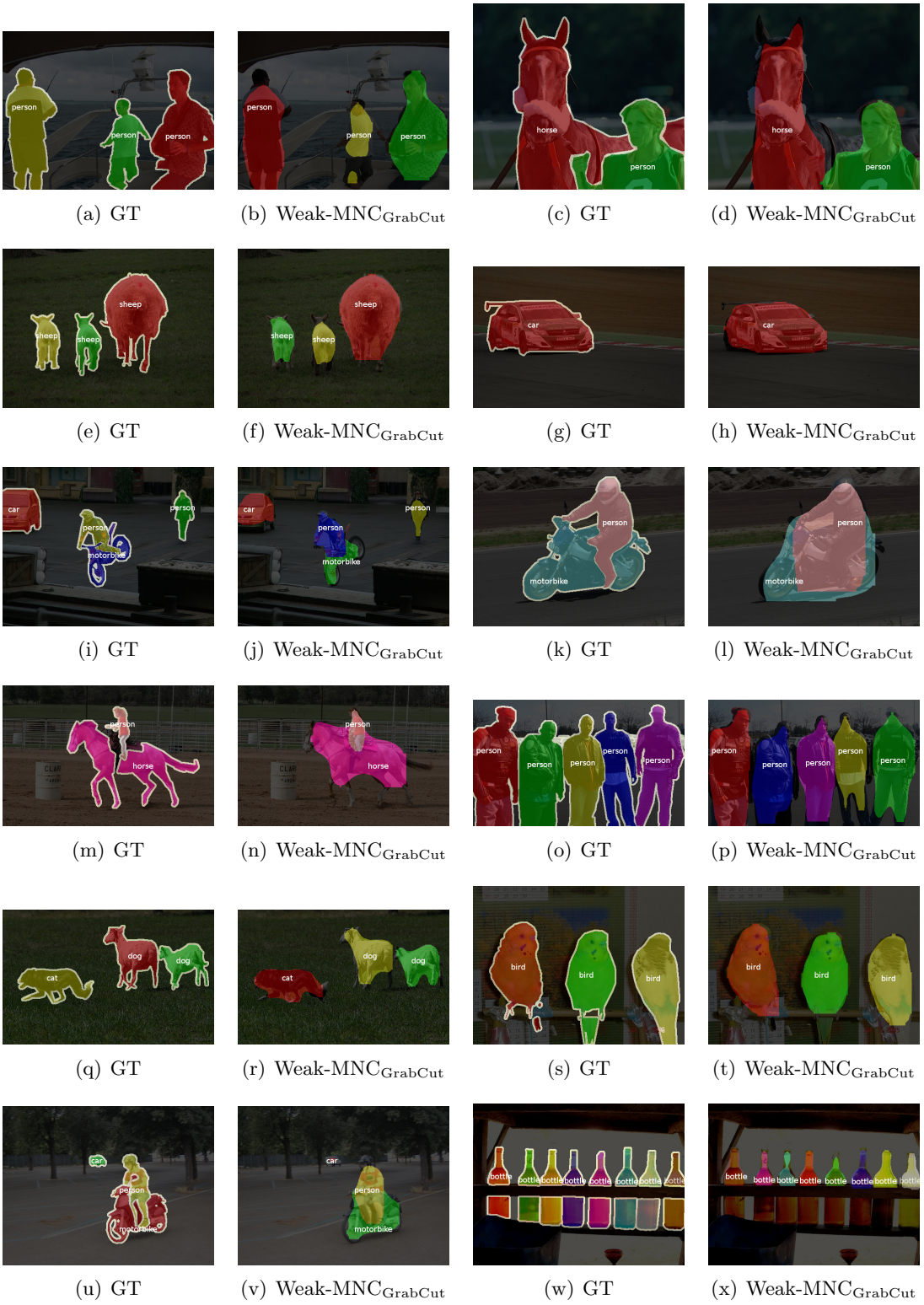
In the following, we compare our results to the results of related work, in particular segmentation methods using bounding boxes as supervision. We train and evaluate our models on different data depending on the related work to ensure comparability.

Comparison to Related Work in Weakly-Supervised Semantic Segmentation

First, we present our results in semantic segmentation in table 5.6 and compare them to the results of weakly-supervised semantic segmentation methods that also use bounding boxes as supervision.

According to table 5.6, our best weakly-supervised achieves worse mIoU results compared to all related work that uses bounding boxes as supervision. However, the self-trained Full-MNC model also achieves worse results compared to most of the related work. Thereby, the

Figure 5.9: **Instance Segmentation Results:** PascalVOC validation images segmented by our best weakly-supervised neural network (Weak-MNC$_{\text{GrabCut}}$ trained on SBD).



(a) GT (b) Weak-MNC$_{\text{GrabCut}}$ (c) GT (d) Weak-MNC$_{\text{GrabCut}}$

(e) GT (f) Weak-MNC$_{\text{GrabCut}}$ (g) GT (h) Weak-MNC$_{\text{GrabCut}}$

(i) GT (j) Weak-MNC$_{\text{GrabCut}}$ (k) GT (l) Weak-MNC$_{\text{GrabCut}}$

(m) GT (n) Weak-MNC$_{\text{GrabCut}}$ (o) GT (p) Weak-MNC$_{\text{GrabCut}}$

(q) GT (r) Weak-MNC$_{\text{GrabCut}}$ (s) GT (t) Weak-MNC$_{\text{GrabCut}}$

(u) GT (v) Weak-MNC$_{\text{GrabCut}}$ (w) GT (x) Weak-MNC$_{\text{GrabCut}}$

Figure 5.10: **Instance Segmentation Results:** PascalVOC validation images segmented by our best weakly-supervised neural network (Weak-MNC$_{\text{GrabCut}}$ trained on SBD).

Method	mIoU
BoxSup [DHS15a]	62.0
Bbox-Seg [PCMY15]	60.6
DeepLab$_{M \cap G+}$ [KBH$^+$17]	**65.7**
Our YOLO + GrabCut	62.0
Our YOLO + MCG	58.4
Full-MNC	60.7
Our Weak-MNC$_{GrabCut}$	52.7
Our Weak-MNC$_{MCG}$	53.9
Our Weak-MNC$_{GrabCut \cap MCG}$	50.8

Table 5.6: **Semantic Segmentation Results:** The results in mIoU of our two best baselines, our MNC models and the related work.

difference of our weakly-supervised model compared to the fully-supervised model is only 6.8% mIoU. In contrast, our GrabCut baseline achieve competitive results. As pointed out in section 5.2, our neural network learns only a coarse 28×28 segmentation mask per bounding box and, thus, leads to worse results in semantic segmentation compared to our baseline and the related work in semantic segmentation.

However, we focus mainly on instance segmentation and due to the small segmentation mask we achieve faster inference as we show in section 5.3.2.

Comparison to Original MNC Results in Instance Segmentation

Second, we compare our results to the original MNC results in the paper from Dai et al. [DHS16] (see table 5.7). We achieve different results when training the MNC model fully supervised due to different initialization of the weights. Similar to Dai et al. [DHS16], we trained our neural network on the SBD training set and evaluate it on the SBD validation set. Our weakly supervised model achieves 13.3% less mAP$^r_{0.5}$ and 15.6% less mAP$^r_{0.7}$ compared to the results of Dai et al. presented in the paper. However, we save annotation time since bounding boxes are ∼15 times faster to obtain than pixel-wise masks [LMB$^+$14].

Method	mAP$^r_{0.5}$	mAP$^r_{0.7}$
MNC [DHS16]	63.5	41.5
Full-MNC	65.1	45.4
Our Weak-MNC$_{GrabCut}$	48.4	25.7
Our Weak-MNC$_{MCG}$	50.2	25.9
Our Weak-MNC$_{GrabCut \cap MCG}$	42.9	23.5

Table 5.7: **Instance Segmentation Results:** Comparison of our self trained MNC models and the results of Dai et al. [DHS16]. Our fully-supervised model achieves different results due to random initialization. All networks are trained on the SBD training set and evaluated on the SBD validation set.

Supervision	Method	$mAP^r_{0.5}$	$mAP^r_{0.75}$	ABO
Full	$DeepLab_{BOX}$	47.5	20.2	51.1
	Full-MNC	<u>58.3</u>	<u>24.4</u>	<u>60.2</u>
Weak	$DeepLab_{BOX}$	44.8	<u>16.3</u>	49.1
	Our Weak-$MNC_{GrabCut}$	<u>45.3</u>	14.1	<u>49.5</u>
	Our Weak-MNC_{MCG}	43.8	11.6	48.2
	Our Weak-$MNC_{GrabCut \cap MCG}$	41.2	12.4	44.6

Table 5.8: **Instance Segmentation Results:** Comparison of our Weak-MNC models with the state-of-the-art model from Khoreva et al [KBH+17]. Our models and the state-of-the-art model are trained on SBD training set and evaluated on PascalVOC validation set.

Comparison to Related Work in Weakly-Supervised Instance Segmentation

Third, we compare the evaluation results of our weakly-supervised models to the results of the state-of-the-art network in weakly-supervised instance segmentation from Khoreva et al. [KBH+17] (see table 5.8).

We follow Khoreva et al. and train our weakly-supervised networks on the SBD training set and evaluate them on the PascalVOC validation set. We achieve better results than Khoreva et al. in $mAP^r_{0.5}$ and ABO. In $mAP^r_{0.75}$, we achieve slightly worse results. Therefore, the state-of-the-art model segments the objects more precisely than our Weak-MNC models. Due to coarse 28×28 segmentation masks, our Weak-MNC model is not able to generate very precise masks. However, increasing the mask size would also increase the training effort including an increased need of training data and, in addition, it would increase the inference time per image. Furthermore, the results imply that our generated masks are more often sufficiently accurate, otherwise Khoreva et al. would achieve also better results in the remaining metrics.

All in all, we generate less precise segmentation masks, but these masks are more often sufficiently accurate compared to the state-of-the-art model.

5.3.2 Runtime

Applications for object detection and instance segmentation are often time-critical, since they have to react to rapidly changing situations (e.g. crossing pedestrians in autonomous driving). Therefore, the inference time per image should be predictable and fast, i.e. the standard deviation should be small.

We already considered the runtime of our baselines and our MNC models in the previous sections. Figure 5.8 summarizes the runtime of all our methods (baselines and MNC models) compared to their instance segmentation results. Our MNC models are very fast and take only ∼300 ms per image. We execute our methods on the previously described evaluation machine that is equipped with an NVIDIA® GeForce GTX 1070. Moreover, our models are independent of the number of objects, i.e. our networks take ∼300 ms for an image with five objects and ∼300 ms for an image with only one object. Therefore, the standard deviation is small.

In contrast, the state-of-the-art model proposed by Khoreva et al. [KBH+17] segments objects in a two-task approach. First, this model detects objects in form of bounding boxes using Fast-RCNN [Gir15]. Second, their approach segments objects by applying a segmentation network for each bounding box. Therefore, this state-of-the-art model is not independent of the number of objects and, thus, has no predictable inference time per image. The segmentation network, called $\text{DeepMask}_{\text{BOX}}$, of Khoreva et al. is a modified DeepLab architecture proposed by Chen et al. [CPK+16]. According to the paper from Chen et al. the DeepLab takes ∼333 ms per image using an NVIDIA® TitanX GPU [CPK+16]. Therefore, the MNC model is already faster than the basic DeepMask network. In addition, Khoreva et al. apply the $\text{DeepMask}_{\text{BOX}}$ network once per bounding box and, thus, the inference time of the whole segmentation increases.

All in all, our Weak-MNC model provides predictable and fast inference time per image and independence from the number of objects. In contrast, the state-of-the-art model depends on the number of objects and, thus, exhibits a higher standard deviation.

6. Conclusion

In this chapter, we conclude with a summary of our results and compare them to the current state-of-the-art model. Furthermore, we present and discuss possible future work.

6.1 Summary

According to the annotation report of the MS COCO dataset [LMB+14], bounding boxes annotations are ∼15 times faster to obtain than pixel-wise segmentation masks. Since neural networks require a large amount of annotated training data, our weakly-supervised segmentation model decreases the annotation time compared to fully annotated pixel-wise segmentation masks. In this thesis we build upon the fully-supervised MNC model for instance segmentation of Dai et al. [DHS16] and enhance it by enabling the use of bounding boxes as weak supervision. Thereby, we use foreground-background segmentation algorithms that generate a segmentation mask within each bounding box. We use these masks to train our network instead of fully annotated masks.

Our weakly-supervised network achieves ∼13% less $\text{mAP}^r_{0.5}$ and ∼7% less mIoU compared to the fully supervised network. Moreover, it improves state-of-the art in weakly-supervised instance segmentation compared to $\text{mAP}^r_{0.5}$ and ABO (see table 5.8). We achieve worse results than the state-of-the-art model compared to $\text{mAP}^r_{0.75}$ and, thus, we generate more coarse segmentation masks. However, we segment objects more often sufficiently accurately and achieve in average better segmentation results. Our network is not able to learn more precise segmentation masks, since the MNC model learns only a 28×28 segmentation mask per bounding box. Increasing the size of this segmentation mask would also increase the inference time per image. A further benefit of our weakly-supervised network is the predictable and fast inference time per image (∼ 300 ms) and independence of the number of objects. In contrast to our network, the runtime of the state-of-the-art model strongly depends on the number of objects in the image. Moreover, the runtime of the weakly-supervised network does not depend on the segmentation mask generation time of the foreground-background segmentation algorithms.

6.2 Future Work

According to table 5.5, our weakly-supervised network achieves worse results in specific classes compared to the fully-supervised network and, thus, there is still potential for further improvements.

First, additional fully-supervised training data enhances the results in classes, where our weakly-supervised model achieves significantly worse results compared to the fully-supervised model. However, a combination of few specifically strongly-labeled and a lot of weakly-labeled training data increases the annotation effort and, thus, the model only benefits if this increase is comparably small measured against the benefits in the segmentation results.

Next, a combination of several foreground-background segmentation algorithms could increase the accuracy of the object segmentation. A possible combination is the average over the score of several segmentation algorithms. Another method is to set a pixel to foreground if at least k algorithms agree. Although a combination of several algorithms increases the segmentation mask generation time, this increase does not effect the final inference time per image.

Finally, introducing the EM scheme to instance segmentation similar to the related work in semantic segmentation has potential to improve the segmentation results. Thereby, the generated segmentation masks are learned equally to our weakly-supervised training and, afterwards, the learned segmentation masks are improved in a postprocessing step. Then, the neural network can learn these improved segmentation masks. This process is iterated according to the EM scheme. In the postprocessing step, a DenseCRF could be used to improve the segmentation masks. Moreover, the extension of the bounding box is a cue for the generation of segmentation mask. At least some pixels near the boundary should be foreground, otherwise the GT bounding box would be smaller. Therefore, the postprocessing step could assure that some pixels (or superpixels) near the boundary are foreground.

Bibliography

[APTB+14] P. Arbeláez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik, "Multiscale Combinatorial Grouping," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[ASS+12] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC Superpixels Compared to State-of-the-art Superpixel Methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[BJ01] Y. Y. Boykov and M.-P. Jolly, "Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D Images," in *Proceedings of the IEEE International Conference on Computer Vision*, 2001.

[BRFFF16] A. Bearman, O. Russakovsky, V. Ferrari, and L. Fei-Fei, "What's the Point: Semantic Segmentation with Point Supervision," in *European Conference on Computer Vision*, 2016.

[COR+16] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[CPK+16] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *International Conference on Learning Representations*, 2016.

[CV95] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, 1995.

[DDS+09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[DFK+04] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, "Clustering Large Graphs via the Singular Value Decomposition," *Machine Learning*, 2004.

[DHS15a] J. Dai, K. He, and J. Sun, "BoxSup: Exploiting Bounding Boxes to Supervise Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[DHS15b] ——, "Convolutional Feature Masking for Joint Object and Stuff Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[DHS16] ——, "Instance-aware Semantic Segmentation via Multi-task Network Cascades," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 1977.

[DSSF+16] I. N. Da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, *Artificial Neural Networks: A Practical Course*, 2016.

[DT05] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Proceedings of the IEEE Computer Vision and Pattern Recognition*, 2005.

[Du14] K.-L. Du, *Neural Networks and Statistical Learning*, ser. SpringerLink: Bücher, 2014.

[DZ13] P. Dollár and C. L. Zitnick, "Structured Forests for Fast Edge Detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013.

[EEVG+15] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: a Retrospective," *International Journal of Computer Vision*, 2015.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[Gir15] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[HAB+11] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik, "Semantic Contours from Inverse Detectors," in *Proceedings of the IEEE International Conference on Computer Vision*, 2011.

[HAGM14] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous Detection and Segmentation," in *European Conference on Computer Vision*, 2014.

[HAGM15] ——, "Hypercolumns for Object Segmentation and Fine-grained Localizatio," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[HNH15] S. Hong, H. Noh, and B. Han, "Decoupled Deep Neural Network for Semi-supervised Semantic Segmentation," in *Advances in Neural Information Processing Systems*, 2015.

[HZRS16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[Jai10] A. K. Jain, "Data Clustering: 50 Years Beyond K-Means," *Pattern Recognition Letters*, 2010.

[Jel05] F. Jelinek, "Some of my Best Friends are Linguists," *Language Resources and Evaluation*, 2005.

[Joh77] D. B. Johnson, "Efficient Algorithms for Shortest Paths in Sparse Networks," *Journal of the ACM (JACM)*, 1977.

[JSD+14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014.

[KBH+17] A. Khoreva, R. Benenson, J. Hosang, M. Hein, and B. Schiele, "Simple Does It: Weakly Supervised Instance and Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[KHH17] S. Kwak, S. Hong, and B. Han, "Weakly Supervised Semantic Segmentation Using Superpixel Pooling Network," 2017.

[KK11] P. Krähenbühl and V. Koltun, "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials," in *Advances in Neural Information Processing Systems*, 2011.

[KL16] A. Kolesnikov and C. H. Lampert, "Seed, Expand and Constrain: Three Principles for Weakly-Supervised Image Segmentation," in *European Conference on Computer Vision*, 2016.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012.

[LAE+16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *European Conference on Computer Vision*, 2016.

[LMB+14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *European Conference on Computer Vision*, 2014.

[Low99] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proceedings of the IEEE International Conference on Computer Vision*, 1999.

[LQD+17] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully Convolutional Instance-aware Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[LSD15] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[MFM04] D. R. Martin, C. C. Fowlkes, and J. Malik, "Learning to Detect Natural Image Boundaries Using Local Brightness, Color, and Texture Cues," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

[Mit97] T. M. Mitchell, *Machine Learning*, ser. McGraw-Hill Series in Computer Science, 1997.

[MLP98] O. Maron and T. Lozano-Pérez, "A Framework for Multiple-Instance Learning," in *Advances in Neural Information Processing Systems*, 1998.

[MP43] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Mervous Activity," *The Bulletin of Mathematical Biophysics*, 1943.

[NHH15] H. Noh, S. Hong, and B. Han, "Learning Deconvolution Network for Semantic Segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[P+99] J. Platt *et al.*, "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods," *Advances in Large Margin Classifiers*, 1999.

[PC15] P. O. Pinheiro and R. Collobert, "From Image-level to Pixel-level Labeling with Convolutional Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[PCD15] P. O. Pinheiro, R. Collobert, and P. Dollár, "Learning to Segment Object Candidates," in *Advances in Neural Information Processing Systems*, 2015.

[PCMY15] G. Papandreou, L.-C. Chen, K. P. Murphy, and A. L. Yuille, "Weakly- and Semi-Supervised Learning of a Deep Convolutional Network for Semantic Image Segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[PKD15] D. Pathak, P. Krahenbuhl, and T. Darrell, "Constrained Convolutional Neural Networks for Weakly Supervised Segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[PTVG15] J. Pont-Tuset and L. Van Gool, "Boosting Object Proposals: From Pascal to COCO," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[RDGF16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[RHGS15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in neural information processing systems*, 2015.

[RHW88] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Cognitive Modeling*, 1988.

[RKB04] C. Rother, V. Kolmogorov, and A. Blake, "GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts," in *ACM Transactions on Graphics (TOG)*, 2004.

[RLO+17] M. Rajchl, M. C. Lee, O. Oktay, K. Kamnitsas, J. Passerat-Palmbach, W. Bai, M. Damodaram, M. A. Rutherford, J. V. Hajnal, B. Kainz *et al.*, "DeepCut: Object Segmentation from Bounding Box Annotations using Convolutional Neural Networks," *IEEE Transactions on Medical Imaging*, 2017.

[Ros58] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review*, 1958.

[SCMS13] R. Strand, K. C. Ciesielski, F. Malmberg, and P. K. Saha, "The Minimum Barrier Distance," *Computer Vision and Image Understanding*, 2013.

[SLJ+15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[SM00] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[SZ14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *International Conference on Learning Representations*, 2014.

[UVDSGS13] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective Search for Object Recognition," *International Journal of Computer Vision*, 2013.

[VWB16] A. Veit, M. J. Wilber, and S. Belongie, "Residual Networks Behave Like Ensembles of Relatively Shallow Networks," in *Advances in Neural Information Processing Systems*, 2016.

[WLC+16] Y. Wei, X. Liang, Y. Chen, X. Shen, M.-M. Cheng, J. Feng, Y. Zhao, and S. Yan, "STC: A Simple to Complex Framework for Weakly-supervised Semantic Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[Wor15] World Health Organization, *Global Status Report on Road Safety 2015*, ser. Nonserial Publication, 2015.

[XB12] R. Xiaofeng and L. Bo, "Discriminatively Trained Sparse Code Gradients for Contour Detection," in *Advances in Neural Information Processing Systems*, 2012.

[ZF14] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *European Conference on Computer Vision*, 2014.

[ZKL+16] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[ZLWS14] W. Zhu, S. Liang, Y. Wei, and J. Sun, "Saliency Optimization from Robust Background Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[ZSL+15] J. Zhang, S. Sclaroff, Z. Lin, X. Shen, B. Price, and R. Mech, "Minimum Barrier Salient Object Detection at 80 FPS," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.