Semantic Segmentation with Deep Neural Networks

by

Gerhard Neuhold, BSc

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisors:

Prof. Dr. Horst Bischof

Dr. Samuel Schulter

Institute for Computer Graphics and Vision

Dr. Christian Leistner

Microsoft Photogrammetry

Graz, January 2016

Acknowledgements

First, I would like to express my sincere gratitude to Christian Leistner from the Microsoft Photogrammetry Team in Graz for sharing his immense knowledge in machine learning and computer vision, providing useful hints and ideas as well as answering all my questions. A special thanks also to my supervisors Prof. Horst Bischof and Samuel Schulter from the Institute for Computer Graphics and Vision for the continuous support, the fruitful discussions and all the ideas.

My sincere thanks also goes to Stefan Bernögger, Konrad Karner, Martin Ponticelli and the Microsoft Photogrammetry Team in Graz, who supported me throughout my studies and provided me the opportunity and the environment to work on such an interesting research project. In addition, I want to thank Alexey Kamenev and the TLC team for granting me access to their high performance computing cluster.

I would also like to thank my family, friends, and of course you, Beate, who all greatly supported me during my whole studies and life.

GRAZ UNIVERSITY OF TECHNOLOGY

Abstract

Faculty of Computer Science
Institute for Computer Graphics and Vision

Master's degree programme: Computer Science

Semantic Segmentation with Deep Neural Networks

by Gerhard Neuhold, BSc

Semantic segmentation is about labeling each single pixel in an image with the category it belongs to. There are several applications in a wide range of areas, like robotics, mapping or medical image analysis, in which pixel-level labels are of primary importance. In recent years, deep neural networks have shown impressive results and have become state-of-the-art for several recognition tasks. In this thesis, we investigate into the use of deep neural networks for the task of semantic image segmentation.

We adjust state-of-the-art fully convolutional networks, which are designed to label general scenes, to the task of aerial image segmentation. In addition, we transfer the learned feature representation from a large-scale image database of everyday objects for classification to pixel-wise labeling of aerial images. Further, we perform a joint training of the deep neural network and a conditional random field in an end-to-end fashion to reduce the errors that are caused by both modules. Finally, we study semi-supervised learning techniques to decrease the manual labeling effort, which is necessary to transfer learned features from pre-trained classification models.

Our proposed semantic segmentation approach is evaluated on a large-scale aerial dataset and improves the state-of-the-art accuracy. We are able to show promising results on two internal aerial datasets used by the Microsoft Photogrammetry team. Our experimental evaluation confirms that end-to-end training of the deep neural network and a conditional random field improves the overall performance. Finally, we show that incorporating unlabeled data to perform finetuning from pre-trained models decreases the manual labeling effort.

Kurzfassung

Semantische Segmentierung bezeichnet die Zuordnung jedes einzelnen Bildpunktes zur entsprechenden Objektkategorie. In verschiedensten Applikationen in einer Vielzahl von Bereichen, wie zum Beispiel Robotik, Kartografie oder medizinische Bildanalyse, ist die Klassifizierung auf Bildpunktebene von primärer Bedeutung. In den letzten Jahren haben tiefe neuronale Netze beeindruckende Ergebnisse gezeigt und sind zum Stand der Technik für unterschiedliche Erkennungsaufgaben geworden. In dieser Arbeit beschäftigen wir uns mit tiefen neuronalen Netzen und deren Anwendung für die semantische Bildsegmentierung.

Wir adaptieren Faltungsnetzwerke, welche für die Klassifizierung allgemeiner Szenen entworfen wurden, für die Luftbildsegmentierung. Des Weiteren transferieren wir die gelernte Repräsentation einer großen Bilddatenbank, bestehend aus Abbildungen von Alltagsgegenständen, um die Klassifizierung von Luftbildern durchzuführen. In einem weiteren Abschnitt bestimmen wir die freien Parameter eines tiefen neuronalen Netzes und eines grafischen Modells in einem gemeinsamen Optimierungsverfahren. Dadurch werden gezielt jene Fehler reduziert, welche aus beiden Methoden resultieren. Zuletzt werden halbüberwachte Lerntechniken näher betrachtet, damit die Datenrepräsentation vortrainierter Klassifizierungsmodelle mit weniger manuellem Aufwand auf neue Anwendungen übertragen werden kann.

Unser vorgestellter Ansatz für die automatische semantische Segmentierung von Luftbildern übertrifft die Genauigkeit bestehender Methoden. Des Weiteren können wir vielversprechende Ergebnisse auf internen Datensätzen des Microsoft Photogrammetry Teams zeigen. Unsere experimentelle Evaluierung bestätigt, dass das gemeinsame Training eines neuronalen Netzes und eines grafischen Modells das Klassifizierungsergebnis verbessert. Abschließend zeigen wir, dass die Einbeziehung nicht-markierter Daten in den Trainingsprozess den manuellen Beschriftungsaufwand reduziert respektive die Genauigkeit des Modells erhöht.

Declaration of Authorship

Contents

Acronyms

CNN Convolutional Neural Network

CRF Conditional Random Field

DNN Deep Neural Network

DSM Digital Surface Model

IoU Intersection Over Union

MAP Maximum A Posteriori

MLE Maximum Likelihood Estimation

MLP Multilayer Perceptron

NIR Near-infrared

SGD Stochastic Gradient Descent

SVM Support Vector Machine

Chapter 1

Introduction

Contents

The first chapter gives an introduction to semantic image segmentation, explains what the challenges are and presents a recipe of how to approach semantic segmentation. In addition, a short introduction to deep learning is given before current state-of-the-art methods are discussed. We conclude with a summary of our contributions and gave an outline of this thesis.

1.1 Semantic Segmentation

1.1.1 Definition and Applications

Semantic segmentation is about automatically extracting information from images. The goal is to assign a category label to each pixel in an image, or in other words, given an input image, we want to know all visible objects, where they are at pixel-level and what category they belong to. For that purpose, it is required to jointly solve localization, segmentation and classification.

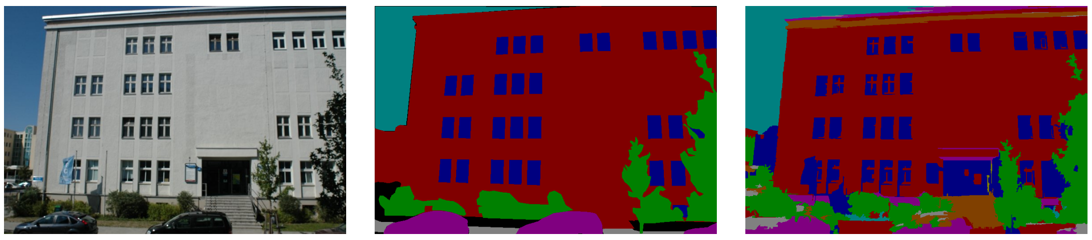

(a) Image (b) Ground Truth (c) Result

FIGURE 1.1: Example of a semantic image segmentation algorithm. Images taken from [1].

Figure 1.1 illustrates an example image and a semantic segmentation result created by the method of [1]. The objective is illustrated by the ground truth image.

There exist many applications that require semantic information from images at pixel-level, like in the fields of medical image analysis, robotics, surveillance and many more. However, automatically extracting semantic information from images is a challenging task, as we will describe in the next section.

1.1.2 The Challenges

Semantic segmentation is still an unresolved problem in computer vision. While recent methods [2–8] show reasonable results, they are still not able to beat human-level performance.

Humans are able to recognize objects with less effort, even if the objects change in view-points, scale, illumination or when they are translated or rotated. Objects can even be recognized when they are partially occluded from view. [9]

To automatically tackle these challenges using computers, state-of-the-art semantic segmentation methods rely on machine learning techniques to learn the various representations of objects from given images. Nevertheless, actual approaches have also their drawbacks. To achieve the accuracy of state-of-the-art methods, pixel-level annotated images are required, which are limited for many applications or even not available [10, 11].

In addition, if we learn the object representation from a finite set of labeled images, the model may achieve a satisfying performance on samples that look similar to those in the training set, but there is no guarantee of the algorithm to generalize well on other images. This problem is also known as the dataset-bias. [12]

For that reasons, semantic segmentation is still a hot topic in research.

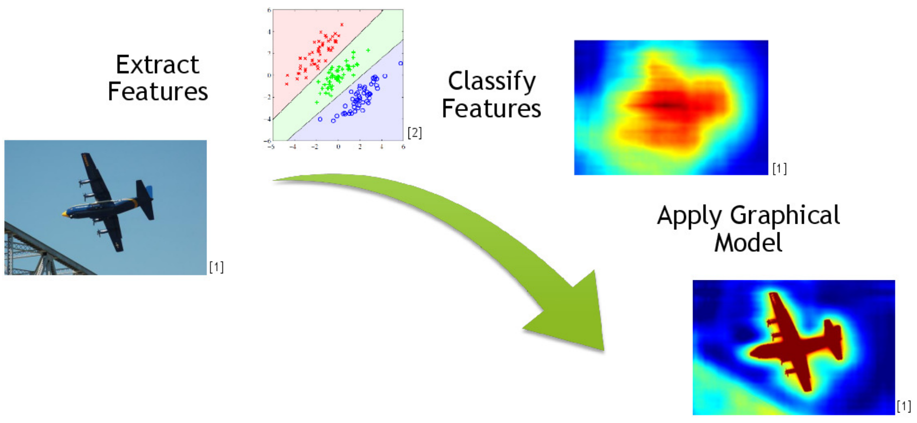


FIGURE 1.2: Recipe for semantic segmentation. Images taken from [5, 13].

1.1.3 Recipe for Semantic Segmentation

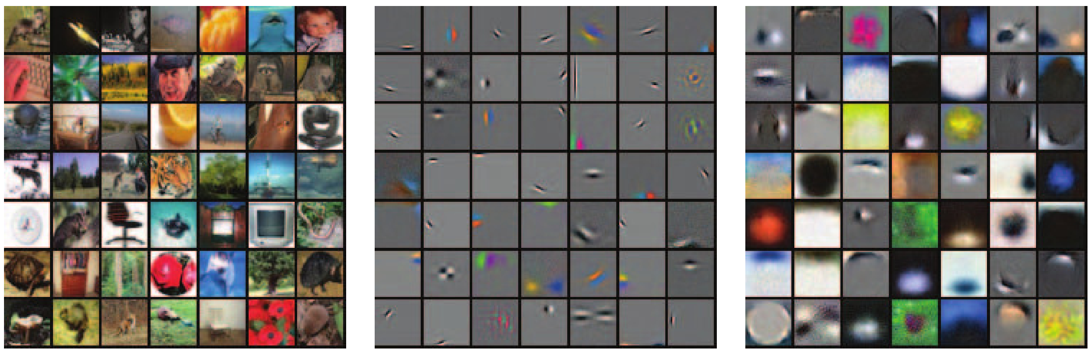
In the last section, we got an insight of the challenges to tackle. Next, we describe a common recipe for semantic segmentation algorithms which is illustrated in figure 1.2.

First, a given image is represented in a feature space. The intention of a feature space is to get a lower-dimensional representation of the original image. Second, a classifier is applied to decide which category label each pixel belongs to. Since we deal with real-world data, the classification results may be noisy or the estimated object boundaries do not coincide with the input image. Therefore, a graphical model is finally applied to counteract these inconsistencies.

In this thesis, we investigate into deep neural networks (DNNs) and conditional random fields (CRFs) to implement these steps. We now continue with an introduction of deep neural networks, before we describe the current state-of-the-art image segmentation methods.

1.2 Deep Neural Networks

In recent years, deep learning methods using neural networks have shown impressive results in many fields, like in speech recognition [14, 15], natural language processing [14, 15] or computer vision [2, 5, 14–18]. Deep neural networks are able to automatically learn low-level and high-level representations from given data [19]. There is no need for hand-crafted features, which often require expensive human efforts and expert knowledge.



(a) Input images (b) Low-Level Features (c) High-Level Features

FIGURE 1.3: Low-level and high-level representation of images using deep neural networks. Image courtesy [20].

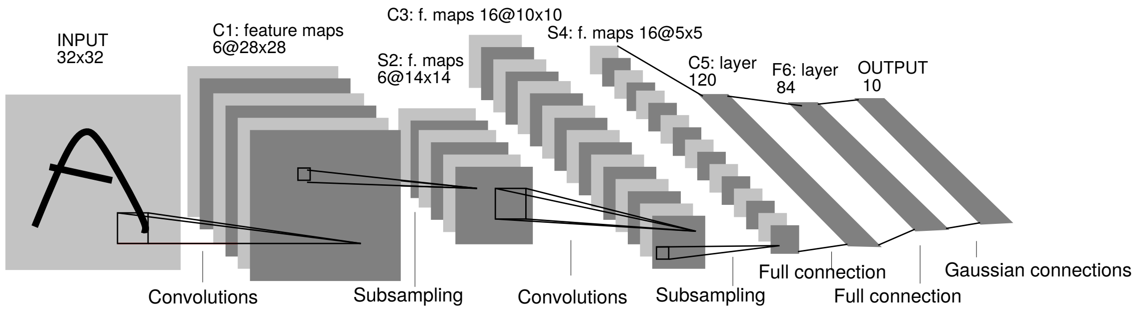


FIGURE 1.4: Network architecture of one of the first CNNs which was used to perform digits recognition. [21]

Examples of learned features are shown in figure 1.3. The visualized features demonstrate the ability of the network to capture low-level features such as edges and corners, as well as high-level features to model the complex composition of object categories. [19]

DNNs are built of several layers to represent the images at different levels of abstraction. The architecture of one of the first deep learning methods, a convolutional neural network (CNN) to recognize digits [21], is shown in figure 1.4.

The drawbacks of deep neural networks are high computational costs and the need for a huge amount of data. The raise of computational power due to graphical processing units and the availability of large-scale datasets lead deep learning methods to be state-of-the-art for several recognition task [2, 5, 16–18].

It is an interesting insight that deep models like CNNs [21], which are used to extract features at different levels of abstraction, are biologically inspired by the primary visual cortex (V1) [22], where simple and complex cells respond to lower and higher level features respectively.

CNNs are also state-of-the-art to perform semantic image segmentation. The next section describes the current best performing image labeling methods.

1.3 State-of-the-Art Image Labeling Methods

In recent years, the progress of deep learning methods improved the accuracy of image labeling algorithms too. The current best performing semantic segmentation methods [2–8] rely on CNNs. We summarize the different variants in this section.

Girshick et al. [2] and Hariharan et al. [3] use region proposals [23] to extract possible object segments and apply a classification network (Krizhevsky et al. [16]) to assign category labels to these regions.

Fully convolutional networks by Long et al. [4] make use of CNNs [21] to classify each pixel in an image, but instead of applying a CNN for each output pixel, they compute the output map more efficiently by applying the network convolutional. In addition, deconvolution layers are added on top of the network to further refine the outputs.

More recent methods combine fully convolutional networks with CRFs [24]. For example, Chen et al. [5] add a fully connected random field (Krähenbühl and Koltun [25]) on top of a fully convolutional network. Another work of Zheng and colleagues [6] shows that the mean-field approximation of a fully connected CRF can be implemented as stacks of CNN layers. The layer-wise representation is added on top of the method proposed by Long et al. [4] and end-to-end training is performed.

Since labeling images at pixel-level is an expensive task, studies [10, 11] also analyzed weakly-supervised learning setups, in which images with image-level annotations are incorporated into the training process. Other work by [7] exploits bounding box annotations to reduce the number of required pixel-level annotated images.

The previous methods described are tuned to detect everyday objects in images. In our work, we also investigate into semantic segmentation of aerial images. The methods proposed by [26, 27] rely on neural networks for aerial image labeling. However, they do not apply them convolutional. Instead, they crop patches from images, which are then classified by a neural network. Furthermore, they do not make use of large-scale datasets like ImageNet [28] to perform a pre-training [19] of the neural network.

In this section, we gave an overview of state-of-the-art image segmentation methods. In the next section, we will summarize the scientific contributions of this thesis.

1.4 Contributions of This Thesis

In this section, we summarize the main contributions of this thesis.

Deep Neural Networks for Aerial Image Labeling

As listed in the previous section, many studies have been published to label general scenes. We study how to adjust state-of-the-art semantic segmentation methods to the task of aerial image labeling. We propose a modified network architecture that is tuned to detect small-scaled objects present in low-resolution aerial images. In addition, we give evidence that the feature representation learned from images of everyday objects can be successfully transferred to the task of aerial image segmentation. Our approach is able to improve the state-of-the-art on a large-scale aerial image benchmark. Finally, we show promising results on two internal aerial image datasets used by the Microsoft Photogrammetry team.

Structured End-to-end Learning

Fully convolutional networks are trained using a loss function that does not consider dependence between adjacent pixels. For that reason, we study an approach to perform structured end-to-end training of a fully convolutional network and a CRF. In particular, we train the method proposed by Chen et al. [5] end-to-end, by using the CRF representation of Zheng and colleagues [6]. Applying a joint training of a deep neural network and a CRF allows to optimize for errors where both methods fail. Our experimental evaluation shows that structured end-to-end learning increases the performance compared to the decoupled approach by Chen and colleagues.

Semi-Supervised Transfer Learning

Labeling images at pixel-level is an expensive task and even finetuning from a pre-trained ImageNet model typically requires a large amount of labeled data. For that reasons, exploiting unlabeled data in the training process may be valuable for many applications where labeled data is scarce. We study semi-supervised learning methods to make use of the huge amount of unlabeled data and our experiments reveal that simple self-learning methods are able to increase the accuracy.

1.5 Outline

We start with related work and theory in chapter 2, which builds as a basis for the understanding of later chapters.

In chapter 3, we define a probabilistic framework for semantic segmentation and introduce fully convolutional networks. In addition, we describe the applied network architecture and how to estimate the model parameters.

Structured prediction and end-to-end learning is covered in chapter 4. We first specify the graphical model. Afterwards, we describe how to combine the CRF and the fully convolutional network and, finally, discuss end-to-end training of both modules.

Chapter 5 focuses on semi-supervised transfer learning with CNNs. An explanation why unlabeled data can help is provided and two self-learning methods are specified.

The experimental evaluations in chapter 6 reveal the qualitative and quantitative accuracy of the proposed methods. We show promising results on various aerial imagery datasets and verify that structured end-to-end learning and semi-supervised learning improves the accuracy.

Finally, in chapter 7, we summarize our findings and give an outlook for possible future research.

Chapter 2

Related Work

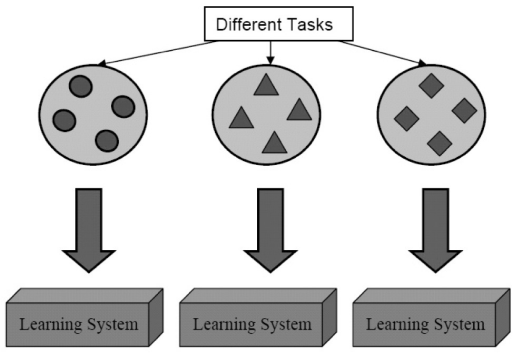
Contents

In this chapter, we describe related work, which builds as a basis for the understanding of later chapters. We start with machine learning basics and logistic regression. Afterwards, we discuss feedforward networks, convolutional neural networks (CNNs) and deep networks, which are the base of our segmentation approach.

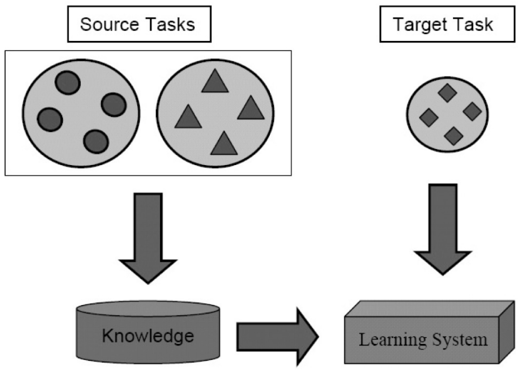
2.1 Definitions

Machine learning [13] is about automatically extracting patterns from given data to further make predictions on future data. It is used in many fields, like computer vision or speech recognition, where manually specifying data representations and carrying out predictions is impracticable.

Generally speaking, one distinguishes between **supervised** and **unsupervised machine learning**. The difference between these two lies in the existence of target labels. In the



(a) Learning Process of Traditional Machine Learning



(b) Learning Process of Transfer Learning

FIGURE 2.1: The learning processes differ between traditional machine learning and transfer learning. Transfer learning re-uses already learned features from a source task and transfers them to a target task. Images taken from [29].

case of supervised learning, the desired output values are given, whereas in unsupervised learning the targets are not available. There also exist learning methods that make use of labeled and unlabeled data, this is known as **semi-supervised learning**.

Depending on the required output of the task, further categorizations of supervised learning algorithms can be made. Typically, we differentiate between **classification** and **regression** tasks. The goal of classification is to assign a category label, whereas in regression the targets are specified via real numbers. Another important term is **structured prediction**, which involves predicting multiple variables jointly, instead of scalar discrete or real values.

Learning is the task to estimate the parameters of a machine learning model and can be further divided into **online learning** and **offline learning**. Offline learning means that all data samples can be used at once to estimate the model parameters, while online learning uses a subset of the data in each update step.

For many applications it is expensive to label data. Therefore, it is beneficial to reduce the number of required labeled samples by transferring already learned knowledge to a new target task. This is known as **transfer learning**. Figure 2.1 shows the learning processes of traditional machine learning and transfer learning. [29]

Since we are now familiar with basic machine learning concepts, we continue with introducing machine learning techniques that are used throughout this thesis. The goal of semantic segmentation is to perform a pixel-level assignment of categories, thus we make use of classification methods, in particular logistic regression.

2.2 Logistic Regression

We now introduce a classifier that is known as logistic regression [13] and describe how to perform inference as well as how to learn the model parameters. Later in this thesis we will use logistic regression to classify each single pixel of an image.

2.2.1 Inference

In the context of machine learning, inference is the task of predicting unknowns based on a given data sample and in the special case of classification, it is about assigning a category label to a given feature.

Therefore, a classifier has to make a decision about the class a feature belongs to. A feature might be an image or any other characterization that can be described in a feature space. The feature space is typically a higher-dimensional space.

Logistic regression makes use of a hyperplane in feature space that represents the decision boundary between one class and all other classes. Consequently, logistic regression is a one-vs-all classifier and to perform a multi-class decision with K classes, one hyperplane has to be specified for each class. The class separating boundary is given by

$$0 = \boldsymbol{w}_k^T \boldsymbol{\phi}(\boldsymbol{x}), \tag{2.1}$$

where the vector $\boldsymbol{x}$ represents a point in feature space, $\boldsymbol{w}_k$ are the parameters of the hyperplane separating class k from the rest, and $\boldsymbol{\phi}(\boldsymbol{x}_n)$ defines a vector of basis functions to enable a non-linear transformation of data samples into feature space. For reasons of clarity, we define w_0 to be the bias and $\phi_0(\boldsymbol{x}_n) = 1$.

Since binary classification is just a special case of multi-class classification, we herein introduce the latter one.

In real world problems it is convenient to have an uncertainty measure, which amounts to how well a given data sample belongs to a certain class. Logistic regression uses the point-plane distance a_k to quantify a specific class assignment. The further way from the hyperplane a point lies, the more likely it corresponds to that class. The point-plane distance a_k is given by the linear combination of the model parameters $\boldsymbol{w}_k$ and the basis functions $\boldsymbol{\phi}(\boldsymbol{x}_n)$.

$$a_k = \boldsymbol{w}_k^T \boldsymbol{\phi}(\boldsymbol{x}_n) \tag{2.2}$$

To make use of probability theory, for example, to estimate the model parameters using maximum likelihood, we need to specify a probability distribution for the point-plane distance. The probability of a single sample $\boldsymbol{x}_n$ belonging to class k is defined by the normalized exponential [13]

$$p(\mathcal{C}_k|\boldsymbol{x}_n, \boldsymbol{w}_k) = f_k(\boldsymbol{x}_n, \boldsymbol{w}_k) = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}, \tag{2.3}$$

which is also known as the softmax function. The exponential function and the sum in the denominator ensure the function to be a valid probability distribution.

Since we now know how to estimate the probability of a specific class given a data sample, the next step is to learn the model parameters from a given training set.

2.2.2 Parameter Estimation

For the reason that logistic regression is a probabilistic model, we can make use of maximum likelihood estimation (MLE) [13] to determine the parameters $\boldsymbol{W}$. Given a training set, MLE selects the values of the model parameters, so the likelihood $p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{W})$ of a given dataset $\{\boldsymbol{x}_n, y_n\}$ is maximized.

The data likelihood is given by

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{W}) = \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n, \boldsymbol{W}), \tag{2.4}$$

where $\boldsymbol{X}$ is a matrix of input features with columns $\boldsymbol{x}_n$ representing the single data samples, $\boldsymbol{y}$ is a vector containing the corresponding classes labels with $y_n \in \{1, \ldots, K\}$, $\boldsymbol{W}$ are the model parameters where each $\boldsymbol{w}_k$ specifies a single hyperplane, and $n \in \{1, \ldots, N\}$ denotes the index of a single data sample. The probability of a data sample n belonging to class k is then given by $p(y_n = k|\boldsymbol{x}_n, \boldsymbol{w}_k)$.

It is an important fact, that the data samples are assumed to be independent and identically distributed, thus the underlying mathematics of many statistical methods simplify.

To ease learning and inference, the class affiliation $y_n \in \{1, \ldots, K\}$ is represented by a one-of-k encoding. This means that the class labels are expressed by a binary vector $\boldsymbol{y}_n$ of length K, where the entries are constrained to be positive and to satisfy $\sum_{k=1}^{K} y_{nk} = 1$.

The data likelihood using the one-of-k encoding is specified by

$$p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{W}) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(\mathcal{C}_k|\boldsymbol{x}_n, \boldsymbol{w}_k)^{y_{nk}}. \tag{2.5}$$

The probability of a single data sample belonging to class C_k is now given by a multinomial distribution [13]. This is the reason why the multi-class variant is also known as multinomial logistic regression [13].

We now show how to apply MLE to train a logistic regression model. First, we apply the logarithm on the previously defined multinomial distribution in equation 2.5 and get the log-likelihood function:

$$\ln p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{W}) = \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln p(\mathcal{C}_k|\boldsymbol{x}_n, \boldsymbol{w}_k). \tag{2.6}$$

To get an *error measure* of how well the estimated distribution matches the given dataset, the maximization problem is turned into a minimization problem by changing the sign. To make the objective function independent of the number of data samples N, we compute the average negative log-likelihood. Consequently, the objective function to be minimized is given by

$$L(\boldsymbol{W}) = -\ln p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{W}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln p(\mathcal{C}_k|\boldsymbol{x}_n, \boldsymbol{w}_k), \tag{2.7}$$

which is also known as the cross-entropy error [13].

Since the specified loss function $L(\boldsymbol{W})$ is non-linear with respect to the unknown model parameters $\boldsymbol{W}$, a closed-form solution is not available. However, we can rely on gradient based optimization techniques [13] to set the model parameters according to

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} L(\boldsymbol{W}).$$

The gradient of $L(\boldsymbol{W})$ with respect to the parameters belonging to a certain class k is defined by

$$\nabla_{\boldsymbol{w}_k} L(\boldsymbol{W}) = \sum_{n=1}^{N} \Big(f_k(\boldsymbol{x}_n, \boldsymbol{w}_k) \; - \; y_{nk} \Big) \; \phi_n(\boldsymbol{x}_n). \tag{2.8}$$

We will discuss gradient based optimization shortly but, first, we introduce the concept of basis functions.

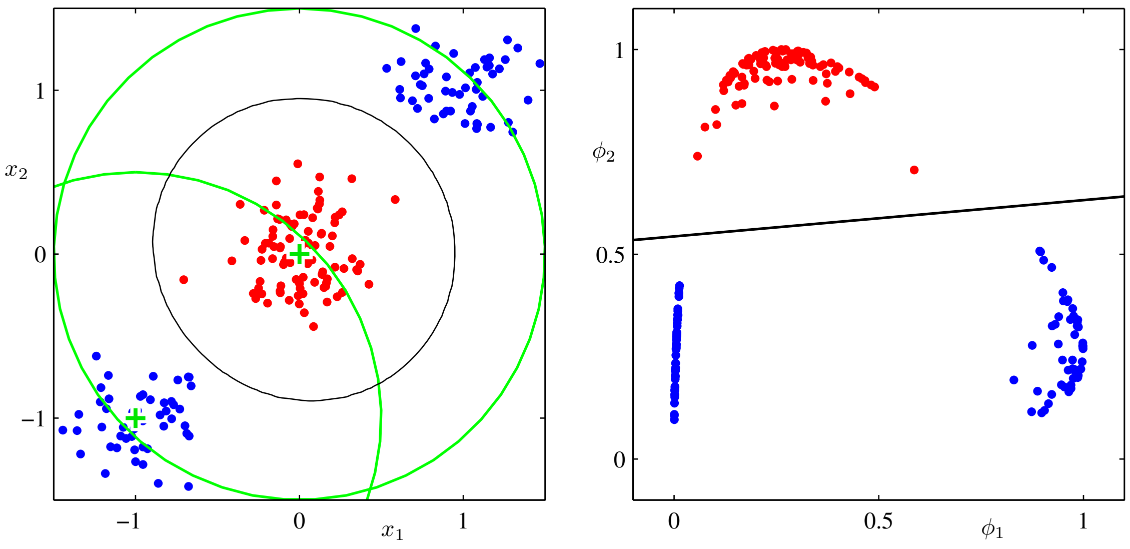


FIGURE 2.2: The left plot shows the input space $\boldsymbol{x}$ and data samples from two classes labeled red and blue. The right plot shows the same data points transformed into the features space (ϕ_1, ϕ_2) together with the linear decision boundary in black. The black curve in the left plot corresponds to the decision boundary in the input space. Image taken from [13].

In this section, we showed a detailed derivation of the logistic regression classifier from a probabilistic perspective because its understanding is important in many chapters throughout this work.

2.2.3 Basis Functions

We already mentioned that basis functions can be used to perform a non-linear transformation of input features, but we did not explain why this is helpful to perform classification.

Basis functions allow the representation of input features to be changed. This is useful if the input features are not linearly separable. By applying non-linear basis functions, the resulting decision boundary will be linear in feature space $\boldsymbol{\phi}(\boldsymbol{x})$ but non-linear in the original space $\boldsymbol{x}$. Figure 2.2 shows an example where the data samples can be separated by a linear decision boundary in feature space but not in input space. [13]

In this thesis, we model the basis functions using neural networks [13]. To be more exact, we use CNNs [21] to perform a non-linear transformation of input images into feature space.

We will now continue with optimization before we discuss neural networks in more detail.

2.3 Optimization

Optimization is about making a system as good as possible measured by an objective function. In our case, we want to optimize machine learning models, so they best fit our target task, namely pixel-level classification of images.

We will now first derive the objective function from another perspective, to see that the maximum likelihood solution of logistic regression is equal to minimizing the distance between the data and the model distribution. Later, gradient descent will be introduced, which is used to optimize the objective function. Finally, we discuss overfitting, which occurs when the model works well on the training set but not on new unseen data.

2.3.1 Objective Function

The goal of the training process is to select the model parameters, so that the produced predictions match the true class labels as close as possible.

For that, we can make use of probability theory and specify our machine learning model as the model distribution $q(\boldsymbol{x})$ and the images with according class labels as the data distribution $p(\boldsymbol{x})$. The goal can now be reformulated by minimizing the distance between the model distribution $q(\boldsymbol{x})$ and the target distribution $p(\boldsymbol{x})$. However, we do not have access to all images of the target distribution $p(\boldsymbol{x})$, but we have an approximation given by labeled data (the training set).

The Kullback-Leibler divergence $D_{KL}(p||q)$ measures the distance between two distributions and is defined in equation 2.9. The equation shows that the Kullback-Leibler divergence consists of the entropy $H(p)$ and the cross-entropy $H(p, q)$.

$$\begin{aligned} D_{KL}(p||q) &= \sum_{\boldsymbol{x}\in\boldsymbol{X}} p(\boldsymbol{x}) \ln \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \\ &= \sum_{\boldsymbol{x}\in\boldsymbol{X}} p(\boldsymbol{x}) \ln p(\boldsymbol{x}) - \sum_{\boldsymbol{x}\in\boldsymbol{X}} p(\boldsymbol{x}) \ln q(\boldsymbol{x}) \\ &= -H(p) + H(p,q) \geq 0 \end{aligned} \tag{2.9}$$

As a result of the Gibbs' inequality, the Kullback-Leibler divergence is guaranteed to be non-negative. Consequently, $D_{KL}(p||q)$ is equal to zero if and only if the probability values $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ are equal for all elements $\boldsymbol{x} \in \boldsymbol{X}$. [30]

Therefore, to learn the model parameters, the goal is to minimize the Kullback-Leibler divergence between the target distribution $p(\boldsymbol{x})$ and the learned distribution $q(\boldsymbol{x})$. Since the entropy $H(p)$ does not depend on the model parameters, it can be neglected. Only the cross-entropy $H(p, q)$ depends on the model parameters. Hence, the objective is defined by minimizing the cross-entropy $H(p, q)$, which is equal to the MLE solution of logistic regression

$$L(\boldsymbol{W}) = -\ln p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{W}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \ln p(\mathcal{C}_k|\boldsymbol{x}_n, \boldsymbol{w}_k). \tag{2.10}$$

Knowing that the loss function is defined by the difference between the model and the data distribution is an interesting insight, thus it allows us to measure how well the trained model already fits our given training and test data.

2.3.2 Gradient Descent

A simple approach to find the minimum of a given objective $L(\boldsymbol{W})$ would be to try for several values of $\boldsymbol{W}$ and then selecting the one that leads to the smallest function value of $L(\boldsymbol{W})$. However, this is very inefficient. A better approach is to make steps in the negative direction of the gradient $\nabla L(\boldsymbol{W})$ at the actual function point $\boldsymbol{W}$.

Gradient descent [13], or also known as steepest descent, can be used to find the model parameters according to

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} L(\boldsymbol{W}).$$

For that, the model parameters are iteratively updated using the following equation

$$\boldsymbol{W}^{(\tau+1)} = \boldsymbol{W}^{(\tau)} - \alpha \nabla L(\boldsymbol{W}^{(\tau)}), \tag{2.11}$$

where α represents the step size of the update step. The model weights are updated until a convergence criterion is satisfied, for example, the estimated loss $L(\boldsymbol{W})$ does not decrease anymore.

In cases where a huge amount of training data is available, performing a single update step using all samples is impossible because of memory limitations. Therefore, online learning can be employed to update the model parameters. In an online setup, the gradient $\nabla L(\boldsymbol{W})$

is defined by a subset of N training samples.

$$\nabla L(\boldsymbol{W}) = \sum_{n=1}^{N} \nabla L_n(\boldsymbol{W}) \tag{2.12}$$

If the function $L(\boldsymbol{W})$ is convex and a minimum is found, then it is guaranteed to be a unique one. However, for non-convex functions it is not guaranteed anymore that the found minimum is a global one.

A common optimization method to train CNNs is stochastic gradient descent (SGD) with momentum [31]. SGD is an online version of gradient descent where either one or several training samples are randomly selected to perform the update steps. The update rule of SGD is defined by

$$\begin{aligned} \boldsymbol{V}^{(\tau+1)} &= \mu \boldsymbol{V}^{(\tau)} - \alpha \nabla L_n(\boldsymbol{W}^{(\tau)}) \\ \boldsymbol{W}^{(\tau+1)} &= \boldsymbol{W}^{(\tau)} + \boldsymbol{V}^{(\tau+1)}, \end{aligned} \tag{2.13}$$

The momentum term $\boldsymbol{V}$ remembers previous updates and helps to escape local minima or saddle points.

We now know the objective of our problem and how to optimize our model. Nevertheless, during training, we optimize the objective using a finite training set, but the final goal is to perform predictions on new unseen data, which are not covered by the objective function. In the next section, we fill discuss overfitting, which describes the behavior when the model works well on the training set but not on unseen data.

2.3.3 Overfitting

In machine learning, we estimate the model parameters using a finite set of training samples. However, since we are interested in a model that performs predictions on future data, we need to ensure that the model generalizes well to unseen data. If a model performs well on the training set but poor on unknown samples, then overfitting occurs. Common approaches to avoid overfitting are early stopping, regularization and data augmentation. [13]

Early Stopping

To measure if the model overfits, we can split the given dataset into train and test sets. The test set is used to determine the performance on new unseen data and is not directly used to update the weights. Only the training set is used to modify the parameters. During training, we compare the classification error on the training and test and when the accuracy on the test set starts to decrease, then the training is aborted to prevent overfitting. This procedure is known as early stopping. [13]

Regularization

In addition, to avoid overfitting a regularization term [13] is usually added to the loss function. Typically, extreme parameter values are observed when overfitting occurs. Thus, the norm of the weight values can be included into the loss function, so the weights are favored to be in a reasonable range. The regularized objective to be minimized is then given by the cross-entropy and the norm of the parameters.

$$L(\boldsymbol{W}) = -\ln p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{W}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \ln p(\mathcal{C}_k|\boldsymbol{x}_n,\boldsymbol{w}_k) + \lambda\|\boldsymbol{W}\|_2^2$$

The influence of the regularization term and, consequently, the ability of the model to adapt to the training set can be controlled by the parameter λ.

Data Augmentation

Data augmentation is often employed to virtually increase the number of data samples for training. Augmenting data samples is done by randomly applying a predefined set of transformations on the training data. For example, in image classification, the data samples might be flipped or rotated randomly to increase the training set size.

2.4 Feedforward Networks

In this chapter, we discuss feedforward networks, in particular multilayer perceptrons (MLPs) which are the basis of CNNs, and show how to estimate the parameters.

2.4.1 Multilayer Perceptron

A MLP is a feedforward network [13] and consists of at least three connected layers, whereby each layer comprises multiple neurons. A single neuron is of the form as specified in equation 2.14, where $h(.)$ denotes a non-linear activation function and $\boldsymbol{\phi}(\boldsymbol{x})$ is a vector

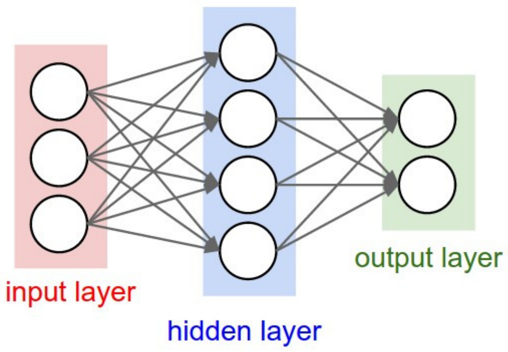


FIGURE 2.3: Network architecture of a 2-layer neural network with 3 input, 4 hidden and 2 output neurons. Image taken from [32].

of basis functions, which are weighted using the coefficients given in vector $\boldsymbol{w}$.

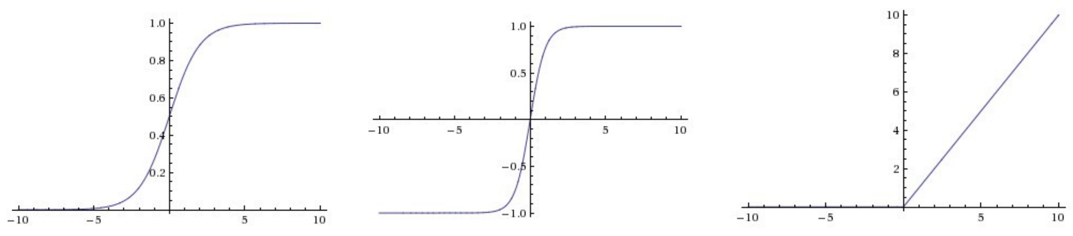
$$f(\boldsymbol{x}, \boldsymbol{w}) = h\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x}) + w_0\right) \tag{2.14}$$

We already described a model that fulfills the form of a single neuron, namely the logistic regression classifier. The difference between logistic regression and a single neuron is that neurons apply adaptive basis functions, whereas the logistic regression model makes use of fixed basis functions.

MLPs are not limited to the task of classification, it is even possible to perform a regression task by using a linear activation function in the output layer. [13]

A neural network puts together several neurons, so that each output of a neuron can be the input of another neuron. The only restriction is that loops are not allowed. Therefore, the graph which represents the connectivity, has to be directed and acyclic. An illustration of a small neural network is shown in figure 2.3. The connections reflect which neurons are the input of other neurons. The layer on the left side is called the input layer and is usually clamped to given data. The middle layer is the hidden layer and adds additional non-linearity to the neural network. The layer on the right side computes the output values of the network. Based on the fact that a single neuron in a network takes inputs of several other neurons, feedforward networks are able to apply many non-linear transformations to represent the data in feature space. [13]

Until now, we only mentioned the softmax function as a non-linear activation function, though there exist others as well, like sigmoid or hyperbolic tangent. In recent years, the rectified linear activation function [16], which is defined as $h(a_k) = max(0, \boldsymbol{x})$, is often used since it offers the required non-linearity as well as a fast computation compared to sigmoid or hyperbolic tangent. Some activation functions are illustrated in figure 2.4.



(a) Sigmoid (b) Hyperbolic Tangent (c) Rectified Linear Unit

FIGURE 2.4: This figure shows the sigmoid, hyperbolic tangent and rectified linear unit activation functions. Images taken from [32].

The non-linearity is important, because otherwise the neural network would only reflect a linear transformation of the input features, which could also be realized using a network without hidden layers.

2.4.2 Parameter Estimation

To estimate the parameters of a feedforward network, we again apply MLE and re-use the probabilistic model that we applied for logistic regression.

The difference between neural networks and linear models like logistic regression is that neural networks use adaptive basis functions, which can be learned from data. Consequently, also the gradients with respect to the parameters of the basis functions, or more specifically the incoming neurons, have to be estimated using the chain rule as follows.

$$\nabla L(\boldsymbol{W}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \left(f_k(\boldsymbol{x}^{(n)}, \boldsymbol{W}) \; - \; \boldsymbol{y}^{(n)} \right) \nabla_{\boldsymbol{W}} f_k(\boldsymbol{x}^{(n)}, \boldsymbol{W}). \tag{2.15}$$

To implement the chain rule efficiently, the gradients are computed in a specified order, namely from output layer to input layer. Because of that, the estimated gradients of the preceding layers can be re-used by successive layers. This is also known as the backpropagation algorithm. [13]

The procedure to train a feedforward neural network is given in algorithm 1. We can again apply gradient descent to estimate the parameters of the neural network.

Dropout: To reduce overfitting of feedforward networks, a method called dropout was proposed by Krizhevsky et al. [16]. Dropout is used during training to randomly set the output values of neurons to zero. For that reason, neurons cannot fully rely on the outputs of other neurons and, as a result, they have to compensate the missing neurons.

Algorithm 1 Training Feedforward Networks using Backpropagation [13]

```
Initialize weights $\boldsymbol{W}$
while not converged do
    Forward pass to compute $p(\mathcal{C}_k|\boldsymbol{X}, \boldsymbol{w}_k) = f_k(\boldsymbol{X}, \boldsymbol{w})$
    Backward pass $\nabla L(\boldsymbol{W})$ via chain rule to get the gradient for each weight
    Update parameters $\boldsymbol{W}^{(\tau+1)} \leftarrow \boldsymbol{W}^{(\tau)} - \eta \nabla L(\boldsymbol{W}^{(\tau)})$
end while
```

2.5 Convolutional Networks

A key property of images is that nearby pixels are more correlated than distant ones. CNNs [21] exploit this property and can be applied more efficiently compared to fully connected feedforward networks. We will now describe the basic concepts of CNNs and further discuss different layer types.

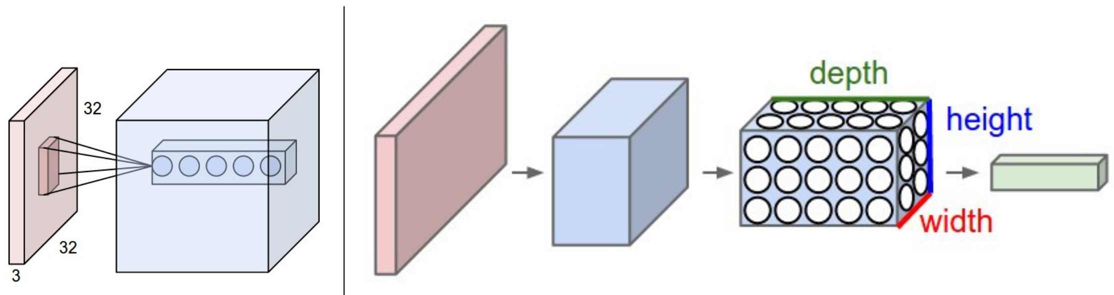
2.5.1 Convolution

The two basic concepts, namely *receptive field* and *weight sharing*, enables this type of network to be applied as convolution. We discuss them next.

Receptive Field: Standard feedforward networks do not restrict the number of input neurons and typically comprise a large number of parameters. This can be problematic with images. For example, if the input of a neural network is an image, then the network contains a weight for each single pixel. This does not scale well to large images. CNNs exploit that neighboring pixels contain more information of each other than distant ones and, thus, restrict the input neurons to smaller regions. The restricted area is called *receptive field* and is illustrated in figure 2.5(a). [32]

Weight Sharing: Another property of images is that the probability of a pattern occurring in an image is mostly independent of its position. Thus, CNNs also share the weights of the receptive fields over the whole image.

Convolution: Both concepts, *receptive fields* and *weight sharing*, lead to a more efficient model because of the reduced number of parameters. In addition, evaluating the neurons can be formulated as a convolution and computed efficiently on GPUs.



(a) Receptive Field (b) Example Architecture

Figure 2.5: Figure 2.5(a) shows the receptive field and the generated output values. Figure 2.5(b) shows a convolutional neural network with neurons arranged in three dimensions (width, height, depth). Each layer transforms a 3-dimensional blob into another 3-dimensional blob. Width and height corresponds to the dimensions of the input and depth depends on the number of feature maps. Images taken from [32].

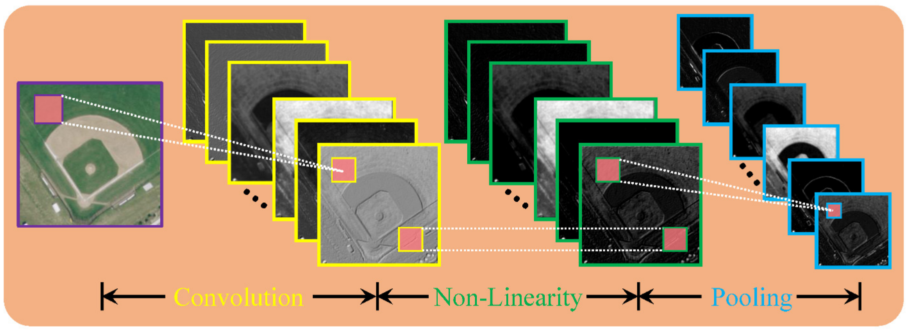


Figure 2.6: The figure shows a typical stage of a CNN in which convolution, non-linearity and pooling layers are applied sequentially. Image taken from [33].

2.5.2 Layer Types

Another important aspect of CNNs are different layer types like pooling, non-linearity and convolution, which can be stacked arbitrary on each other. Figure 2.6 illustrates a typical stage of a CNN. We will describe these types next.

Convolution

The convolution layer computes the linear combination of input neurons and model weights. If the network is applied to images, then the convolution is typically applied on a 3-dimensional input blob which creates a 2-dimensional output blob. State-of-the-art network architectures [16, 17] apply several filters per layer, so the resulting output feature map consists of three dimensions again, as shown in figure 2.5(b).

Non-Linearity

A non-linearity layer applies the activation function on the input layer. The same activation functions as already described earlier, like sigmoid, hyperbolic tangent or rectified linear unit, are typically applied.

Pooling

Pooling is applied to a restricted number of input neurons and usually a maximum, sum or average is computed over that area. In addition, a step size (also called stride) can be defined, so the resulting output map consists of less dimensions as the input layer. Pooling introduces translation invariance, allows to reduce the dimensions of the feature space and is able to decrease the computational costs.

2.6 Deep Neural Networks

In this section, we mention some other deep neural network variants and list state-of-the-art convolutional network architectures [16, 17, 34], that are commonly used in computer vision. Afterwards, we describe how present methods overcome the need of large-scale datasets.

2.6.1 Deep Neural Network Variants

We have already introduced deep learning in chapter 1 and mentioned CNNs. Furthermore, there exist also other deep learning variants, like autoencoders [35], deep Boltzmann machines [36] and deep belief networks [37].

However, taking a look at state-of-the-art image recognition methods [2, 4–6, 16], these methods are hardly observed and, therefore, are not discussed here. In this thesis, we focus on CNNs [21] with deep architectures.

2.6.2 Deep Convolutional Architectures

Actual state-of-the-art methods in image recognition [2, 4–6, 16] rely on CNNs with deep architectures, like VGG [17], AlexNet [16] or GoogleLeNet [34].

These architectures have in common that they stack different layer types, like pooling, convolution and non-linearity, but they vary in the kernel sizes, number of layers, or in the order of layers. Figure 2.7 illustrates an example of a deep architecture.

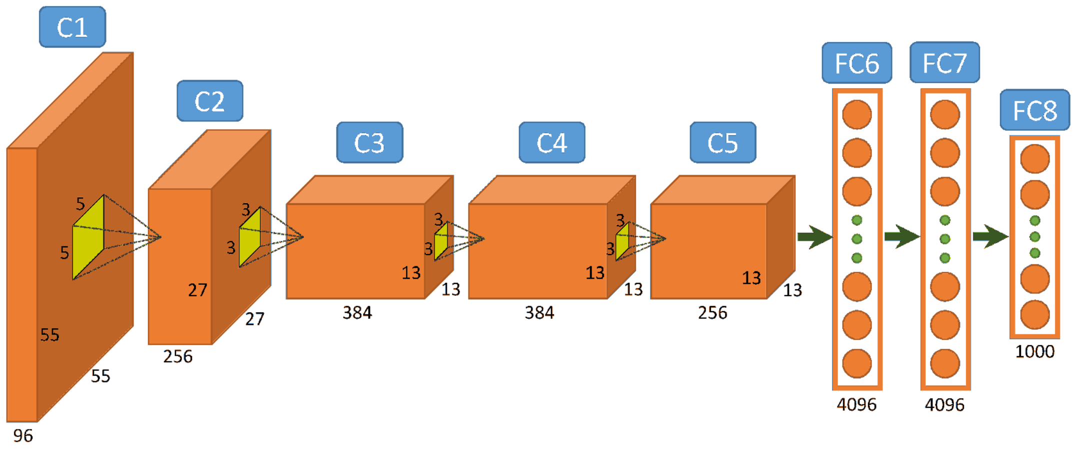


FIGURE 2.7: Illustration of the AlexNet [16]. The architecture consists of five convolutional layers with subsequent non-linearity and pooling layers, and three fully connected layers on top. Image taken from [33].

Compared to AlexNet [16], the VGG-16 architecture uses stacks of several convolution layers with smaller filter kernels, instead of single layers with large filters. This reduces the number of parameters while maintaining the same receptive field size. For example, applying three convolutions with kernels of size 3×3 leads to the same receptive field as filtering with one 7×7 kernel (assumed the feature maps are padded properly), but the overall number of parameters to estimate reduces from 49 to 27. [17]

Due to the millions of parameters these networks comprise [16, 17, 34], they have a high model complexity and are able to learn rich features [19, 38]. Nevertheless, they are also prone to overfitting, which is compensated by training on large-scale datasets like ImageNet [28].

For many applications labeled data is scarce or even not available. However, studies [19, 38, 39] showed that the learned features from large-scale datasets can be re-used for other tasks. This is achieved via transfer learning, which we will discuss next.

2.6.3 Feature Extraction and Finetuning

Learning representative features from large-scale datasets like ImageNet [28] and transferring them to other recognition tasks, like object detection or semantic segmentation, using a smaller set of training samples, is important for several machine learning applications. Transferring the learned feature representation helps to speed up the training process and to improve generalization. [19, 38, 39]

We will now discuss the different variants to perform transfer learning using pre-trained models. Pre-trained models are publicly available and do not need to be trained from scratch.

Feature Extraction

To perform feature extraction, the final layer of a pre-trained model has to be removed. The remaining layers are then used as a feature extractor, where the new top layer represents the feature vector. At training, feature vectors are extracted from all training samples and, afterwards, a linear classifier, like linear support vector machine (SVM) or Logistic Regression, is trained on that features. This method is useful if only a few labeled samples are available. [38]

Finetuning

The second method also removes the last layer of the CNN but adds a new layer that is designed for the target task (e.g. regression or classification). The resultant network architecture is then jointly retrained using labeled samples. Usually, the last layer has set a higher learning rate as the lower ones, because otherwise the rich feature representation of lower-layers might be discarded. [19]

Feature Mapping

Feature Mapping is about transfer learning if the feature types of the source and the target task differ.

Recent work [40] already successfully showed that a pre-trained ImageNet model can be finetuned to perform object detection and semantic segmentation using RGB-D images. For that, first, sets of feature channels are defined and each set is restricted to have the same number of feature channels as the source model. Like in the case of a pre-trained RGB ImageNet model, the number of channels is restricted to three. Afterwards, at training, the network is first initialized using the pre-trained RGB color model and further finetuned using the new data channels.

Even though the feature channels are different from the RGB color channels, finetuning from a pre-trained ImageNet model works better as training the network from scratch. The reason for that is because of other feature channels also contain similar low-level and high-level dependencies, like edges or higher level object correlations [40].

2.7 Chapter Summary

In this chapter, we discussed related work and theory that is relevant for the understanding of later chapters. The chapter started with an introduction of machine learning basics before the logistic regression classifier was specified in detail. This classifier is fundamental for feedforward networks because a single neuron is specified by a function form that is equivalent to the one of logistic regression. We concluded this chapter with a description of CNNs, which are a specialization of feedforward networks, and transfer learning using pre-trained ImageNet models.

Chapter 3

Semantic Segmentation with Fully Convolutional Networks

Contents

In this chapter, we begin by defining a probabilistic framework for the task of semantic segmentation, in which the probability of a label assignment for a single pixel is modeled by a convolutional neural network (CNN). Afterwards, we describe a method to apply CNNs more efficiently by transforming the architecture into a fully convolutional network. Further, we propose a modified network architecture to improve the detection accuracy for objects at small scales.

3.1 Probabilistic Framework

As we already know, the task of semantic image segmentation is to assign a category label to each pixel of a given image. We can also formulate this as a classification problem, in which we have given an image and the goal is to predict a class label for each pixel.

To perform classification, it is useful to make use of probability theory. The probabilistic view allows us to measure the uncertainty of predictions using probability values and to learn the model parameters via maximum likelihood estimation (MLE).

For that reasons, we will now specify a probability distribution $p(\boldsymbol{y}|\boldsymbol{x})$ that estimates the probability of a specific label assignment $\boldsymbol{y}$ given an input image $\boldsymbol{x}$ with M pixels.

$$p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{m=1}^{M} p(y_m|\boldsymbol{x}). \tag{3.1}$$

The probability $p(\boldsymbol{y}|\boldsymbol{x})$ represents the uncertainty of the joint label assignment and $p(y_m|\boldsymbol{x})$ corresponds to a single label assignment y_m to pixel x_m. In this thesis, we make use of CNNs to model the probability $p(y_m|\boldsymbol{x})$, in which the output units are given by a multinomial logistic regression classifier. We already introduced the logistic regression classifier in chapter 2 and showed the derivation from a probabilistic perspective. Therefore, integrating the CNN into our probabilistic framework is straightforward.

In this chapter, the neighboring class labels y_m are assumed to be independent and identically distributed. This simplifies training and inference because the joint distribution of the complete assignment factorizes into a product of the individual probabilities (as specified in equation 3.1). However, if one takes a closer look, it is obvious to see that neighboring class labels are related because adjacent pixels that have similar color intensity are also likely to be of the same class. For that reason, we study a structured approach, where the conditional independence assumption of neighboring labels is relaxed, in chapter 4.

In this section, we defined semantic segmentation from a probabilistic perspective. This allows us to measure the uncertainty of the predictions using probability values and to use MLE to estimate the model parameters.

3.2 Parameter Estimation

Since we now know how to compute the probability of a label assignment given an image, the next step is to infer the model parameters. In this section, we derive a learning objective with respect to the model weights and estimate the gradient, so we can make use of stochastic gradient descent (SGD) for optimization.

The goal of parameter estimation is to learn the unknown data distribution. The unknown data distribution is approximately given by the labeled training images. We assume that the given data samples are independent and identically distributed and, thus, the joint probability $p(\boldsymbol{Y}|\boldsymbol{X})$ of the training data is given by the product of the individual probabilities $p(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)})$ for each image $\boldsymbol{x}^{(n)}$.

$$p(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{n=1}^{N} p(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}) \tag{3.2}$$

Substituting the probabilities for each single label $y_{mk}^{(n)}$ and using the one-of-k encoding scheme leads to the joint distribution given by

$$p(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{n=1}^{N} \prod_{m=1}^{M} \prod_{k=1}^{K} p(\mathcal{C}_k|\boldsymbol{x}^{(n)})^{y_{mk}^{(n)}}, \tag{3.3}$$

where the matrix $\boldsymbol{X}$ contains the set of N input images and $\boldsymbol{Y}$ holds the corresponding labels. For simplicity, we define that all training images are of equal size M.

Given the probability distribution of the data samples, the loss function can now be specified by the cross-entropy error function, which is

$$L(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{W}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{M} \sum_{k=1}^{K} y_{mk}^{(n)} \ln f_{mk}(\boldsymbol{x}^{(n)}, \boldsymbol{W}), \tag{3.4}$$

where $y_{mk}^{(n)}$ is the target value given in the training data and $f_{mk}(\boldsymbol{x}^{(n)}, \boldsymbol{W})$ is the softmax output of the CNN for pixel m belonging to class k. We compute the average over all training instances, so the objective is independent of the number of samples.

Since we now derived the objective function of the model parameters, the next step is to compute the gradient and to define the optimization procedure.

The gradient of the objective is given by the difference of the estimated probability of our model and the ground truth, scaled by the gradient determined using the chain rule.

$$\nabla_{\boldsymbol{w}_k} L(\boldsymbol{W}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{M} \left(f_{mk}(\boldsymbol{x}^{(n)}, \boldsymbol{w}) \; - \; y_{mk}^{(n)} \right) \nabla_{\boldsymbol{w}_k} f_{mk}(\boldsymbol{x}_u, \boldsymbol{w}). \tag{3.5}$$

CNNs represent a non-linear mapping of input to output values. Therefore, a closed form solution cannot be computed, but gradient based optimization methods can be applied to minimize the objective function. In this work, we use SGD [31], which we already discussed in chapter 2. The loss-function to be optimized is non-convex too and, as a consequence, if a minimum is found, it is not guaranteed to be a global one [13]. Nevertheless, deep

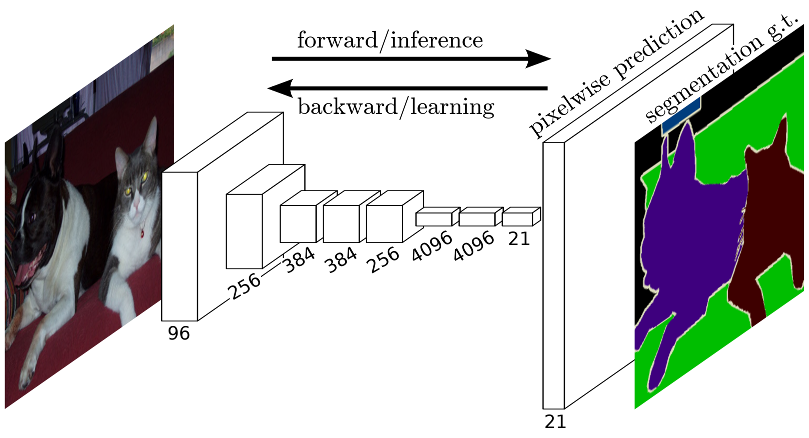


Figure 3.1: Fully convolutional networks perform end-to-end inference and learning at pixel-level. Image taken from [4].

networks are less prone to stuck in local minima than shallow ones, and usually repeating the training process several times leads to almost the same solutions [41].

In this section, we showed how to estimate the model weights using gradient based optimization. Evaluating a CNN for each single label assignment y_m can be quite computationally demanding. Therefore, we follow the approach by Long et al. [4] and convert the CNN into a fully convolutional network to speed up the computation. We will discuss this method in the next section.

3.3 Fully Convolutional Networks

A simple implementation to perform semantic segmentation with CNNs is to compute the whole network for each output pixel but this is computationally expensive. Since the receptive fields of each single output pixel overlap significantly, the CNN can be evaluated much more efficiently using fully convolutional networks, as shown by Long et al. [4].

Currently, the best performing state-of-the-art semantic image segmentation methods [5–8] rely on fully convolutional networks [4]. These networks are able to produce dense pixelwise predictions of arbitrary sized inputs and can be trained end-to-end, pixels-to-pixels. Furthermore, these networks build upon CNNs for image classification [16, 17, 34], which allows to re-use the rich feature representations from pre-trained models [19] and to further finetune them to the task of semantic image segmentation.

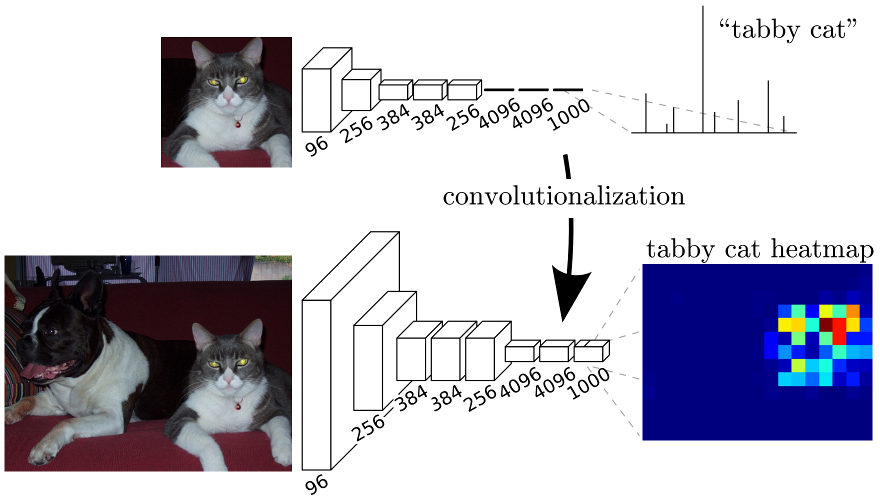


FIGURE 3.2: Transformation of a classification network into a fully convolutional network. The fully connected layers have to be replaced by convolution layers with a kernel of size 1×1. Image courtesy of [4].

Figure 3.1 illustrates end-to-end inference and learning with fully convolutional networks. The forward pass computes the per-pixel class probabilities and the backward pass computes the gradient of the loss function with respect to the model parameters. [4]

In the next section, we describe how to convert a CNN, that is designed for the task of classification, to a fully convolutional one.

3.3.1 From Classification to Dense Prediction

A simple implementation to perform semantic segmentation with CNNs is to compute the whole network for each output pixel, but this is computationally expensive. As shown by Long et al. [4], when the receptive fields overlap significantly, the forward as well as the backward passes could be applied more efficiently if the computation is performed layer-by-layer over the complete image instead of patch-by-patch.

To convert a CNN to a fully convolutional network, the fully connected layers at the top of the network have to be replaced by convolutional layers with a kernel size of 1×1. A convolutional layer with a filter size of 1×1 is equal to a fully connected layer, but the network can be applied to arbitrary sized input images. This is also represented in figure 3.2. [4]

We have now discussed how to improve the runtime by transforming a classification network into a fully convolutional network. However, these networks produce coarse output maps, which are much smaller than the input image. To resolve this issue, we discuss some methods in the next section.

3.3.2 Handling of Course Output Maps

The resultant output images of fully convolutional networks have a lower resolution than the input image. This is because of the max-pooling layers that are applied with a stride larger than one. To overcome this issue, several approaches exist. [4]

In [42] a method called shift-and-stitch is presented. They compute coarse output maps at several positions of the input image. More precisely, if the downsampling factor of the resultant predictions is equal to f, then the coarse output map is computed at every position $(x, y) \in \{0, ..., f-1\} \times \{0, ..., f-1\}$. Afterwards, the estimated predictions are interlaced to get the final class assignments.

The method proposed by [4] makes use of deconvolution to refine the predictions. The coarse output map is not just bilinear up-sampled by a fixed filter, instead, the upsampling filter weights are learned from data.

Chen et al. [5] decrease the stride in upper max-pooling layers to reduce the downsampling factor of the feature maps, but to retain the same receptive field, it is required to adapt the filter sizes of subsequent layers. For example, if the stride of a max-pooling layer gets decreased by a factor of two, then the kernel size of all subsequent layers has to be doubled. Since the reduced stride increases the computational costs, they apply the hole algorithm [43], which we will describe in the next section. Additionally, they perform a bilinear upsampling of the coarse output map and apply a graphical model [24] to align the predicted object boundaries to the edges seen in the input image.

In this work, we re-use the idea by Chen et al. [5]. This method achieves state-of-the-art accuracy while being efficient in terms of runtime and memory. In the next section, we describe the hole algorithm and in chapter 4 the graphical model.

3.3.3 Efficient Dense Evaluation

To speed up the computation in higher-level layers of the fully convolutional network, we use the hole algorithm [43], as proposed by Chen et al. [5]

The idea of the algorithm is to apply the convolutions not in a dense manner, instead, holes are incorporated to reduce the number of operations. This is done by increasing the

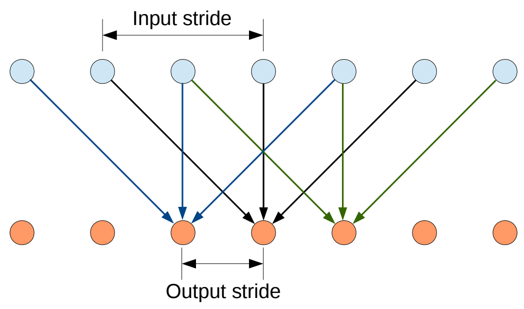


Figure 3.3: Illustration of the hole algorithm. In this example, the kernel size is equal to five, the input stride is two and the output stride is one. Image courtesy of [5].

input stride of the kernel, while the output stride remains the same. The hole algorithm is illustrated in figure 3.3.

3.3.4 Receptive Field Tuning for Small-scaled Objects

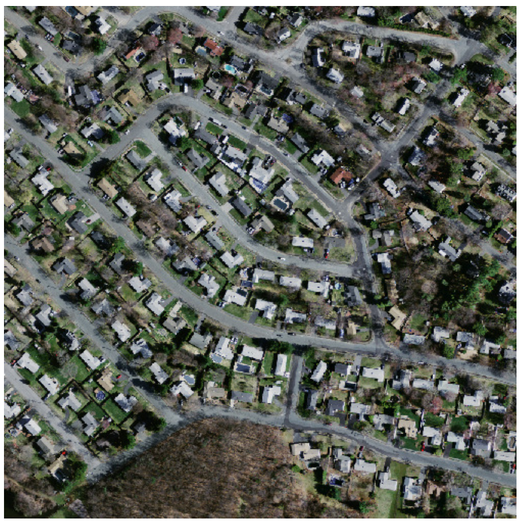
The right selection of the network architecture is crucial for a semantic segmentation algorithm because it has a significant influence to the accuracy. Recent studies achieved the best results using the VGG-16 architecture. [4, 5, 17]

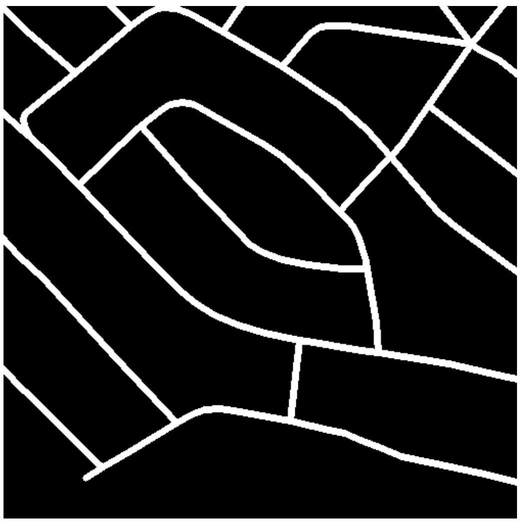
One drawback of the VGG-16 architecture is that it lacks in recognizing small-scaled objects because of the max-pooling operations. Pooling layers reduce the dimensions of the feature space, which leads to aliasing effects if the scale of the objects is too small.

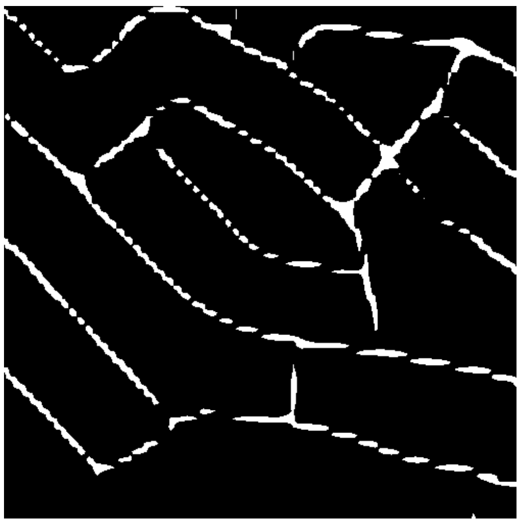
Since we are also interested in performing semantic segmentation of aerial images, in which objects may only be of a few pixels in size, we have to resolve this issue. Therefore, we adjust the DeepLab-LargeFOV [5] architecture, which is based on the VGG-16.

The two modifications are: (1) The stride of the $pool3$ is set to one. (2) The hole algorithm is applied to subsequent convolution layers from $conv4\_1$ to $fc6$.

The modified architecture reduces the downsampling factor while maintaining the receptive field and the computational efficiency. In addition, finetuning from a pre-trained image classification model is still supported. In figure 3.4, we illustrate the results using the DeepLab-LargeFOV and our modified architecture.


(a) Input image


(b) Ground truth


(c) DeepLab

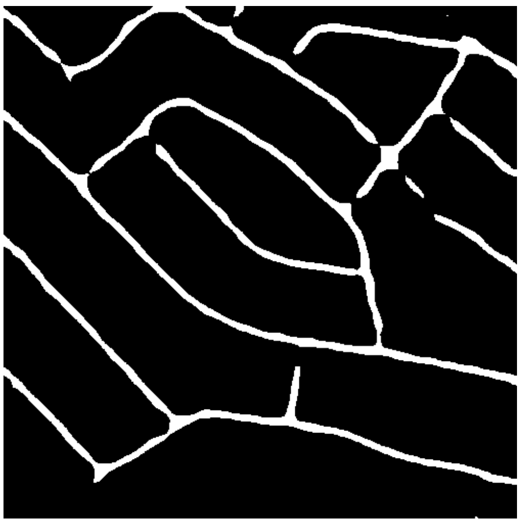

(d) Ours

FIGURE 3.4: The images illustrate the segmentation results using the DeepLab-LargeFOV network and our proposed architecture. Figure 3.4(a) and 3.4(b) show the input image and the ground truth annotation. Figure 3.4(c) shows the aliased results using the DeepLab architecture and figure 3.4(d) shows our approach with the improved detection accuracy at small-scaled objects.

3.4 Chapter Summary

This chapter started with a definition of a probabilistic framework for semantic segmentation, where we assumed neighboring pixels to be independent. The probabilistic view allows us to perform inference and learning with tools provided by probability theory.

A simple approach to perform semantic image segmentation would be to evaluate a CNN for each single output pixel, but this is computationally expensive. Therefore, we make use of fully convolutional networks to overcome this issue.

In addition, we discussed methods to deal with coarse output maps generated by fully convolutional networks. We provided a description of the hole algorithm, which allows a more efficient evaluation of the network, and described a modified version of the DeepLab architecture to reduce the aliasing effects at small-scaled objects.

Chapter 4

Structured Prediction and End-to-end Learning

Contents

We already know that neighboring pixels tend to correlate with each other. However, in the previous chapter, we assumed neighboring class labels to be independent to ease learning and inference.

In this chapter, we perform structured prediction and learning, where we relax the independence assumption of adjacent class labels. The goal of structured prediction is to infer a joint label assignment for a given image, instead of scalar individual class labels for each pixel. For that, we make use of graphical models, in particular conditional random fields (CRFs) [24], to model the dependence between pixels.

Chen et al. [5] showed that applying the fully connected CRF by Krähenbühl and Koltun [25] on top of a fully convolutional network leads to state-of-the-art accuracy. The proposed method by Chen et al. is illustrated in figure 4.1.

The fully connected CRF [25] is able to capture fine edge details as well as global dependencies. Figure 4.2 shows the confidences of the deep neural network (DNN) for the class plane and how these are sequentially refined by the CRF.

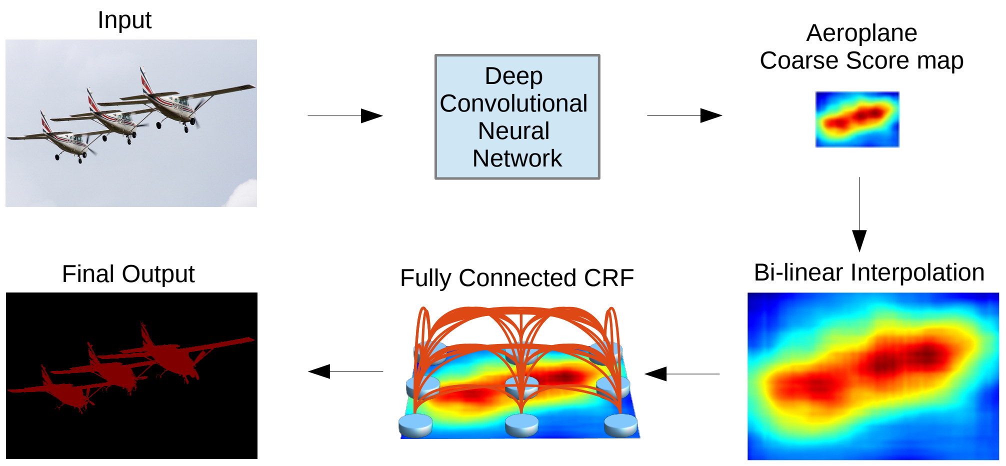


FIGURE 4.1: Structured prediction with fully convolutional networks. A given input image is passed through the neural network. Afterwards, the coarse score map has to be rescaled to match the input image size. Finally, the fully connected CRF [25] is applied to reduce alignment errors. Image courtesy of [5].

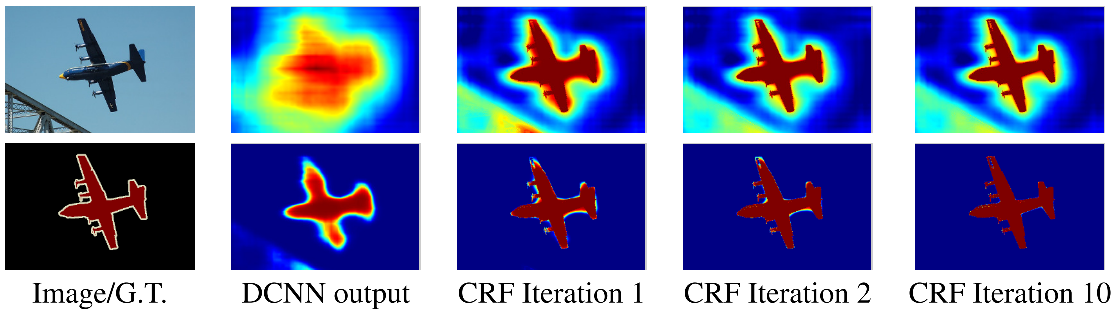


FIGURE 4.2: Illustration of the CRFs effectiveness. After each iteration the predictions are better aligned with the underlying intensity values. Image courtesy of [5].

However, Chen et al. did not investigate into end-to-end learning of both modules. Training the model jointly has the benefit that both methods can interact with each other and, hence, collectively minimize the objective function. Therefore, we propose an end-to-end training, which allows to compensate for errors that are caused by both modules.

We now continue with a description of the probabilistic model, which respects the dependency of adjacent labels, and discuss how to perform approximate inference. The last section deals with parameter estimation of the graphical model and training the joint model end-to-end.

4.1 Probabilistic Framework

In this section, we redefine our probabilistic model from chapter 3 by considering the conditional dependence of adjacent pixels. To model the dependency of nearby random variables, we make use of CRFs [24].

The probability of a label assignment $\boldsymbol{y}$ given an input image $\boldsymbol{x}$ is now defined by

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(-E(\boldsymbol{y},\boldsymbol{x}))}{\sum_{\boldsymbol{y'}} \exp(-E(\boldsymbol{y'},\boldsymbol{x}))}, \tag{4.1}$$

where the energy function $E(\boldsymbol{y},\boldsymbol{x})$ is given by the sum over unary and pairwise terms:

$$E(\boldsymbol{y},\boldsymbol{x}) = \sum_i \psi(y_i,\boldsymbol{x}) + \sum_{(i,j)\in\mathcal{E}} \psi(y_i,y_j,\boldsymbol{x}). \tag{4.2}$$

The unaries reflect the class uncertainty for each single pixel and are given by the fully convolutional network, while the pairwise terms model the dependency between connected pixels.

In this work, we follow the approach by Chen et al. [5] and model the pairwise terms as proposed by Krähenbühl and Koltun [25]. The potentials $\psi(y_i,y_j,\boldsymbol{x})$ are defined by a label compatibility function $\mu(y_i,y_j)$ and weighted feature kernels $k_m(\boldsymbol{x_i},\boldsymbol{x_j})$, as given by

$$\psi(y_i,y_j,\boldsymbol{x}) = \mu(y_i,y_j)\sum_{m=1}^{K} w_m \; k_m(\boldsymbol{x_i},\boldsymbol{x_j}). \tag{4.3}$$

The compatibility function is defined by the Potts model $\mu(y_i,y_j) = \delta[y_i \neq y_j]$ [44], which adds a penalty if the given class labels are different. The feature kernels are used to weight the class compatibility via pixel-related information, like the distance between pixels or color intensity.

The Potts model is simple and works well in practice, but it does not support a class specific compatibility transform. For example, it is more likely for class bird and sky to be close than cat and sky. For that reason, we will also explore to learn the class compatibilities from data.

The feature kernels are defined in equation 4.4 and 4.5. The first kernel is a bilateral filter [45], which considers the distance between the pixels and the intensity difference. Consequently, minimizing the energy function encourages the predictions to be better aligned with the image gradients.

$$k_1(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\frac{\|p_i - p_j\|^2}{2\sigma_\alpha^2} - \frac{\|I_i - I_j\|^2}{2\sigma_\beta^2}) \tag{4.4}$$

The second kernel, given in equation 4.5, adds a penalty if close by output variables are assigned with different class labels. Therefore, this kernel hampers the existence of small object blobs.

$$k_2(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\frac{\|p_i - p_j\|^2}{2\sigma_\gamma^2}) \tag{4.5}$$

Since neighboring pixels are not independent anymore, the joint probability of the label assignments does not factorize into the individual probabilities as in chapter 3. Although, one can easily verify that if the pairwise terms get removed in equation 4.2, the probability defined in equation 4.1 becomes again a product of softmax functions and, thus, is equal to the probability distribution as specified in equation 3.1. This also shows that logistic regression is a special form of a CRF.

4.2 Structured Prediction

Since the output variables are not independent anymore, the inference task is to predict a joint assignment of the class labels, instead of scalar ones, as in the previous chapter. This is known as structured prediction.

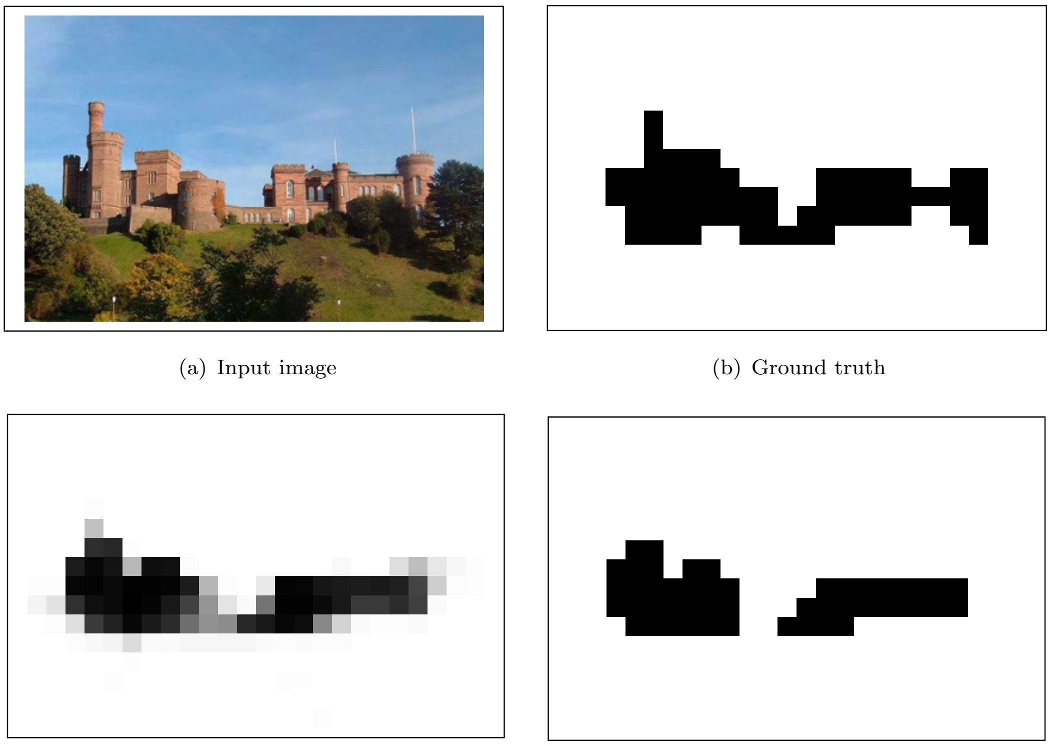
4.2.1 Inference Tasks

The goal of structured prediction in semantic image segmentation is to infer a joint label assignment $\boldsymbol{y}$ from a given image $\boldsymbol{x}$. The typical inference tasks used in computer vision are maximum a posteriori (MAP) and probabilistic inference, which are defined as follows. [46]

Maximum A Posteriori Inference

Given: *Dependency graph, parameterization, weight vector* $\boldsymbol{w}$ *and the observation* $\boldsymbol{x}$.
Goal: *Find the state* $\boldsymbol{y}^{MAP}$ *of maximum probability.*

$$\boldsymbol{y}^{MAP} = arg\,max_{\boldsymbol{y}}\; p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}). \tag{4.6}$$



(a) Input image (b) Ground truth

(c) Probabilistic Inference (d) Maximum A Posteriori Inference

FIGURE 4.3: This figure shows the difference between *Probabilistic Inference* and *Maximum A Posteriori Inference*. The task of the example provided is to classify buildings. The images 4.3(a) and 4.3(b) show the input image and the corresponding ground truth image. Image 4.3(c) shows the result using probabilistic inference, where probabilities near zero and one are colored white and black respectively. Using MAP inference gives the most probable label assignment $\boldsymbol{y}^{MAP}$, as shown in figure 4.3(d). Images taken from [46].

Probabilistic Inference

Given: *Dependency graph, parameterization, weight vector* $\boldsymbol{w}$ *and the observation* $\boldsymbol{x}$*.*

Goal: *Find the marginal distributions for each random variable.*

$$p(y_i|\boldsymbol{x}, \boldsymbol{w}) = \sum_{\boldsymbol{y}_{\setminus i}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}), \ \forall i \in \{1, \dots, N\} \tag{4.7}$$

The difference between these two methods is that the former estimates the mode of $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w})$, whereas the latter computes the marginal probability $p(y_i|\boldsymbol{x}, \boldsymbol{w})$ for each variable y_i. To see how these two methods differ qualitatively, figure 4.3 shows an example where both solutions are provided. [46]

Even though MAP provides the best solution, we are interested in a possibility to trade-off between precision and recall by varying an acceptance threshold. For that reason, we compute the marginal probabilities for each class label instead of the MAP solution.

4.2.2 Computational Complexity

Before starting with approximate inference, we shortly illustrate why exact inference is intractable for the specified CRF.

It is well known that both inference problems, MAP and probabilistic inference, for general graphs and factors are NP-hard [47]. The reason for NP-hardness is because of the sum in the denominator of equation 4.1. It consists of all possible assignments to the output space, which is exponentially in the number of output variables and, therefore, computing the partition function is even for small models intractable.

Nevertheless, there are several classes of CRFs where exact inference is possible or approximate techniques can be applied [46]. For example, graph cut methods [48] can be applied in polynomial time to solve exact inference if the random variables are binary and the pairwise terms satisfy the submodularity condition

$$\psi(y_i, y_j, \boldsymbol{x}) + \psi(y_j, y_i, \boldsymbol{x}) \leq \psi(y_i, y_i, \boldsymbol{x}) + \psi(y_j, y_j, \boldsymbol{x}). \tag{4.8}$$

Because we are interested in the marginal probabilities rather than exact inference, we make use of the mean-field approximation, as described next.

4.2.3 Mean-Field Approximation

Since inferring the exact marginal probabilities of the defined model is intractable, we have to resort to an approximate inference method. Usually, variational or stochastic inference methods are used, where the former approximates the structure of the probabilistic model and the latter exploits sampling to compute the marginal probability values [24].

In this work, we use a variational method called the mean-field approximation [49, 50] because it can be applied very efficiently to perform inference in fully connected models, as proposed by Krähenbühl and Koltun [25], while still achieving state-of-the-art accuracy, as shown by Chen et al. [5].

The mean-field approximation finds a fully-factored distribution $q(\boldsymbol{y}|\boldsymbol{x}) = \prod_i q_i(y_i|\boldsymbol{x})$, that is closest to the true distribution $p(\boldsymbol{y}|\boldsymbol{x})$ with respect to the Kullback-Leibler divergence [30]. As a result, the mean-field approximation can be formulated as an optimization task, where the goal is specified as

$$q(\boldsymbol{y}|\boldsymbol{x}) = \arg\min_{q^*} D_{KL}(q^*||p) \quad \text{s.t.} \quad q^*(\boldsymbol{y}|\boldsymbol{x}) = \prod_i q_i^*(y_i|\boldsymbol{x}). \tag{4.9}$$

As already discussed in chapter 2 in the context of learning probability distributions, the Kullback-Leibler divergence [30] measures the distance between two given probability distributions. To perform the mean-field approximation, the Kullback-Leibler divergence is used to measure the distance between the intractable distribution $p(\boldsymbol{y}|\boldsymbol{x})$ and the approximate distribution $q(\boldsymbol{y}|\boldsymbol{x})$.

Minimizing the Kullback-Leibler divergence with respect to $q^*(\boldsymbol{y}|\boldsymbol{x})$ while constraining $q^*(\boldsymbol{y}|\boldsymbol{x})$ and all $q_i^*(y_i|\boldsymbol{x})$ to be valid distributions gives the update formula as specified in equation 4.10. [25]

$$q_i(y_i|\boldsymbol{x}) = \frac{1}{Z_i} \exp\left\{ -\psi(y_i, \boldsymbol{x}) - \sum_{y' \in \mathcal{Y}} \mu(y_i, y') \sum_{m=1}^{K} w_m \sum_{j \neq i} k_m(\boldsymbol{x_i}, \boldsymbol{x_j}) q_j(y'|\boldsymbol{x}) \right\} \tag{4.10}$$

This update equation is used in the mean-field inference method given in algorithm 2.

Algorithm 2 Mean-field inference algorithm for fully connected CRFs [6, 25]

```
$q_i(y_i|\boldsymbol{x}) \leftarrow \frac{1}{Z_i}\exp\{-\psi(y_i,\boldsymbol{x})\}$ for all $i$          ▷ Initialization
while not converged do
    $\tilde{q}_{im}(y_i|\boldsymbol{x}) \leftarrow \sum_{j\neq i} k_m(\boldsymbol{x_i},\boldsymbol{x_j}) q_j(y_j|\boldsymbol{x})$ for all $m$          ▷ Message Passing
    $\check{q}_i(y_i|\boldsymbol{x}) \leftarrow \sum_{m=1}^{K} w_m \tilde{q}_{im}(y_i|\boldsymbol{x})$          ▷ Weighting Filter Outputs
    $\hat{q}_i(y_i|\boldsymbol{x}) \leftarrow \sum_{y'\in\mathcal{Y}} \mu(y_i,y') \check{q}_i(y'|\boldsymbol{x})$          ▷ Compatibility Transform
    $\breve{q}_i(y_i|\boldsymbol{x}) \leftarrow \psi(y_i,\boldsymbol{x}) + \hat{q}_i(y_i|\boldsymbol{x})$          ▷ Adding Unary Potentials
    $q_i(y_i|\boldsymbol{x}) \leftarrow \frac{1}{Z_i}\exp\{-\breve{q}_i(y_i|\boldsymbol{x})\}$          ▷ Normalization
end while
```

Each iteration of the mean-field algorithm consists of a message passing step, a compatibility transformation and an update of the local probability value. The local update and the compatibility transformation run in linear time. Only the message passing step has a quadratic runtime in the number of output variables because for each variable a weighted sum over all other variables has to be computed. However, message passing can also be performed by approximate high-dimensional filtering [25, 51] to reduce the computational costs.

4.2.4 Efficient Message Passing

Since message passing requires the computation of a sum for each variable over all other variables, the computational complexity is quadratic. To make message passing more efficient, we use the idea of Krähenbühl and Koltun [25]. They show that message passing in a fully connected network can be reformulated as a convolution and, in the special case where Gaussian feature kernels are used, they propose to use the permutohedral lattice [51] to perform an efficient approximation in linear time.

We now describe why message passing in the specified fully connected model can be seen as a convolution and, afterwards, we give more details about the permutohedral lattice.

The following equation shows the message passing step from algorithm 2 rewritten as a convolution.

$$\begin{aligned}\tilde{q}_{im}(y_i|\boldsymbol{x}) &= \sum_{j\neq i} k_m(\boldsymbol{x_i}, \boldsymbol{x_j}) q_j(y_j|\boldsymbol{x}) \\ &= \sum_{j} k_m(\boldsymbol{x_i}, \boldsymbol{x_j}) q_j(y_j|\boldsymbol{x}) - q_i(y_i|\boldsymbol{x}) \\ &= \underbrace{(k_m * q(y|\boldsymbol{x}))[\boldsymbol{x_i}]}_{\bar{q}_{im}(y_i|\boldsymbol{x})} - q_i(y_i|\boldsymbol{x})\end{aligned} \tag{4.11}$$

The subtraction of $q_i(y_i|\boldsymbol{x})$ is required since the convolution sums over all random variables, whereas message passing does not include $q_i(y_i|\boldsymbol{x})$.

As already mentioned, the permutohedral lattice [51] is used to perform the convolution in linear time. The permutohedral lattice is a general method to rapidly solve equations of the form

$$\boldsymbol{v}_i = \sum_j \exp(-\frac{1}{2}\|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2)\boldsymbol{u}_j. \tag{4.12}$$

It is easy to verify that this a more general form of the previously defined message passing procedure, as shown in the following equation

$$\begin{aligned}\bar{q}_{im}(y_i|\boldsymbol{x}) &= \sum_{j} k_m(\boldsymbol{x_i}, \boldsymbol{x_j}) q_j(y_j|\boldsymbol{x}) \\ &= \sum_j \exp(-\frac{1}{2}\|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2) q_j(y_j|\boldsymbol{x}).\end{aligned} \tag{4.13}$$

The processing steps of the permutohedral lattice to apply the high-dimensional Gaussian filter can be summarized as follows. First, the input values $\boldsymbol{x}$ at positions $\boldsymbol{p}$ are embedded

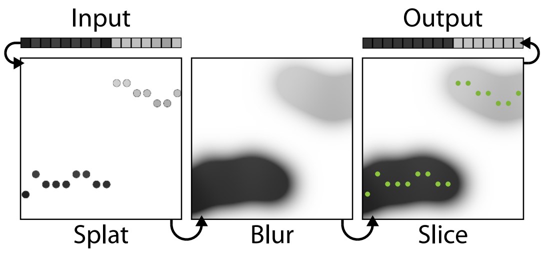


FIGURE 4.4: The figure illustrates the three steps of the permutohedral lattice to perform Gaussian filtering in features space. Image taken from [51].

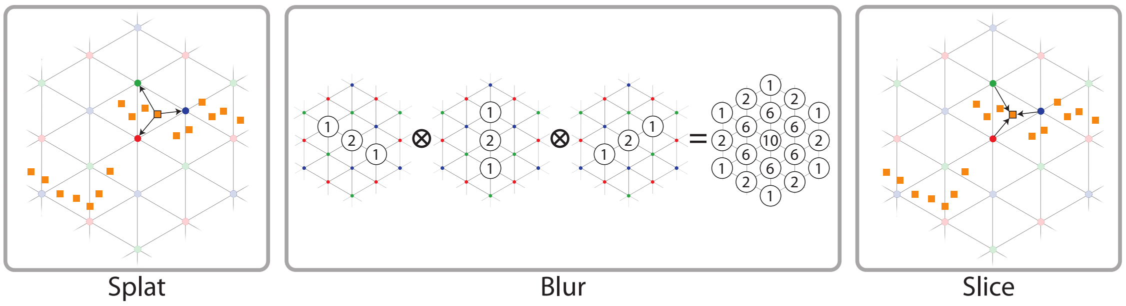


FIGURE 4.5: A high dimensional data structure is used to perform splatting, blurring and slicing. In the splatting stage the position vectors are assigned to the predefined vertices. The predefined vertices are then blurred using separable filters. At slicing, the blurred values at the fixed points are used to interpolate the values belonging to the output positions. Image taken from [51].

in feature space (splatting). Then, the values are blurred in feature space and, finally, the output values are interpolated from the blurred feature space using the original positions $\boldsymbol{p}$ (slicing). Figure 4.4 shows an example of a bilateral filter applied on a one-dimensional input feature. [51]

Furthermore, the permutohedral lattice uses an efficient data structure to perform Gaussian filtering. The data structure tiles the feature space with simplices along $d+1$ axis. The filter values are computed at the vertices of the simplices and interpolated along the edges in between. An illustration of the high dimensional data structure is shown in figure 4.5. [51]

In addition, the separability of unit variance Gaussian kernels is exploited. Applying one d-dimensional Gaussian filter kernel with unit variance is equivalent to make use of d one-dimensional filters, but the runtime of applying a single one-dimensional filter reduces to linear in the number of variables. To get unit variance kernels, a data whitening operation is first performed in feature space. [51]

Because of using the permutohedral lattice, the approximate message passing is highly efficient, even without using a GPU. In this work, we re-use the publicly available CPU

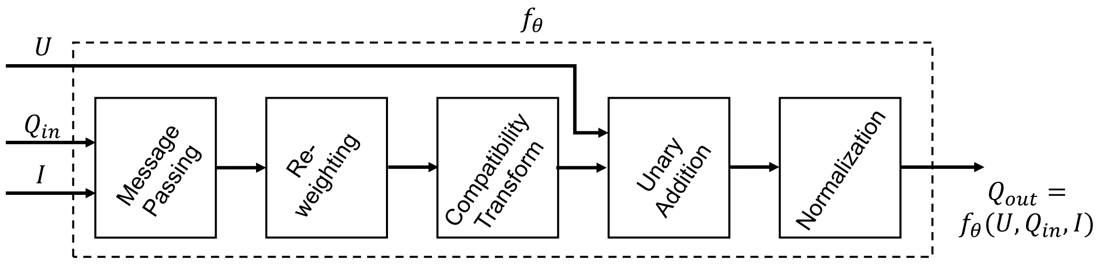


FIGURE 4.6: Figure illustrates the mean-field approximation as a stack of CNN layers. Image taken from [6].

code [51], but to speed up the mean-field iterations and to reduce the amount of data that has to be copied between CPU and GPU memory, we implemented a GPU version of the splatting, blurring and slicing operations.

4.2.5 Layer-wise Representation

In this section, we take a closer look at the implementation of the mean-field approximation. The mean-field approximation given in algorithm 2 can mostly be implemented using standard layers of a convolutional neural network (CNN). Figure 4.6 shows a graphical representation of the CNN layer stack. [6].

The initialization of the mean-field algorithm is implemented by a softmax layer. Message passing is implemented by a specific layer, which applies the efficient approximation described in the previous section. Weighting the filter outputs is achieved by a convolution layer with a kernel of size 1×1, M input channels, where M is the number of feature kernels, and one output channel for each class. It is beneficial to use one kernel weight per category since each kernel performs differently depending on the class. Applying the compatibility transform is done by another convolutional layer with a kernel of size 1×1, where the number of input and output channels are equal to L. The subtraction of unary and pairwise potentials is represented by a simple element-wise addition layer. Finally, the normalization is another softmax layer. [6].

One stack of CNN layers represents a single iteration of the mean-field inference algorithm. Therefore, to perform several iterations, multiple stacks have to be added on top of each other.

Implementing the mean-field inference as a stack of CNN layers allows to use a deep learning framework to implement the CRF. As for fully convolutional networks, we use the popular deep learning framework Caffe [52] to implement the mean-field approximation as a stack of CNN layers.

The last sections covered the inference task of the structured model. The next step is to learn the parameters of the joint model.

4.3 Structured End-to-end Learning

The structured model, described in earlier sections, consists of several weights, which have to be inferred from the training data. In this section, the estimation of these parameters is discussed. In addition, the parameters of the CRF and CNN are trained jointly in an end-to-end manner to collectively optimize the objective function. End-to-end training of the deep neural network and the CRF allows to optimize for errors where both methods fail.

The trainable parameters $\boldsymbol{W}$ of our specified model include the filter weights of the Gaussian kernels, the class compatibilities and the neural network parameters.

As already known, the probability of a specific label assignment $\boldsymbol{y}$ given an image $\boldsymbol{x}$ using the structured model is defined by

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(-E(\boldsymbol{y},\boldsymbol{x}))}{\sum_{\boldsymbol{y'}} \exp(-E(\boldsymbol{y'},\boldsymbol{x}))}. \tag{4.14}$$

Again, we assume the training images to be identically and independent distributed. Therefore, the joint distribution of all images is given by

$$p(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{n=1}^{N} p(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}). \tag{4.15}$$

To estimate the parameters, we apply the maximum likelihood estimation (MLE) method and compute the expected loss of the training samples, so that the objective does not depend on the dataset size. Hence, we get the loss function, which is defined as follows.

$$\begin{aligned} L(\boldsymbol{W}) &= -\ln p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{W}) \\ &= -\frac{1}{N}\sum_{n=1}^{N} \ln p(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}) \\ &= -\frac{1}{N}\sum_{n=1}^{N} \left(-E(\boldsymbol{y}^{(n)},\boldsymbol{x}^{(n)}) - \ln \sum_{\boldsymbol{y'}} \exp(-E(\boldsymbol{y'},\boldsymbol{x}^{(\boldsymbol{n})}))\right) \end{aligned} \tag{4.16}$$

The parameters that best fit the given training samples are given by

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \; L(\boldsymbol{W}).$$

Since $L(\boldsymbol{W})$ is a non-linear function as in the case of the unstructured model, we cannot get a closed form solution. However, we can resort to gradient based optimization. The gradient with respect to the parameters is specified as

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{W}) = \frac{1}{N} \sum_{n=1}^{N} \left(p(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}) \; - \; \boldsymbol{y}^{(n)} \right) \nabla_{\boldsymbol{w}} p(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)}). \tag{4.17}$$

Because the calculation of the gradient requires to infer the class probabilities $p(\boldsymbol{y}^{(n)}|\boldsymbol{x}^{(n)})$ for each training image, the exact computation of the gradient is intractable for the given model. Nevertheless, we can again apply the same mean-field approximation method as earlier to make the gradient estimation tractable.

Since we now know the gradient of the structured model, we can make use of an optimization procedure to estimate the parameters.

To perform backpropagation of the gradients with respect to the parameters of the joint CRF and CNN model, we follow the proposed method by Zheng et al. [6], where the CRF iterations are represented via a stack of CNN layers.

Computing the gradients of the mean-field algorithm steps that are represented by common neural network layers, like convolution or softmax, is done as usual. The only difference has to be employed in the message passing layer, namely to get the gradients with respect to the inputs of the Gaussian filter kernels, the blur kernels have to be applied in reverse order. [6]

Since the gradients and the parameter updates are computed individually for each CRF iteration, the parameters of the CRF have to be averaged after the parameter update, so that all CNN layer stacks have the same weights again. [6]

To train the whole system end-to-end, the gradients of each CRF iteration with respect to the parameters of the fully convolutional network have to be passed back to the CNN. This is straightforward because the CRF layers that directly depend on the CNN outputs are represented using standard convolutional network layers and, thus, backpropagation can also be performed as usual.

In this section, we described how to estimate the parameters of the CRF and how to train the whole system end-to-end. Using the fact that the mean-field iterations can be seen as a stack of CNN layers simplifies the parameter estimation of the CRF. Common deep neural network layers can be re-used to perform the required forward and backward passes. The only required modifications are the shared network parameters across the mean-field layer stacks and the reverse applied blur kernels in the message passing layer.

4.4 Chapter Summary

This chapter started with a redefinition of the probabilistic framework defined in chapter 3 to consider the dependence of nearby pixels.

Afterwards, we specified the CRF that is used to model the neighbor relationships and described how to add the CRF on top of the deep neural network. Since exact inference of the specified model is intractable, we applied the mean-field approximation and the permutohedral lattice to perform inference in linear time. Finally, we showed how to train the parameters of the graphical model and the deep neural network end-to-end. For that, we represent the mean-field algorithm as a stack of CNN layers, which allows us to perform backpropagation as usual.

Chapter 5

Incorporating Unlabeled Data

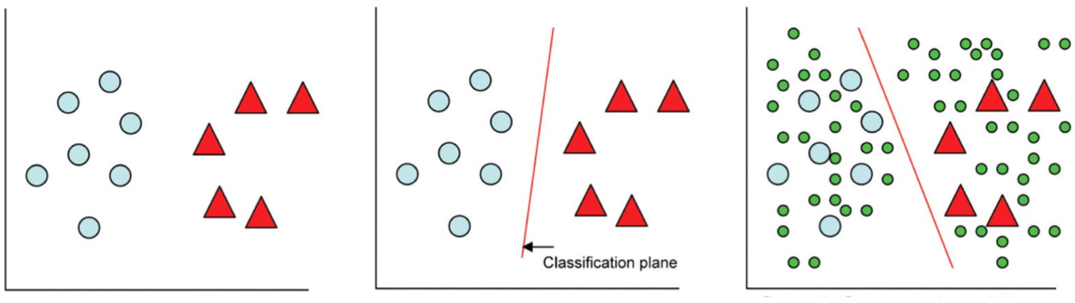
Contents

As we already know, in many applications convolutional neural networks (CNNs) are first pre-trained on the large-scale ImageNet dataset and then finetuned to a specific task of interest. However, finetuning still requires a large number of labeled samples.

In this chapter, we want to address this issue by extending the finetuning process of fully convolutional networks to make use of unlabeled data.

Since there are no studies published that rely on state-of-the-art semantic segmentation methods in a semi-supervised setup, we explore self-learning to perform transfer learning using labeled and unlabeled examples.

We start this chapter by discussing why unlabeled data helps, continue with a survey of general semi-supervised learning approaches and, finally, introduce two semi-supervised transfer learning approaches for the task of semantic segmentation.



(a) Labeled Data (b) Supervised Learning (c) Semi-Supervised Learning

FIGURE 5.1: Illustration of how unlabeled data can help. Blue circles and red triangles represent labeled samples, whereas green circles are unlabeled samples. Image courtesy of [53].

5.1 How Can Unlabeled Data Help?

Before we start with an overview of existing semi-supervised learning methods, we discuss the usefulness of unlabeled data.

Figure 5.1 shows three charts. The first contains the labeled training data, the second one illustrates the estimated decision boundary with labeled data only and the third depicts the improved decision boundary using additional unlabeled data. This example demonstrates that unlabeled samples can be used to estimate the low density between classes and, hence, to arrange the classification plane, so it better fits the given data samples. [53]

It is not guaranteed that incorporating unlabeled data always helps. In a more mathematical formulation one could say that the information in $p(\boldsymbol{x})$ obtained by unlabeled data has to contain information in the inference of $p(\boldsymbol{y}|\boldsymbol{x})$. [54]

To include unlabeled data, assumptions about the underlying data distribution have to be made. For example, samples of the same class are assumed to be in the same cluster, or the area between the classes to separate has a low density. If these assumptions do not hold, unlabeled data may not help or even degenerate the accuracy of the learned model. [54, 55]

Smoothness Assumption: "*If two points x_1, x_2 in a high-density region are close, then so should be the corresponding outputs y_1, y_2.*" [54]

In supervised learning it is assumed that the output varies smoothly with the distance. In the semi-supervised setting the density of the inputs is considered too. The semi-supervised smoothness assumption applies to both classification and regression.

Cluster Assumption: "*If points are in the same cluster, they are likely to be of the same class.*" [54]

If one knows that the points of each class tend to form a cluster, then the unlabeled data could provide useful information about the boundaries of each cluster. This assumption could also be formulated as that the decision boundary between two classes should lie in a low-density area.

Manifold Assumption: "*The (high-dimensional) data lie (roughly) on a low-dimensional manifold.*" [54]

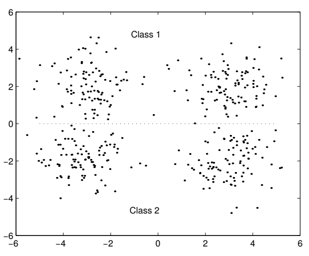
If the high-dimensional data lies in a low dimensional manifold, then unsupervised methods can be applied to reduce the number of dimensions. For example, this is useful for discriminative methods where the distance between two points tends to become more similar in higher dimensional feature spaces and, hence, less expressive.

5.2 Survey of Semi-Supervised Methods

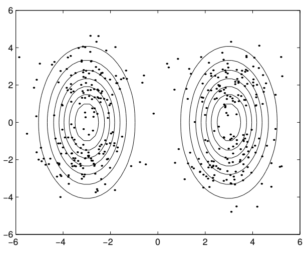
This section gives an overview of existing semi-supervised learning methods. In particular, generative models, self-learning, co-training and graph-based methods are discussed. [54, 56]

Generative Models [56] use unlabeled data to identify clusters and labeled samples to determine the corresponding classes. To train a mixture model in a semi-supervised setting, the EM algorithm [57] can be used. The difference to the EM algorithm in unsupervised learning is to handle the labeled samples not as latent variables but rather setting the proper labels, instead of summing of all possible values. An important fact is that the cluster assumption has to be correct, because otherwise incorporating unlabeled data leads to a degraded accuracy, as shown in figure 5.2.

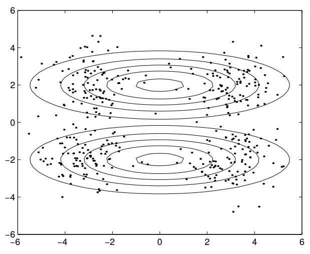
Self-Learning [56] methods are able to learn new samples by itself. To perform self-learning, a model is first trained on labeled data only. Further, the trained model is applied on unlabeled samples and the resultant predictions are used to update the model. Making predictions on unlabeled data and retraining the model is repeated until convergence. It is also possible to use only the most confident samples or to weight each unlabeled sample by the estimated confidence. Self-learning methods are simple to implement and can be applied to almost all classifiers. A disadvantage of self-training methods is that mistakes could reinforce themselves.



(a) Horizontal class separation



(b) High probability



(c) Low probability

FIGURE 5.2: If the assumption about the underlying model is wrong, higher likelihood may lead to lower classification accuracy. The data samples illustrated in 5.2(a) are not generated from two Gaussians. However, if they are assumed to be Gaussian the model shown in 5.2(b) gets a higher probability as the one in 5.2(c), even if the accuracy in 5.2(c) is much better. [56]

Co-Training and Multi-view Learning [56] use different classifiers to teach each other new samples. First, all classifiers are trained on labeled samples and, afterwards, the predicted class labels on unlabeled data are used to retrain the learned models. Co-training works because the classifiers are assumed to capture different features and, hence, the methods are able to teach each other new training samples. As self-learning methods, co-training methods are easy to implement and can be applied to existing classifiers. Co-training and multi-view learning are less sensitive to mistakes than self-training.

Graph-Based Methods [56] employ a graph structure where labeled and unlabeled samples are connected via edges. The edge weights of the graph determine if two samples tend to have the same label. To estimate the edge weight, a similarity measure is required. If two samples look similar, they are more likely to have the same class label. Graph-based methods perform well if the estimated graph structure fits the task, but do badly if the structure or weights do not reflect the underlying data distribution.

5.3 Semi-Supervised Transfer Learning

The last sections gave an overview of how unlabeled data can help and which types of semi-supervised learning methods exist. The reader also already knows from previous chapters how to train the parameters of a deep neural network in a fully supervised setting.

This section is about semi-supervised transfer learning. In particular, we study semi-supervised transfer learning to finetune a deep neural network to the target task of semantic image segmentation where labeling at pixel-level is expensive and, hence, extracting additional information from unlabeled data is desired.

We will now propose two self-learning variants to incorporate unlabeled data. These methods can be easily integrated into the training process by extending the loss function as described below.

To perform self-learning, we first extend the loss function of our deep neural network by a regularization term, as shown in equation 5.1. The goal of this term is to minimize the uncertainty on unlabeled data because if the uncertainty of predicted class labels is low, the decision boundary is well placed between the classes. This is related to the cluster assumption, which states that two classes are separated by an area that has low-density.

$$\begin{aligned} L(\boldsymbol{W}) &= L_l(\boldsymbol{W}) + \lambda L_u(\boldsymbol{W}) \\ &= -\frac{1}{L}\sum_{l=1}^{L}\sum_{k=1}^{K} y_{lk} \ln p(\mathcal{C}_k|\boldsymbol{x}_l, \boldsymbol{w}_k) - \lambda\frac{1}{U}\sum_{u=1}^{U}\sum_{k=1}^{K} h_{uk} \ln p(\mathcal{C}_k|\boldsymbol{x}_u, \boldsymbol{w}_k) \end{aligned} \tag{5.1}$$

The first sum is defined over all labeled samples and the second one over unlabeled data. The influence of unlabeled data is controlled by λ. The weight of the regularization term is important since as we already know, unlabeled data can also harm the classification accuracy. It is noted here that one could also add other regularization methods, for example, co-training methods use the confidence values of other classifiers to regularize the train loss.

The target values $\boldsymbol{y}_l$ of the labeled samples are set to the ones given in the training data. For unlabeled samples, we study two possible choices of their respective target values $\boldsymbol{h}_u$, namely hard assignment and soft assignment.

Hard Assignment

Hard assignment, or also called pseudo-label [58], uses the class that has the highest probability as target label.

$$h_u = arg\,max_k \; p(\mathcal{C}_k|\boldsymbol{x}_u, \boldsymbol{w}_k) \tag{5.2}$$

We assume that the generated pseudo-label does not depend on the model parameters and, thus, the derivation of the regularization term with respect to the model parameters has the same form as the one of labeled samples.

$$\nabla_{\boldsymbol{w}_k} L_u(\boldsymbol{W}) = \frac{1}{U}\sum_{u=1}^{U} \Big(f_k(\boldsymbol{x}_u, \boldsymbol{w}_k) \; - \; h_{uk} \Big) \; \nabla_{\boldsymbol{w}_k} f_k(\boldsymbol{x}_u, \boldsymbol{w}_k).$$

Soft Assignment

Here, the target value is set to the estimated probability value.

$$h_u = p(\mathcal{C}_k|\boldsymbol{x}_u, \boldsymbol{w}_k) \tag{5.3}$$

The previous assumption, that the target values are independent of the model parameters, cannot be made anymore because the gradient would be zero as shown here

$$\nabla_{\boldsymbol{w}_k} L_u(\boldsymbol{W}) = \frac{1}{N}\sum_{u=1}^{U} \underbrace{\Big(f_k(\boldsymbol{x}_u, \boldsymbol{w}_k) \; - \; f_k(\boldsymbol{x}_u, \boldsymbol{w}_k)\Big)}_{=0} \; \nabla_{\boldsymbol{w}_k} f_k(\boldsymbol{x}_u, \boldsymbol{w}_k).$$

However, if we relax the independence assumption, the second sum is now equal to the entropy and this leads to the semi-supervised learning method called entropy regularization [59]. We compute the gradient of the entropy with respect to the parameters and get

$$\begin{aligned}\nabla_{\boldsymbol{w}_k} L_u(\boldsymbol{W}) = \frac{1}{N}\sum_{u=1}^{U} \Bigg(& f_k(\boldsymbol{x}_u, \boldsymbol{w}_k)\Big(\ln f_k(\boldsymbol{x}_u, \boldsymbol{w}_k) \\ & - \sum_{j=1}^{K} f_j(\boldsymbol{x}_u, \boldsymbol{w}_k) \ln f_j(\boldsymbol{x}_u, \boldsymbol{w}_k)\Big)\Bigg) \nabla_{\boldsymbol{w}_k} f_k(\boldsymbol{x}_u, \boldsymbol{w}_k).\end{aligned} \tag{5.4}$$

The different loss functions and proper derivations with respect to the model parameters of both semi-supervised methods are shown in figure 5.3. As can be seen, soft assignment leads to small parameter changes in areas where the model is unconfident about the predicted class, whereas the gradient of the hard assignment leads to more significant parameter updates.

The algorithm to perform semi-supervised transfer learning with deep neural networks is given in algorithm 3. The first step is to initialize the model weights using a pre-trained model (e.g. ImageNet). Afterwards, the model is continuously updated using the semi-supervised learning loss given in equation 5.1. In the beginning, λ is set to zero to learn the model using labeled data only, but λ will be increased when the model is accurate enough to incorporate unlabeled samples. Updating the parameters is continued until the loss (measured on a validation set) stops decreasing.

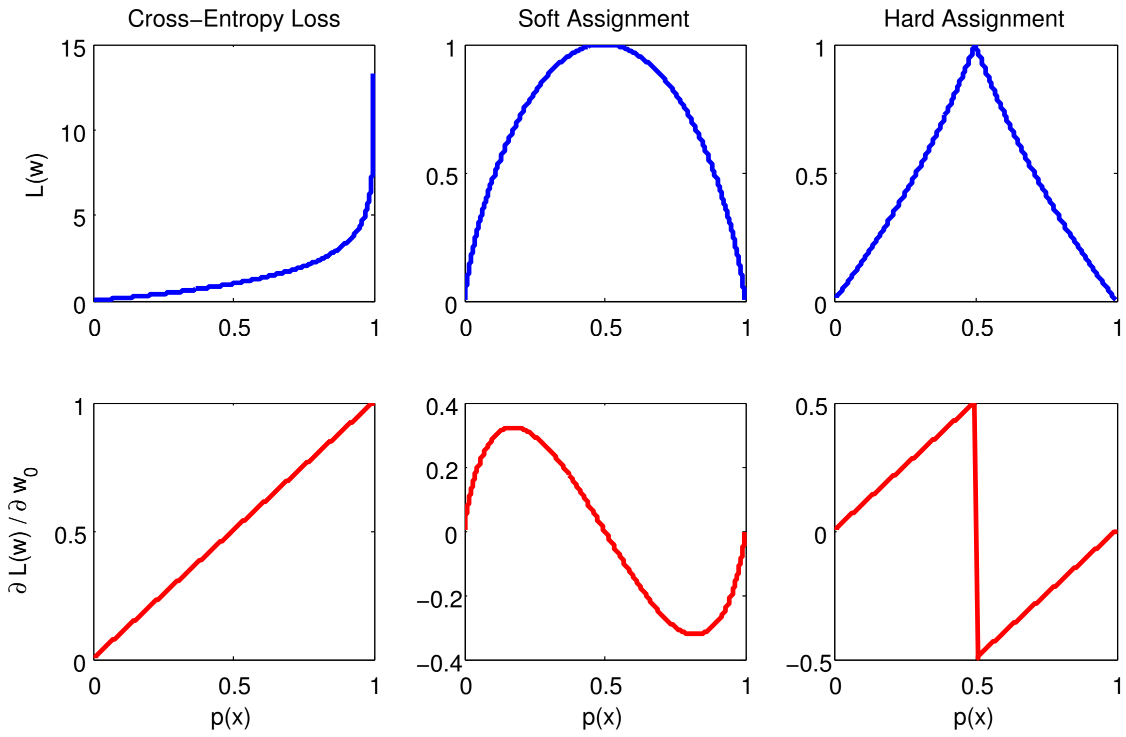


FIGURE 5.3: The plots show the difference between the cross-entropy loss and the two proposed loss functions to incorporate unlabeled data. Compared to hard assignment, soft assignment changes the model parameters less in areas where the model is uncertain about the label the random variable belongs to.

Algorithm 3 Semi-Supervised Transfer Learning with CNNs

```
$\boldsymbol{W}^{(0)} \leftarrow \boldsymbol{W}^{Pre-trained}$    ▷ Initialize from a pre-trained model
while not converged do
    $p(\mathcal{C}_k|\boldsymbol{X}, \boldsymbol{w}_k) \leftarrow f_k(\boldsymbol{X}, \boldsymbol{w}_k)$ for all $k$    ▷ Forward pass
    $L(\boldsymbol{W}) \leftarrow L_l(\boldsymbol{W}) + \lambda L_u(\boldsymbol{W})$    ▷ Compute loss
    $\nabla L(\boldsymbol{W}) \leftarrow \nabla L_l(\boldsymbol{W}) + \lambda \nabla L_u(\boldsymbol{W})$    ▷ Backward pass
    $\boldsymbol{W} \leftarrow \boldsymbol{W}^{(\tau)} - \eta \nabla L(\boldsymbol{W})$    ▷ Update parameters
end while
```

5.4 Chapter Summary

State-of-the-art methods use a pre-trained ImageNet model to initialize the weights of the CNN and further apply transfer learning to the target task of semantic segmentation.

Since this finetuning step usually requires a large amount of labeled samples and annotating images at pixel-level is expensive, we studied semi-supervised learning methods to reduce the manual effort.

We proposed two self-learning methods to incorporate unlabeled data. The extension of fully supervised CNNs is straightforward because only the loss function has to be extended. Our empirical evaluations will show that incorporating unlabeled data can reduce the manual effort of labeling data.

Chapter 6

Experimental Evaluation

Contents

We now turn to the last part of the thesis, the experimental evaluation of the proposed methods. This chapter is organized as follows. At the beginning, the datasets used for evaluation and the evaluation criteria are described. We continue by our implementation details before we study the quantitative and qualitative results of the suggested approaches. The chapter is completed with a runtime analysis of the CPU and GPU implementation.

The first experiment discussed herein is about applying fully convolutional networks to the task of aerial image segmentation. The performance is measured on three different datasets, where the first is the publicly available Toronto roads and buildings dataset [26] and the latter two are internal datasets used by the Microsoft Photogrammetry team in Graz. We perform a qualitative and quantitative evaluation on all three datasets. We show that fully convolutional networks improve the state-of-the-art performance measured on the Toronto datasets. In addition, we present promising segmentation results on internal aerial datasets used by the Microsoft Photogrammetry team.

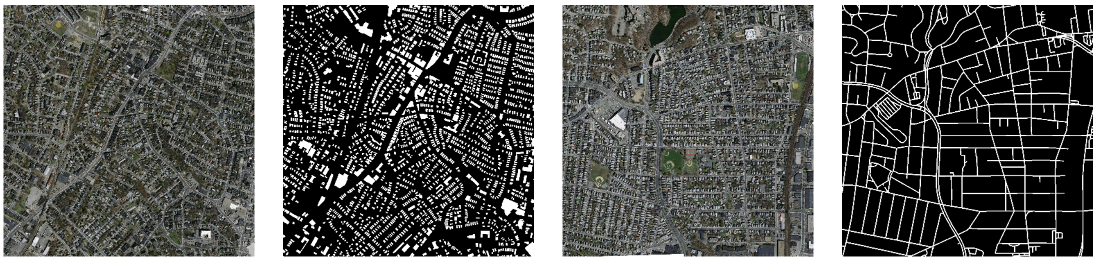

(a) Buildings - RGB (b) Buildings - Labels (c) Roads - RGB (d) Roads - Labels

FIGURE 6.1: Example images of the Toronto roads and buildings dataset.

Further, we evaluate the end-to-end learning approach presented in chapter 4. Empirical results verify the effectiveness of jointly training the deep neural network and the conditional random field (CRF). In addition, we analyze the ability of the CRF to learn the class compatibilities.

The last experiment in this chapter examines our semi-supervised transfer learning approach proposed in chapter 5. Different ratios of labeled vs. unlabeled data samples are studied. The results confirm the usefulness of incorporating unlabeled data into the training process.

6.1 Description of Datasets

In this section, the datasets used to evaluate our methods are described. All datasets provide pixel-level annotated images for training and testing.

Toronto Roads and Buildings

The Toronto roads and buildings datasets [26] consist of aerial images with binary label maps. The images of both datasets are of equal size and have a resolution of 1500×1500. A pixel in the image corresponds to one meter on the ground. The roads and buildings datasets comprise 151 and 1171 images respectively. Some example images and annotations are shown in figure 6.1.

Aerial Water Dataset

The aerial water dataset is an internal dataset used by the Microsoft Photogrammetry team. This dataset contains aerial images with pixel-level annotated water masks. The images are captured at five different areas: Barcelona, Bay Area, New York, Orlando and Takamatsu. In addition to the red, green and blue color channels, this dataset also provides the near-infrared (NIR) channel.

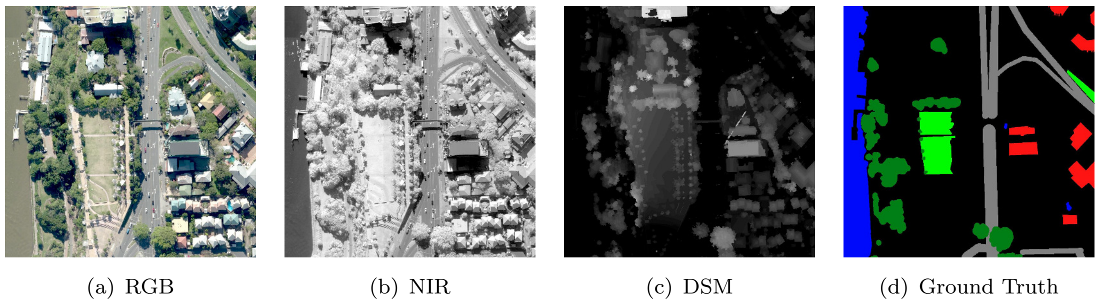


FIGURE 6.2: Example images of the aerial multi-class dataset.

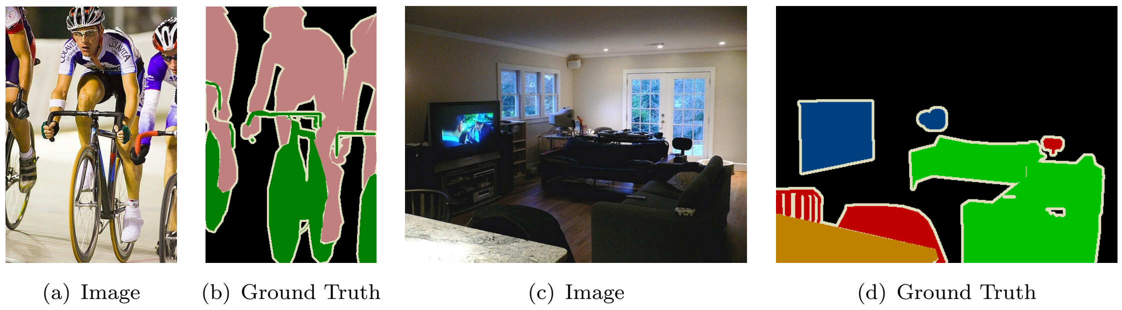


FIGURE 6.3: Example images of the Pascal VOC 2012 segmentation dataset.

Aerial Multi-Class Dataset

This dataset comprises pixel-level labeled aerial images, where the classes water, low vegetation, high vegetation, impervious surfaces, buildings and background are annotated. It contains RGB and near-infrared images as well as a digital surface model (DSM) (examples are shown in figure 6.2). The point of origin of the DSM is defined by the digital terrain model (DTM).

Pascal VOC 2012 Segmentation Dataset

We use the Pascal VOC 2012 benchmark [60] to evaluate our structured end-to-end and semi-supervised learning approaches. The dataset consists of pixel-level annotated images with 20 foreground classes labeled. The training, validation and testing splits contain 1,464, 1,449 and 1,456 images respectively. Moreover, we use additional pixel-level annotations provided by [61] to get a total number of 10,582 training images. Examples are shown in figure 6.3.

6.2 Evaluation Criteria

To quantitatively evaluate the performance of our proposed semantic image segmentation methods two measures are used, precision vs. recall and intersection over union (IoU). Both methods are justified to be used for evaluation and each method has its own advantages. Assessing segmentation results of aerial images is usually performed by the precision vs. recall measure [26, 62], whereas studies evaluated on the Pascal VOC dataset make use of use the IoU measure [60].

6.2.1 Precision vs. Recall

This method allows to trade-off between precision and recall by varying an acceptance threshold. A higher threshold leads to more correctly classified pixels but reduces the number of relevant instances that are retrieved.

Precision defines how many of the discovered class labels are correct:

$$Precision = \frac{TP}{TP + FP} \tag{6.1}$$

Recall defines the number of correct labels that are actually found:

$$Recall = \frac{TP}{TP + FN} \tag{6.2}$$

To compare different methods, we also report the break-even point where precision and recall are equal.

Relaxed Precision and Recall: Since the ground truths of the publicly available roads and buildings dataset are only accurate up to a few pixels [26], relaxed precision and recall scores are computed. It is common practice to perform such a relaxation when evaluating road detection systems [62]. The relaxed precision is given by the fraction of predicted road pixels that are within p pixels of a true road pixel, and the recall represents the fraction of true road pixels that are within p pixels of a predicted road pixel. In all our experiments, in which precision and recall is used, p is set to 3 pixels.

6.2.2 Intersection over Union

The intersection over union score is often used to compare multi-class semantic segmentation methods [60]. The IoU measure is defined as follows

$$IoU = \frac{TP}{TP + FP + FN}. \tag{6.3}$$

Compared to precision and recall, the IoU score results in lower scores because it considers true positives, false positives and false negatives. To evaluate the performance of multi-class datasets, we first compute the IoU measure for each class separately and, afterwards, compute the mean of all class scores.

6.3 Implementation Details

We now continue with the implementation details associated with all experiments.

Deep Learning Framework

We implemented the fully convolutional network as well as the CRF by extending the publicly available Caffe framework [52]. The framework is also used by most state-of-the art semantic image segmentation methods [4–6, 8]. The framework supports GPU acceleration and cuDNN [63], which gives an additional performance boost when running on NVIDIA GPUs. Since Caffe is officially only supported on Linux, we first had to port the source code to Windows.

Data Preprocessing

For training and inference, we transform each single input channel to have zero-mean. The mean values are estimated for each dataset and input channel independently on the training set.

Training and Inference

During training we augment the data by randomly cropping and flipping patches from labeled images. [16]

To optimize the objective function, stochastic gradient descent (SGD) is used. The weight decay and momentum are set to 0.0005 and 0.9. The learning rate is dropped by a factor of 10 after the number of iterations specified by the step size. The batch size, learning rate, number of iterations and step size are model dependent and are described in each experiment.

Finetuning from a pre-trained ImageNet model is done by replacing the top fully connected layer. The learning rate of the new classification layer is set ten times higher as the base learning rate used for all other layers. Finetuning is performed by adapting all weights jointly.

Hyper-parameters are estimated using the training and validation sets to avoid an adaptation of the model to the test set. For each hyper-parameter setting we first train on the training set and test on the validation set. Afterwards, we select the best setup and train the final model using the data included in the training and validation set. The final performance is measured on the test set.

To perform inference using the defined CRF, five mean-field iterations are applied.

6.4 Deep Neural Networks for Aerial Image Labeling

In this section, we show that state-of-the-art semantic image segmentation methods, which are tuned to perform predictions on images of general scenes, can be successfully adjusted to the task of aerial image labeling.

This section starts with aerial imagery related implementation details, before discussing the results on various aerial datasets.

6.4.1 Tile-based Processing

Aerial images are typically larger in size and, as a consequence, inferring the class labels of the whole image in one forward pass is impossible because of the GPU memory limitations. To overcome this limitations, a tile-based processing is implemented. Each image is first cropped into tiles and, afterwards, each individual patch is processed by the neural network. After the class labels have been inferred, the image is reassembled. To reduce artifacts at tile-borders, a padding is used, so the patches do overlap each other. The padding size depends on the network architecture and on the dataset.

6.4.2 Additional Data Channels

Aerial imagery usually comprises additional data channels like NIR or a DSM. We also perform transfer learning in such a setting. We already described in chapter 2 how to map new features to a pre-trained RGB color model to enable transfer learning in such a setting.

In addition to NIR and DSM, we feed the network with derivations of the RGB and NIR channels, which are commonly used to perform pixelwise labeling of aerial images [64]:

Normalized Difference Vegetation Index (NDVI)

The spectral response of vegetation is characterized by radiance in the NIR channel, much higher than in the bands of the visible spectrum. The lowest values are taken

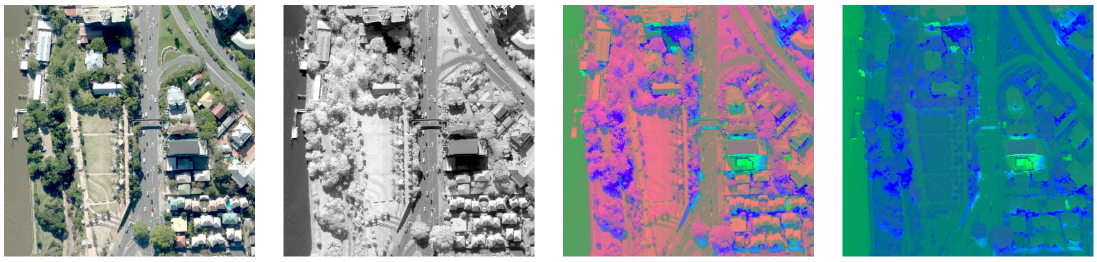

(a) RGB (b) NIR (c) NNN (d) NND

FIGURE 6.4: Examples of combined feature channels of the aerial multi-class dataset.

on in blue and red bands. The NDVI index combines the red and NIR bands to better identify vegetation. [64]

$$NDVI = \frac{NIR - RED}{NIR + RED} \tag{6.4}$$

Normalized Difference Water Index (NDWI)

Typically, water has a high reflectance in green but not in NIR, whereas vegetation and soil have a high response in the NIR channel. According to this, water usually has a positive NDWI index, while vegetation and soil have a negative one. Therefore, the NDWI index allows to better discriminate these classes. [64]

$$NDWI = \frac{GREEN - NIR}{GREEN + NIR} \tag{6.5}$$

In the following, we denote NNN as the combination of the NDVI, NDWI and NIR features, and NND as the combination of the NDVI, NDWI and DSM features. Example images with combined feature channels are shown in figure 6.4.

As can be seen in our evaluations, incorporating such domain knowledge additionally improves the classification accuracy for certain classes.

6.4.3 Toronto Roads and Buildings

In this section, the results on the Toronto roads and buildings dataset are presented. We compare our proposed method with the two best performing methods on this dataset [26, 27].

The challenges on the Toronto roads and buildings dataset are given by detecting small-scaled objects. State-of-the-art image segmentation methods based on fully convolutional networks lack in recognizing very small objects due to the pooling operations in higher-level layers. As already discussed in chapter 3, we overcome this issue by reducing the stride of the *pool*3 layer of the DeepLab-LargeFOV [5] network and applying the hole-algorithm to subsequent layers. In addition, we upscale the input images by a factor of two to further improve the accuracy in detecting small-scaled objects. The crop size of the patches during training is set to 153×153 pixels and at inference to 257×257. Padding is set to 15 pixels.

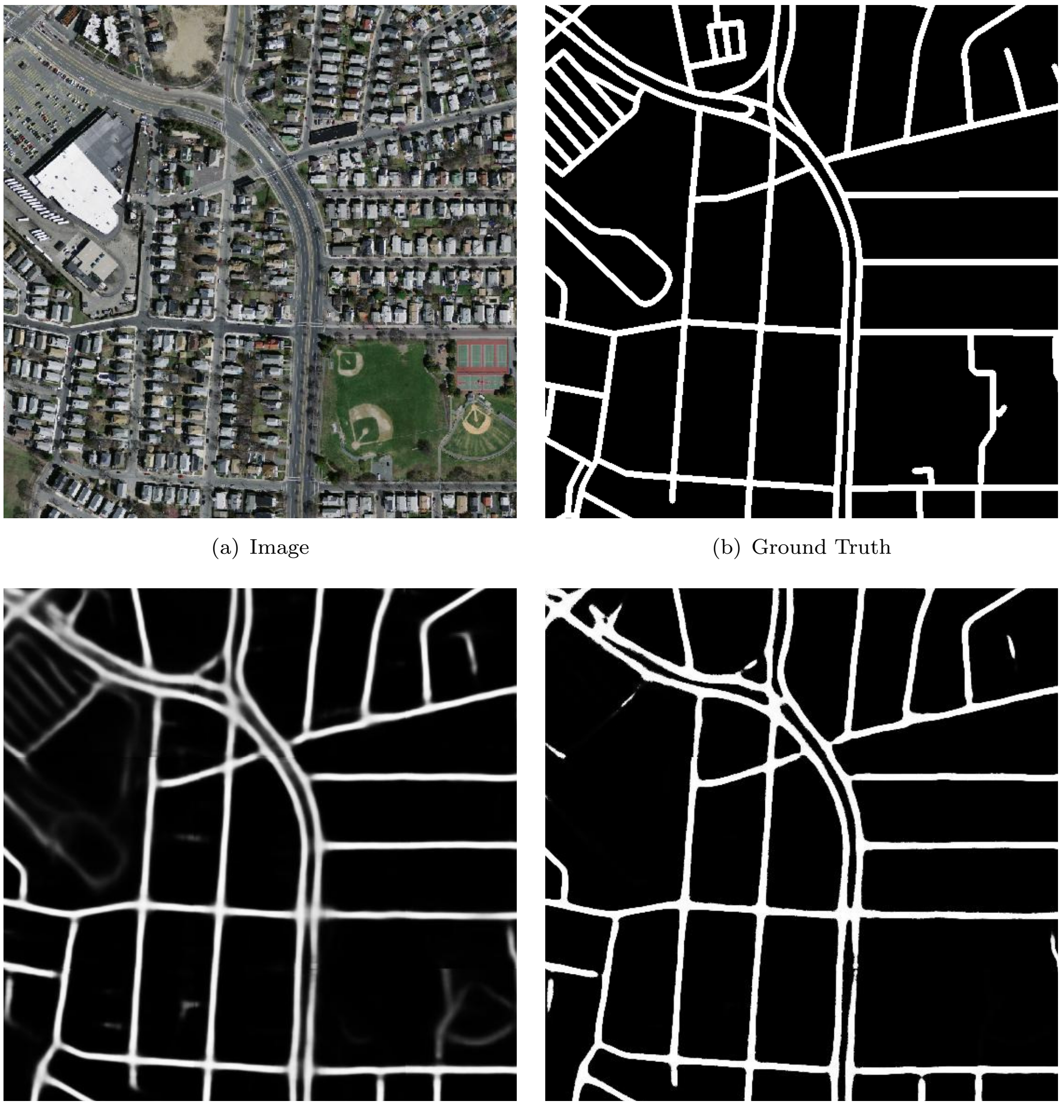
To train the modified architecture, a pre-trained VGG-16 model [17] is used to initialize the weights of the network. The classification layer, which is trained from scratch, will be initialized with random values sampled from a Gaussian distribution with standard deviation 0.01. The network is finetuned using the training and validation set for $90,000$ iterations, the step size is set to $30,000$ iterations and the learning rate is set to 0.001. The right choice of the learning rate and the number of iterations is important because they have a wide influence on the overall performance. To estimate these parameters, we perform a grid search by first varying the learning rate and keeping the number of iterations fixed. Afterwards, we fix the best found learning rate and search for the number of training iterations. Due to memory limitations, only one patch is used in each update step. The parameters of the CRF with $w_1 = w_2 = 2$, $\sigma_\alpha = 10$ and $\sigma_\beta = \sigma_\gamma = 1$ turned out to work well on both datasets.

Model	Dataset	
	Roads	Buildings
Volodymyr - Neural Network	0.8873	0.9150
Volodymyr - CRF	0.8904	0.9211
Volodymyr - Post-processing net	0.9006	0.9203
Saito and Aoki - Neural Network	0.8905	0.9241
Neural Network	**0.9148**	**0.9558**
Neural Network + CRF	**0.9224**	**0.9584**

Table 6.1: Comparison of our approach with previous released methods. The values are the estimated precision-recall break-even points.

The final results of the proposed method are presented in table 6.1 and compared to previous released studies by Volodymyr [26] and Saito and Aoki [27]. The table shows the break-even points of the estimated precision and recall. The quantitative results verify that state-of-the-art semantic segmentation methods, which are tuned for general scenes, can be successfully adapted to the task of aerial image labeling.

Both competing methods apply neural networks too but with less network layers as we do. Furthermore, they do not take advantage of the large-scale ImageNet dataset [28]. Our empirical evaluation shows that using deep neural networks and finetuning a pre-trained model to the task of aerial image segmentation pushes the state-of-the-art in aerial image

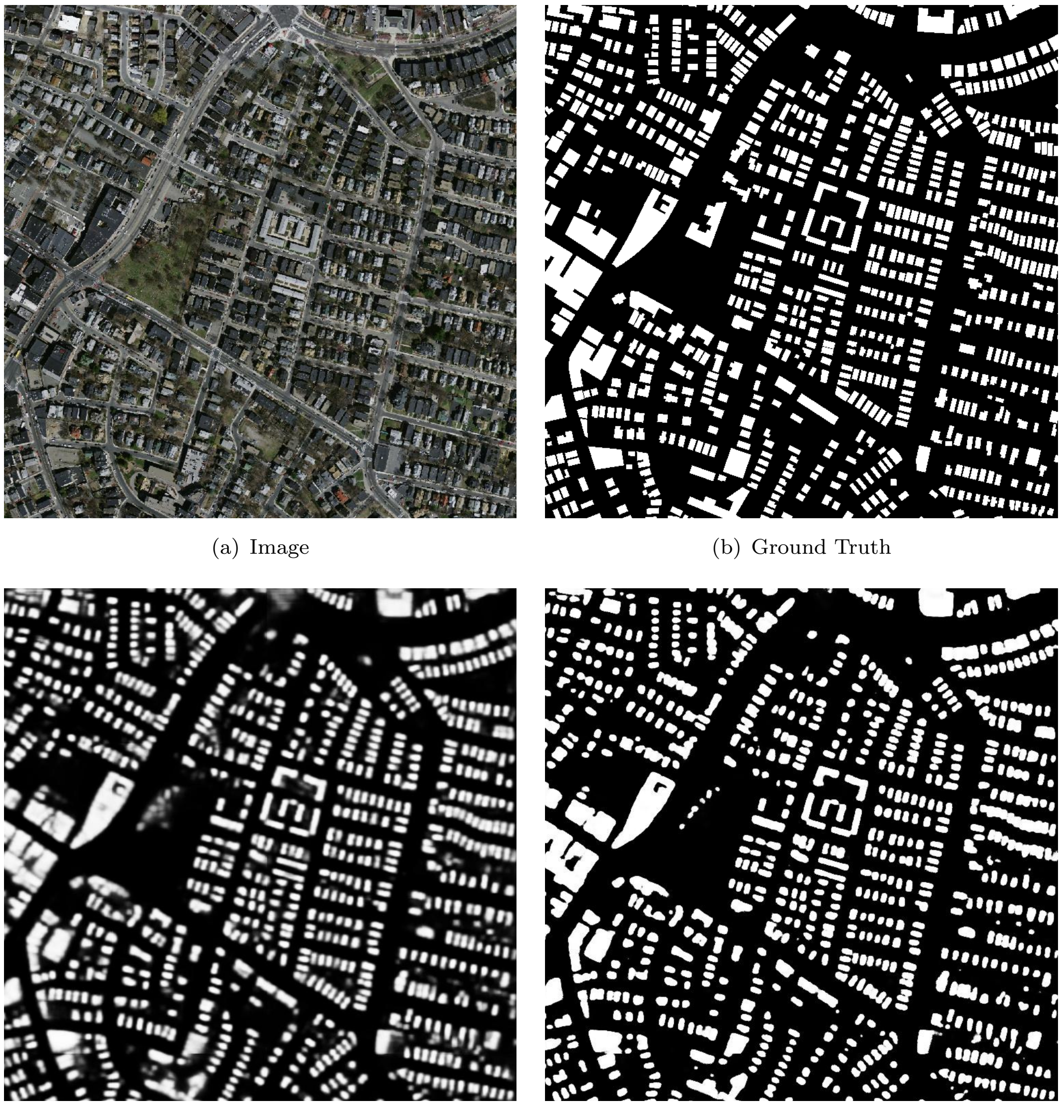

(a) Image (b) Ground Truth

(c) Neural Network (d) Neural Network + CRF

FIGURE 6.5: Example results on the Toronto roads dataset.

labeling. Applying a CRF to refine the predictions of the network additionally increases the overall performance.

Qualitative results of detected roads and buildings are shown in figure 6.5 and 6.6, respectively. In general, roads and buildings are predicted well. Even if streets are occluded, the neural network is able to predict them. Failures occur, for example, if the roads are very small or at large paved areas like company or parking areas. The precision of the predictions is very high. Improvements could be achieved by increasing the recall. Usually, at intersections one is often able to see gaps. Applying a CRF on top of the neural network leads to a small increase in accuracy. The visual results of the CRF show the typical effects



(a) Image

(b) Ground Truth

(c) Neural Network

(d) Neural Network + CRF

FIGURE 6.6: Example results on the Toronto buildings dataset.

when applying mean-field inference, namely the confidence values are driven to either zero or one. The influence of the CRF can be seen in areas where the confidences of the neural network are neither near zero nor one.

Future work could comprise adding constraints to the predictions. For example, considering the typical outlines of buildings and roads could be beneficial because the predictions are not perfectly aligned with them.

6.4.4 Aerial Water Dataset

In this section, we evaluate our proposed architecture on the aerial water dataset. One of the main challenges on the aerial water dataset is given by the high intra-class variability of water. To overcome this, the dataset additionally comprises the NIR channel, which is beneficial to detect water [64].

As in the previous experiment, we use the modified DeepLab-LargeFOV [5] network architecture, which is tuned to detect small-scaled objects, but we do not upscale the input images because water areas are typically large in size. During training, the crop size of the patches is set to 306×306 pixels and at inference to 514×514 pixels. The padding to crop the tiles is set to 30 pixels.

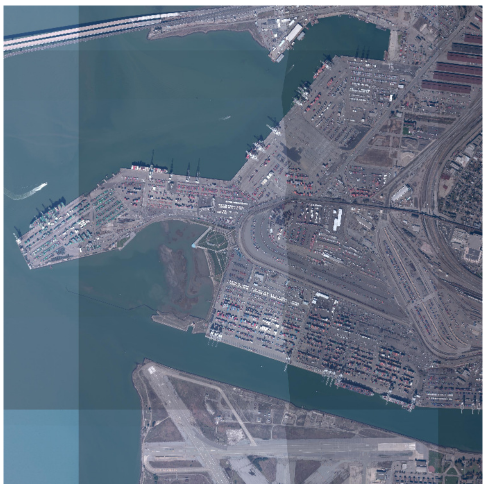
To train the modified architecture, a pre-trained VGG-16 model [17] is used to initialize the weights of the network and the classification layer is initialized with Gaussian distributed values and standard deviation 0.001. The network is finetuned using the training and validation set for $45,000$ iterations, the step size is set to $15,000$ iterations and the learning rate is set to 0.0001. At training, the batch size is set to three. The parameters of the CRF with $w_1 = w_2 = 3$, $\sigma_\alpha = 50$ and $\sigma_\beta = \sigma_\gamma = 3$ lead to good results.

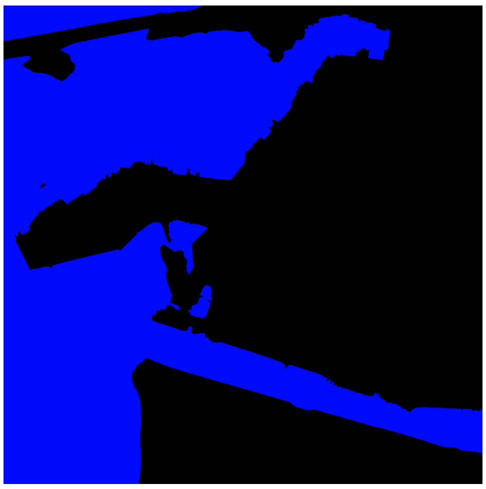
Model	Input Channels		
	RGB	NNN	RGB+NNN
Neural Network	0.9528	0.9695	0.9700
Neural Network + CRF	0.9595	0.9730	0.9700

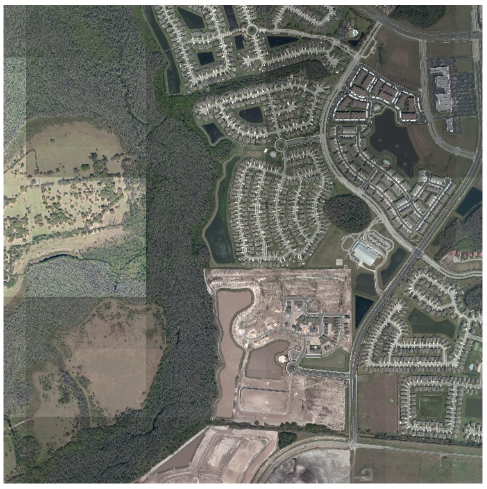
Table 6.2: Accuracy on the Aerial Water dataset. The values are the precision-recall breakeven points.

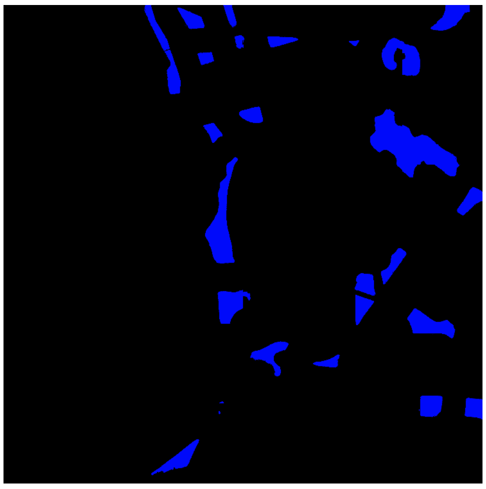
The final results of the water classifier are shown in table 6.2. Using a combination of the NDVI, NDWI and NIR channels (NNN) leads to better results as compared to RGB intensity. We also tried to combine both models by averaging the confidences of the RGB and NNN model, but it did not lead to noticeable improvements, or even decreased the accuracy slightly if a CRF is applied in addition.

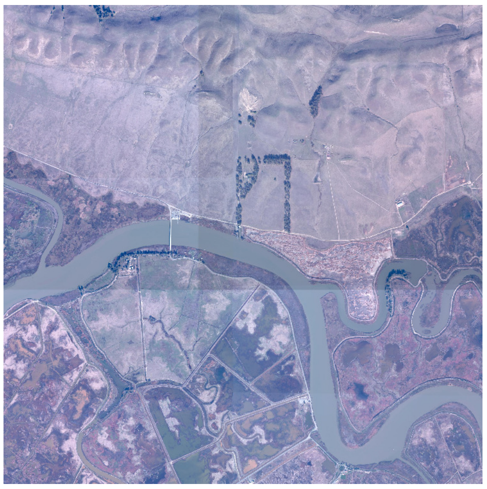
Figure 6.7 shows images and corresponding predictions of the water classifier. Overall, the precision is again very high. Improvements could be achieved by increasing the recall. The classifier has problems on areas that are dissimilar to ones given in the training set. For example, when the water or the surroundings of water look very different compared to the labeled samples in the training set, then the classifier has a degraded performance. An example is illustrated in figure 6.8, which shows a water fountain that is not detected by classifier. Problems also arise at borders of the cropped tiles, but this could be prevented by


(a) Image


(b) Neural Network + CRF


(c) Image


(d) Neural Network + CRF


(e) Image

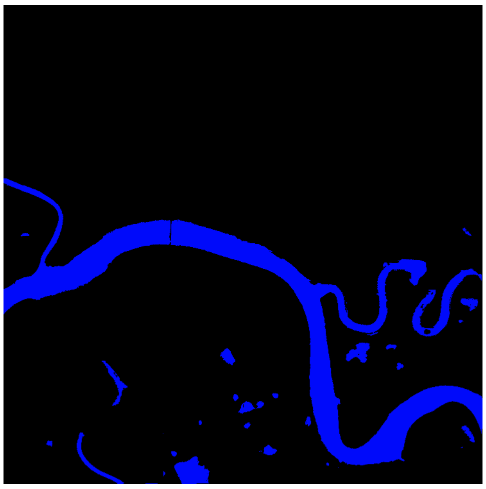

(f) Neural Network + CRF

FIGURE 6.7: Example results on the Aerial Water dataset using the RGB model.

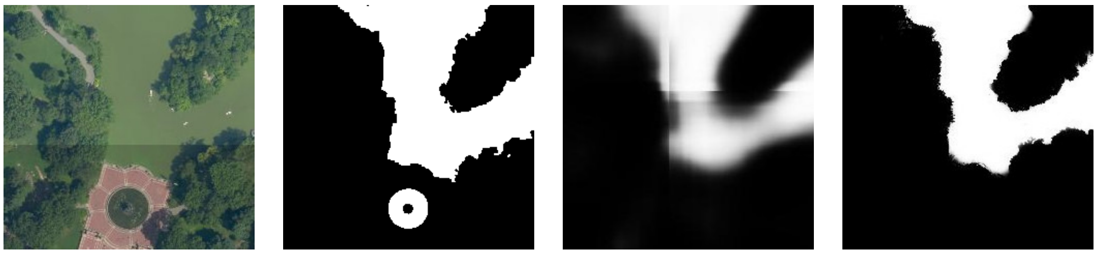


FIGURE 6.8: Example results on the Aerial Water dataset using the RGB model.

increasing the overlap between tiles if the associated computational costs can be neglected. Artifacts at border areas are also an indicator that the context is considered to classify water and not only the colors itself. False negatives can be observed at very small rivers or swimming pools. This can be improved by upscaling the images or by using aerial images of higher resolution, but doing so would increase the computational costs too. The confidence maps generated by the neural network are not well aligned with the image gradients, but the refined results of the CRF are quite close beside the water bounds given in the input images, as shown in figure 6.8.

Since most errors occur in areas that are dissimilar to ones given in the training set, this indicates that the classifier has problems to generalize well on new areas. To overcome this issue, domain adaptation methods, which allow models to adapt to new data distributions, could be studied.

6.4.5 Aerial Multi-Class Dataset

So far, we have only tackled binary classification problems. In this experiment, we apply the proposed method on the aerial multi-class dataset.

Since the images in this dataset have a high resolution and, hence, objects appear larger as in the previous used datasets, we use the standard DeepLab-LargeFOV [5] architecture without upscaling the input images. We saw that the architecture tuned for small-scaled objects does not perform well on this dataset. This might be because of the decreased receptive field, which goes along with reducing the stride in the *pool*3 layer, as we did in the previous sections. The crop size of patches during training is set to 321 × 321 pixels and at inference to 513 × 513 pixels. The padding to crop the tiles is set to 30 pixels.

The network is again initialized using a pre-trained VGG-16 model [17] and the classification layer is initialized with Gaussian distributed weights with standard deviation 0.01.

The network is finetuned on the training and validation set for 150, 000 iterations, the step size is set to 50, 000 iterations and the learning rate is set to 0.0001. At training, the batch size is set to three. We set the parameters of the CRF to $w_1 = w_2 = 5$, $\sigma_\alpha = 50$ and $\sigma_\beta = \sigma_\gamma = 3$. Since the predicted class boundaries of the DNN are not well aligned with the given input image, the weights of the filter kernels are increased compared to previous experiments.

The estimated accuracy of the neural network and the CRF are shown in table 6.3. In addition to RGB and NIR, this dataset also includes a DSM. Thus, we also train a model that makes use of NDVI, NDWI and the additional DSM channel.

We provide the accuracy for each class in table 6.4 and table 6.5, one table for the accuracy of the neural network and one with additionally applying the CRF.

Measured by the mean IoU, the RGB model works overall slightly better than the NNN (NIR, NDWI and NDVI) model. The NND (NDWI, NDVI and DSM) is on par with the RGB model. Taking a closer look on the individual class accuracies, one can see that the NND model, which additionally uses the DSM, performs better on impervious surfaces, high vegetation and buildings, but shows degraded performance on other classes, where height information is not expressive, which is as expected. The biggest improvement using the DSM channel can be seen at buildings. Additionally, incorporating the NIR channel improves the accuracy of the class water, as we have seen on the aerial water dataset.

Model	Input Channels		
	RGB	NNN	NND
Neural Network	0.8797	0.8633	0.8776
Neural Network + CRF	0.8884	0.8740	0.8886

Table 6.3: Accuracy on the Aerial Multi-Class dataset. The values represent the mean IoU.

Classes	Input Channels		
	RGB	NNN	NND
Water	0.9793	0.9851	0.9825
Low Vegetation	0.7987	0.7743	0.7762
High Vegetation	0.9605	0.9604	0.9675
Impervious Surfaces	0.8587	0.8375	0.8665
Buildings	0.8891	0.8606	0.9327
Background	0.7918	0.7617	0.7400

Table 6.4: Accuracy of the neural network on the Aerial Multi-Class dataset. The values represent the mean IoU.

Qualitative examples are presented in figure 6.9. The figure shows the results created by the RGB model after applying the CRF. As already indicated by the quantitative results,

Classes	Input Channels		
	RGB	NNN	NND
Water	0.9842	0.9874	0.9847
Low Vegetation	0.8119	0.7919	0.7968
High Vegetation	0.9627	0.9632	0.9716
Impervious Surfaces	0.8702	0.8528	0.8821
Buildings	0.8975	0.8737	0.9464
Background	0.8038	0.7752	0.7499

Table 6.5: Accuracy of the neural network and the CRF on the Aerial Multi-Class dataset. The values represent the mean IoU.

the class water is detected pretty well. Taking a closer look on other classes, one can see that the predicted boundaries are not well aligned to the objects present in the images. This is contrastive to what we have seen in other experiments in this thesis. However, if one neglects the errors made at boundaries, the accuracy is quite good.

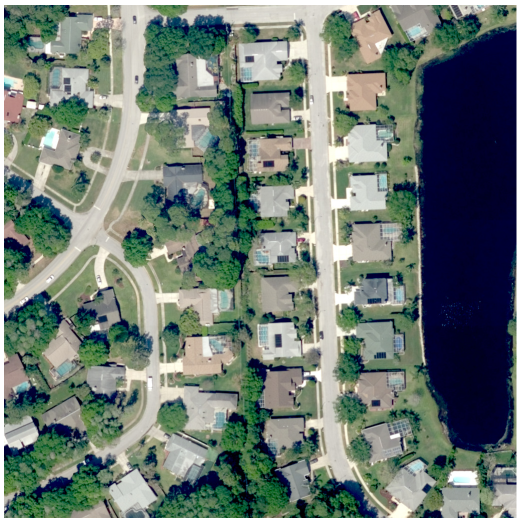
A reason why the boundaries are not predicted is because of the model needs to decide between several classes for each pixel and some categories are conflicting each other. For example, if a tree occludes a road, then the classifier has to decide between one of these categories, but the correct answer would be to assign both categories. Another explanation could be that the dataset is labeled differently. As shown in figure 6.2, not all object instances are labeled and, therefore, marked as *unknown*. Thus, problems may arise if the classifier needs to assign labels to areas that are often labeled as unknown in the training set. Such areas can be seen between buildings or in general between objects.

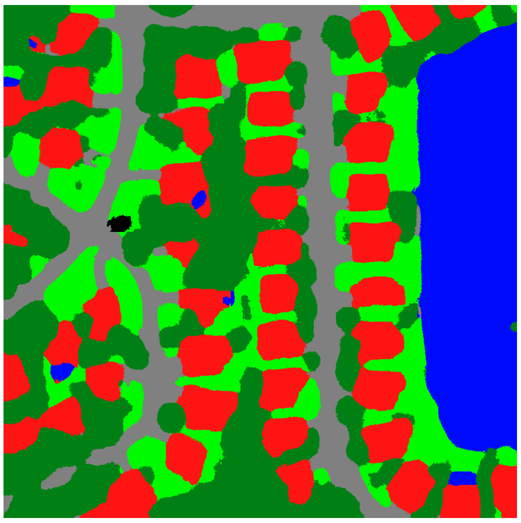
Further investigations should focus on the object boundaries that are not well aligned with the intensities of given images. For example, a binary classifier could be trained for each class separately. This should decrease the errors where the classes overlap each other. Or, even better, a multioutput-multiclass classifier could be used, which reduces the need to apply a deep neural network for each binary classification problem.

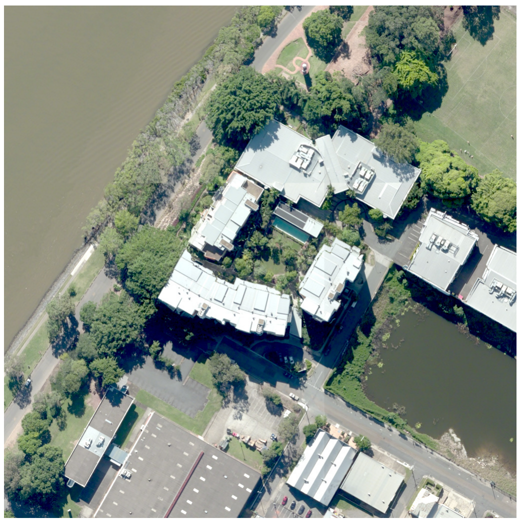
6.4.6 Transfer Learning using Pre-trained ImageNet Models

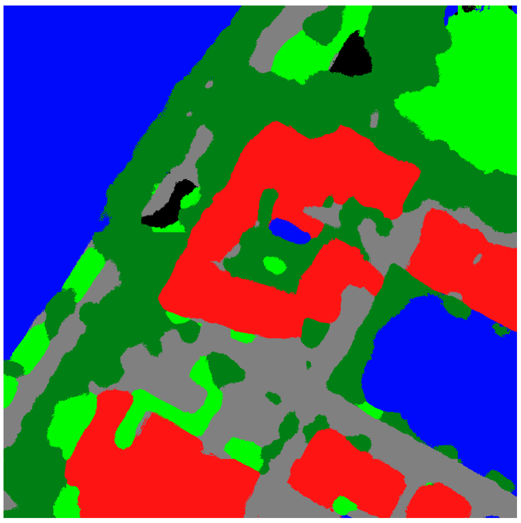
In this section, we show that the learned feature representation of images of general scenes can be transferred to perform aerial image labeling. For that purpose, we train a binary classifier on the aerial water dataset and compare the estimated loss during training.

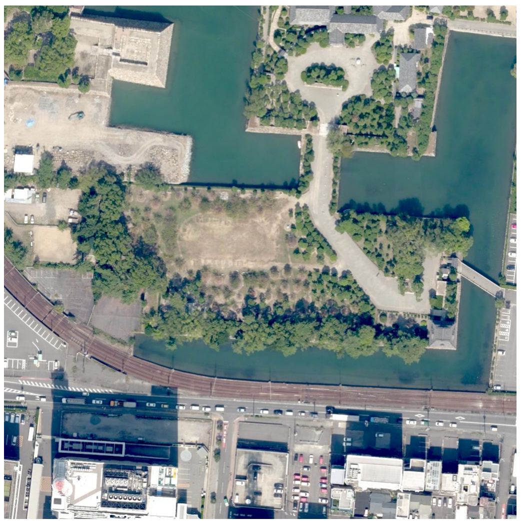
The network architecture is a VGG-16 variant that is proposed by Chen et al. [5] and called DeepLab-LargeFOV. To train the models from scratch, the network weights are initialized using the suggested method by Xavier and Bengio [65]. The initialization is


(a) Input image


(b) Neural Network + CRF


(c) Input image


(d) Neural Network + CRF


(e) Input image

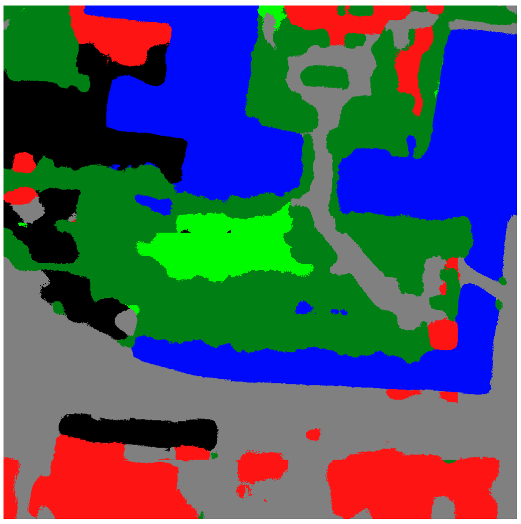

(f) Neural Network + CRF

FIGURE 6.9: Example results on the Aerial Multi-Class dataset using the RGB model.

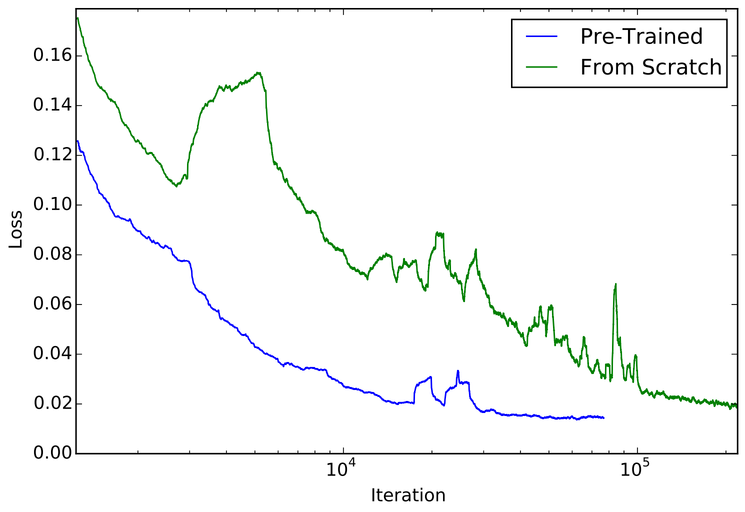


FIGURE 6.10: Comparison of the train progress by using a pre-trained ImageNet model and training a model from scratch. Even, if the pre-trained model is trained on images of general scenes, the learned features can be transferred to the task of aerial image labeling.

important, because otherwise training such a deep network may not be successful or has to be done layer-by-layer [17].

Figure 6.10 shows a comparison of the train progress by using a pre-trained ImageNet model and training a model from scratch. The pre-trained model starts with a lower loss and converges faster to the minimum than the one that is trained from scratch. This verifies that the learned feature representation from images of general scenes can also be re-used to label aerial images.

6.5 Structured End-to-end Learning

In this section, we show the results using structured end-to-end learning. End-to-end learning of the deep neural network and the CRF allows to optimize for errors that are made by both modules.

We compare our end-to-end approach against the decoupled DeepLab version [5], in which the deep neural network and the CRF are not trained end-to-end. The performance is evaluated on the Pascal VOC 2012 benchmark.

We choose the DeepLab-LargeFOV [5] architecture because of its efficient runtime and state-of-the-art accuracy. The crop size of patches during training is set to 321×321

pixels and at inference to 513×513 pixels. The standard deviations of the filter kernels are set to $\sigma_\alpha = 50$ and $\sigma_\beta = \sigma_\gamma = 3$.

Before the system is trained end-to-end, we first train the neural network parameters only. Due to memory limitations, we choose a batch size of ten but increase the number of iterations to $18,000$ to ensure the same number of epochs. The step size is set to $6,000$ and the learning rate is 0.001. The accuracy of the base deep neural network (DNN) without applying a CRF on top is 65.27 and with a decoupled CRF is 67.66.

Finally, we train the whole system end-to-end. For that, we initialize the weights of the Gaussian feature kernels to $w_1 = w_2 = 5$ and the compatibility weights to the Potts model to ensure convergence. The batch size is set to one because of memory limitations. The learning rate of the CRF parameters are set to $10e$-10 and the one for the DNN to $10e$-5. We update the model for additional $15,000$ iterations with a step size of $5,000$ iterations. The final accuracy of the end-to-end trained model is 68.23 and, thus, verifies that end-to-end training can additionally improve the overall accuracy.

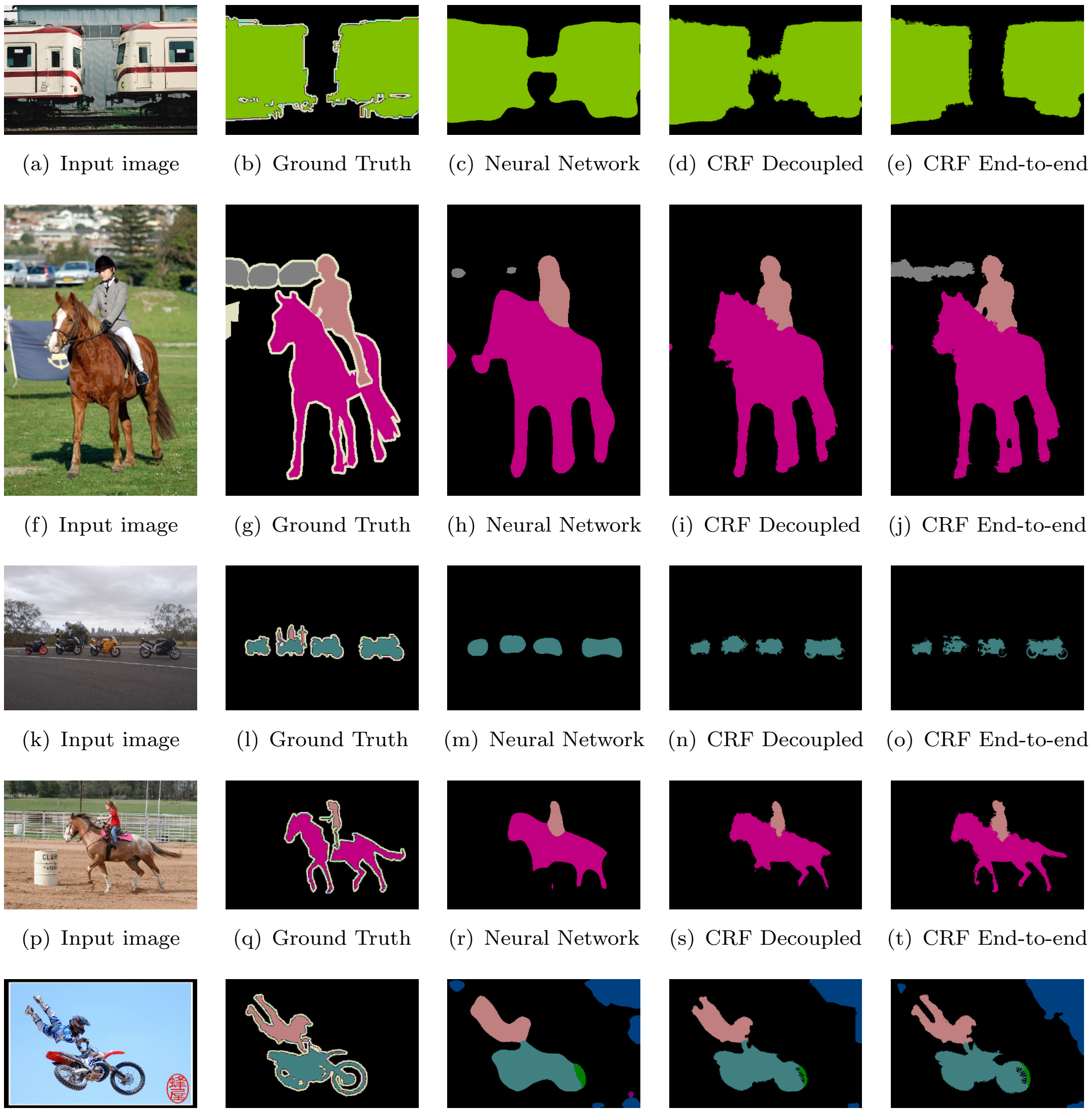
Model	Mean IoU
Neural Network	65.27
Neural Network + CRF Decoupled	67.66
Neural Network + CRF End-to-end	**68.23**

Table 6.6: Comparison of end-to-end and decoupled training on the Pascal VOC 2012 validation dataset. The values represent the mean IoU.

The stepwise improvements from using a single neural network up to end-to-end learning are shown in table 6.6. Qualitative results are illustrated in figure 6.11, which reveal the advantage of training the whole system end-to-end, namely the predictions by the end-to-end trained model contain more fine-grained structures. This is because of the estimated loss through the CRF pushes more weight at object boundary regions.

It should be noted here that finding the right learning parameters, like the number of iterations or different learning rates for convolutional layers and the CRF, is crucial to get satisfying results.

Finally, we analyze the learned parameters of the CRF. Therefore, we train a model with increased learning rate for the CRF layers to amplify the learned class compatibility weights and plot the parameter values in figure 6.12. Taking a closer look, one sees that the class compatibilities are not symmetric, which is obvious because, for example, images of motorbikes frequently contain persons, whereas only some images of persons also comprise motorbikes. An interesting insight is that the diagonal entries correlate with the number of objects per class given in the training set. This is explainable because if a specific



(a) Input image (b) Ground Truth (c) Neural Network (d) CRF Decoupled (e) CRF End-to-end

(f) Input image (g) Ground Truth (h) Neural Network (i) CRF Decoupled (j) CRF End-to-end

(k) Input image (l) Ground Truth (m) Neural Network (n) CRF Decoupled (o) CRF End-to-end

(p) Input image (q) Ground Truth (r) Neural Network (s) CRF Decoupled (t) CRF End-to-end

(u) Input image (v) Ground Truth (w) Neural Network (x) CRF Decoupled (y) CRF End-to-end

FIGURE 6.11: Example results on the Pascal VOC 2012 validation set.

class occurs more often, then the algorithm will make more errors at predicting this class. Consequently, the probability of that class occurring with oneself decreases.

6.6 Semi-Supervised Transfer Learning

Finally, we analyze the proposed method to perform semi-supervised transfer learning from a pre-trained ImageNet model to the target task of semantic segmentation.

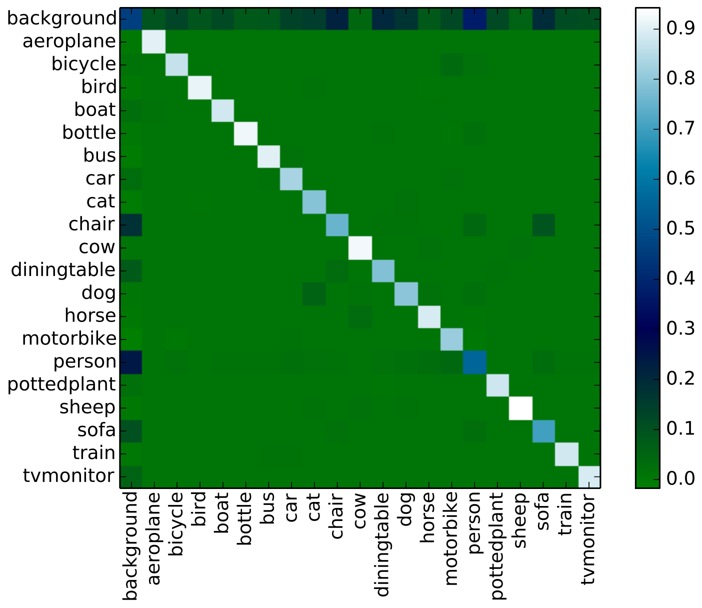


FIGURE 6.12: The learned class co-occurrence using the specified conditional random field.

The performance is again evaluated on the Pascal VOC 2012 segmentation dataset. In particular, we analyze the performance if 25, 50, and 75 percent of the labels are given, the remaining samples are incorporated as unlabeled samples during training.

The network architecture in this experiment is again the efficient DeepLab-LargeFOV architecture [5].

To perform semi-supervised learning, we follow the self-learning approach and first train the model using labeled data only. Due to memory limitations, the batch size is set to ten and the number of iterations are set to $4,500$, $9,000$ and $13,500$ to train the model with 25, 50, and 75 percent of the labeled training samples. The step size is set to a third of the number of iterations and the learning rate is set to 0.001. The parameter λ, which controls the influence of unlabeled data, is set to 1.0 in the case of hard assignment and to 0.15 in the case of soft assignment.

To perform semi-supervised learning, we retrain the fully supervised models with the loss function specified in chapter 5. We update the models with the same number of iterations as given above and repeat this process until convergence. Experiments showed that usually after two or three restarts the accuracy stops increasing.

Model	Labeled Training Samples			
	25%	50%	75%	100%
Supervised - Standard Training	54.38	60.64	64.11	65.27
Supervised - Continued Training	56.04	61.71	64.79	66.27
Semi-Supervised - Soft Assignment	55.02	61.76	64.83	-
Semi-Supervised - Hard Assignment	**57.16**	**62.80**	**65.04**	-

Table 6.7: Comparison of the proposed semi-supervised transfer learning approaches on the Pascal VOC 2012 validation dataset. The values represent the mean IoU.

We report the final results in table 6.7. The results of the model *Supervised - Standard Training* are the ones using labeled data only. *Semi-Supervised - Soft Assignment* and *Semi-Supervised - Hard Assignment* are the results using the proposed self-learning methods in chapter 5. *Supervised - Continued Training* are the results using labeled data only but with continued training until the accuracy stops increasing.

Both self-learning variants are able to improve the accuracy by a small margin. Interestingly, hard assignment achieves a higher accuracy compared to soft assignment. The reason for that might be the more aggressive weight updates by hard assignment.

6.7 Runtime Performance

In many applications, like in aerial image segmentation, the runtime is critical since usually a huge amount of data has to be processed.

In this section, the runtime of the DeepLab-LargeFOV [5] architecture is analyzed. We first specify the utilized hardware and software before we present the performance values.

The runtimes of the GPU and CPU implementations are determined. The used GPU is a NVIDIA Tesla K20Xm with 6GB memory and the CPU is a Xeon E5-2670 with 8 physical and 16 logical cores. We utilize the public available Caffe [52] framework with activated cuDNN [63] and Intel MKL support to test the performance of the DNN. For the CRF, we use the public available code of the fully connected CRF [25].

The goal of the performance test is to measure the time required to infer the class labels of a single image patch with 512×512 pixels without reading and writing files.

On the GPU the DNN requires 0.6 seconds to infer the class labels of a given patch. Using the CPU lasts approximately 11 seconds if multi-threading is activated and 33 seconds using a single thread. The runtime on the CPU can be further reduced to approximately 3.8 seconds in average if five processes are started in parallel.

The CPU implementation of the fully connected CRF requires approximately 0.6 seconds on a CPU.

6.8 Chapter Summary

In this chapter, we started with a description of the datasets we used to benchmark our proposed methods. In addition, we gave details about the implementation and evaluation criteria.

The first experiment was about applying fully convolutional networks to the task of aerial image labeling. We were able to increase the state-of-the-art on the Toronto roads and buildings datasets and achieved promising results on two internal datasets used by the Microsoft Photogrammetry team in Graz. Furthermore, we verified that the learned representation from the large-scale dataset ImageNet can be successfully transferred to perform semantic segmentation of aerial images.

We further analyzed end-to-end training of the proposed method by Chen et al. [5] and were able to increase the performance by jointly training the deep neural network and the CRF.

Finally, our semi-supervised transfer learning approach was measured by taking only a subset of the training set as labeled samples and the remaining ones as unlabeled ones. Our empirical results showed that incorporating unlabeled data increases the accuracy against using labeled data only. Therefore, semi-supervised learning methods are able to reduce the manual labeling efforts.

We concluded this chapter with a runtime analysis of our CPU and GPU implementation.

Chapter 7

Conclusions and Future Work

Contents

In this chapter, we will summarize our contributions and give an outlook for future research.

7.1 Conclusions

In this work, we showed that state-of-the-art fully convolutional networks [4], which are tuned to detect general scenes, can be successfully adjusted to the task of aerial image labeling. Based on the work of Chen et al. [5], we proposed a modified network architecture to improve the accuracy at detecting small-scaled objects. In addition, we were able to evince that the learned feature representations from the large-scale ImageNet dataset [28] can be re-used to perform pixelwise classification of aerial images. We evaluated our proposed method on the Toronto roads and buildings benchmark [26] and were able to improve the state-of-the-art accuracy. Furthermore, we demonstrated promising results on two internal aerial datasets used by the Microsoft Photogrammetry team.

To perform semantic segmentation, state-of-the-art methods [4–6] rely on fully convolutional networks, but these networks do not consider dependence of adjacent pixels. To overcome this issue, recent methods [5, 6] apply a conditional random field (CRF) on top of the fully convolutional network to model the neighbor relationship. Chen et al. [5] combined a fully connected CRF [25] and a fully convolutional network, but they did not investigate into end-to-end training of these components. End-to-end training allows to reduce the errors that are caused by both modules. For that reason, we studied end-to-end training and were able to increase the accuracy compared to the original work by Chen et al. In addition, we saw that the CRF is able to learn the class co-occurrences from data.

State-of-the-art semantic segmentation methods make use of pre-trained ImageNet models to first initialize the weights of the fully convolutional network and, afterwards, finetune [19] to the target task using a labeled dataset. Because of finetuning usually requires a large amount of labeled samples and annotating images at pixel-level is expensive, we studied semi-supervised learning methods to reduce the manual annotation effort. In particular, we adapted two self-learning methods [58, 59] to additionally exploit unlabeled data. The extension of fully supervised convolutional neural networks (CNNs) to apply the proposed methods is straightforward since only the loss function has to be extended. Our empirical evaluations confirmed that unlabeled data is beneficial to reduce the manual labeling effort.

7.2 Future Work

As we saw in the experimental evaluations, there is room for improvements and major challenges are still unresolved, like the huge manual effort to generate pixel-level annotated images.

To reduce the need of pixel-level annotated images, we applied self-learning techniques. Future work could comprise to delve into more sophisticated algorithms like graph-based methods. In combination with structured prediction, these methods achieve promising results compared to self-learning approaches, as shown by Li and Zemel [66].

To model the dependency of nearby class labels, we used a CRF with pairwise potentials that are defined via Gaussian feature kernels. These simple kernels could be replaced by more expressive models like CNNs to further improve the accuracy, as proposed by Liu et al. [67].

Since the classifier is trained on a finite dataset, it is not guaranteed that the classifier achieves the same accuracy on new samples. For example, in the case of aerial images, the classifier is typically trained using a dataset that consists of a few areas of interest and, as a result, the classifier has a degraded performance on new unseen areas that are dissimilar to the given training samples. Therefore, it would be beneficial to investigate into algorithms that facilitate classifiers to adapt to new areas. For example, by continuing training with a few labeled samples, or fully automated in an unsupervised manner.

In the case of multi-class aerial image labeling, our qualitative evaluation revealed that the predicted object boundaries are not well aligned to the contours given in the input image. Since classes like buildings have typical outlines, future work could also comprise adding constraints to the contours of segmented objects.

In addition, aerial images are typically captured with an overlap, therefore, exploiting this redundancy could be advantageous for semantic segmentation too.

Bibliography

[1] Björn Fröhlich, Erik Rodner, and Joachim Denzler. Semantic segmentation with millions of features: Integrating multiple cues in a combined random forest approach. In *Computer Vision–ACCV 2012*, pages 218–231. Springer, 2013.

[2] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[3] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *Computer Vision–ECCV 2014*, pages 297–312. Springer, 2014.

[4] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *International Conference on Learning Representations (ICLR)*, 2015.

[6] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip Torr. Conditional random fields as recurrent neural networks. In *International Conference on Computer Vision (ICCV)*, 2015.

[7] Jifeng Dai, Kaiming He, and Jian Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *International Conference on Computer Vision (ICCV)*, 2015.

[8] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *International Conference on Computer Vision (ICCV)*, 2015.

[9] Neil R Carlson. *Physiology of behavior*. Allyn & Bacon, 1986.

[10] Pedro O Pinheiro and Ronan Collobert. From image-level to pixel-level labeling with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1713–1721, 2015.

[11] George Papandreou, Liang-Chieh Chen, Kevin Murphy, and Alan L Yuille. Weakly- and semi-supervised learning of a dcnn for semantic image segmentation. In *International Conference on Computer Vision (ICCV)*, 2015.

[12] Antonio Torralba and Alexei Efros. Unbiased look at dataset bias. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1521–1528. IEEE, 2011.

[13] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[14] Li Deng and Dong Yu. Deep learning: Methods and applications. Technical report, May 2014.

[15] J Schmidhuber. Deep learning in neural networks: An overview. *Neural networks: the official journal of the International Neural Network Society*, 61:85–117, 2014.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision (ICCV)*, 2015.

[19] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[20] Ruslan Salakhutdinov, Joshua B Tenenbaum, and Antonio Torralba. Learning with hierarchical-deep models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1958–1971, 2013.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[23] Pablo Arbelaez, Jordi Pont-Tuset, Jonathan Barron, Ferran Marques, and Jagannath Malik. Multiscale combinatorial grouping. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 328–335. IEEE, 2014.

[24] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[25] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in Neural Information Processing Systems*, pages 109–117, 2011.

[26] Volodymyr Mnih. *Machine Learning for Aerial Image Labeling*. PhD thesis, University of Toronto, 2013.

[27] Shunta Saito and Yoshimitsu Aoki. Building and road detection from large aerial imagery. In *IS&T/SPIE Electronic Imaging*, pages 94050K–94050K. International Society for Optics and Photonics, 2015.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, pages 1–42, 2015.

[29] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[30] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, pages 79–86, 1951.

[31] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[32] Andrej Karpathy. CS231n: convolutional neural networks for visual recognition. `http://cs231n.github.io/neural-networks-1/`, accessed on 2015-11-25.

[33] Fan Hu, Gui-Song Xia, Jingwen Hu, and Liangpei Zhang. Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery. *Remote Sensing*, 7(11):14680, 2015.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. 2015.

[35] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[36] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

[37] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[38] Ali S Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

[39] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328, 2014.

[40] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *Computer Vision–ECCV 2014*, pages 345–360. Springer, 2014.

[41] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pages 2654–2662, 2014.

[42] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[43] Stephane Mallat. *A wavelet tour of signal processing: the sparse way.* Academic press, 2008.

[44] Renfrey Burnard Potts. Some generalized order-disorder transformations. In *Mathematical proceedings of the cambridge philosophical society*, volume 48, pages 106–109. Cambridge Univ Press, 1952.

[45] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[46] Sebastian Nowozin and Christoph H Lampert. Structured learning and prediction in computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 6 (3–4):185–365, 2011.

[47] Solomon Eyal Shimony. Finding maps for belief networks is np-hard. *Artificial Intelligence*, 68(2):399–410, 1994.

[48] DM Greig, BT Porteous, and Allan H Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 271–279, 1989.

[49] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2): 183–233, 1999.

[50] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2): 1–305, 2008.

[51] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast high-dimensional filtering using the permutohedral lattice. In *Computer Graphics Forum*, volume 29, pages 753–762. Wiley Online Library, 2010.

[52] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[53] Anuj R Shah, Christopher S Oehmen, and Bobbie-Jo Webb-Robertson. Svm-hustle—an iterative semi-supervised machine learning approach for pairwise protein remote homology detection. *Bioinformatics*, 24(6):783–790, 2008.

[54] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, et al. Semi-supervised learning. 2006.

[55] Aarti Singh, Robert Nowak, and Xiaojin Zhu. Unlabeled data: Now it helps, now it doesn't. In *Advances in neural information processing systems*, pages 1513–1520, 2009.

[56] Xiaojin Zhu. Semi-supervised learning literature survey. 2005.

[57] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using em. *Machine learning*, 39 (2-3):103–134, 2000.

[58] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *ICML Workshop*, 2013.

[59] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536, 2004.

[60] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2014.

[61] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 991–998. IEEE, 2011.

[62] Christian Wiedemann, Christian Heipke, Helmut Mayer, and Olivier Jamet. Empirical evaluation of automatically extracted road axes. In *Empirical Evaluation Techniques in Computer Vision*, pages 172–187, 1998.

[63] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. In *Advances in Neural Information Processing Systems*, 2014.

[64] Borja Rodríguez-Cuenca and Maria C Alonso. Semi-automatic detection of swimming pools from aerial high-resolution images and lidar data. *Remote Sensing*, 6(4):2628–2646, 2014.

[65] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[66] Yujia Li and Rich Zemel. High order regularization for semi-supervised learning of structured output problems. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1368–1376, 2014.

[67] Ziwei Liu, Xiaoxiao Li, Ping Luo, Chen Change Loy, and Xiaoou Tang. Semantic image segmentation via deep parsing network. In *International Conference on Computer Vision (ICCV)*, 2015.