

Technische Universität Dresden
Fakultät Informatik
Institut für Künstliche Intelligenz
Lehrstuhl Computer Vision

Diplomarbeit

Semantische Segmentierung historischer topographischer Karten

Daniel Schemala
Mat.-Nr. 3221669

14. März 2016

Betreuer: Dr. Dmytro Shlezinger
Verantwortlicher Hochschullehrer: Prof. Ph.D. Carsten Rother

Aufgabenstellung der Diplomarbeit

Entwicklung eines Verfahrens zur automatischen Segmentierung topographischer Pläne

Das Ziel der Arbeit ist es, ein Modell zur Segmentierung topographischer Pläne auf semantische Regionen zu entwickeln. Das Modell besteht dabei aus zwei Teilen. Zum einen ist das ein auf Entscheidungsbäumen basierender Klassifikator, der für jedes Pixel des zu bearbeitenden Bildes die Aussagen über die Segmentzugehörigkeit dieses Pixels liefert. Die Ergebnisse des Klassifikators dienen als Ausgangsdaten für ein darüberliegendes Conditional Random Field (CRF).

Sind alle Modellparameter bekannt, besteht die Inferenzaufgabe darin, jedem Pixel eine semantische Bedeutung (Segmentlabel) zuzuordnen. Dabei sind zwei Varianten zu untersuchen: Maximum-A-posteriori-Entscheidung (auch bekannt unter dem Namen Energieminimierungsproblem) und Maximum-Marginal- Entscheidung. Diese zwei Varianten sind miteinander bezüglich der erreichten Qualität der Lösung sowie der Zeitkomplexität zu vergleichen.

Für das Lernen sind zwei Vorgehensweisen zu untersuchen. Zum einen, dass die Entscheidungsbäume zunächst unabhängig vom dem darüberlegenden CRF gelernt werden. Zum anderen, dass beide Bestandteile zusammen gelernt werden. Die erste Variante kann dabei der Ausgangspunkt für die zweite sein. Beide Varianten sind miteinander bezüglich der erreichten Qualität sowie bezüglich der Robustheit zu vergleichen.

Die Arbeit erfolgt in enger Kooperation mit dem Institut für ökologische Raumentwicklung, der Ansprechpartner ist Herr Hendrik Herold. Die für die Durchführung der Arbeit benötigten Daten werden von Herrn Herold zur Verfügung gestellt.

Betreuer:	Dr. Dmytro Shlezinger
Verantwortlicher Hochschullehrer:	Prof. Ph.D. Carsten Rother
Beginn am:	15. 07. 2015
Einzureichen am:	15. 03. 2016

Zusammenfassung

Ziel dieser Arbeit ist es, eine Methode für die semantische Segmentierung topographischer Karten zu entwickeln, die für den Aufbau eines geographischen Informationssystems verwendet werden soll. Das Hauptaugenmerk lag auf der Aufgabe, Scans historischer Messtischblätter in Siedlungsgebiete und Nicht-Siedlungsgebiete zu unterteilen. Als Modell wurde ein Conditional Random Field (CRF) verwendet, dessen Unary Potentials aus einem Random Forest stammen. Die Methode wurde unter Verwendung zweier Inferenzalgorithmen (Graph Cut und Gibbs Sampling) implementiert und getestet.

Abstract

A method for the semantic segmentation of topographic maps was developed for use in a geographical information system. The main focus was on the task to segment scans of historical maps into settlement and non-settlement area. The model used is a Conditional Random Field (CRF) which obtained its unary potentials from a Random Forest. The method was implemented and tested using two inference algorithms (Graph Cut and Gibbs Sampling).

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Dresden, 14. März 2016

Inhaltsverzeichnis

1. Einleitung	1
1.1. Anforderungen an die Software	2
1.2. Einordnung	2
1.2.1. Überwachte Segmentierung	3
1.2.2. Unüberwachte Segmentierung	3
1.2.3. Semiüberwachte Segmentierung	3
2. Grundlagen	4
2.1. Random Forests	5
2.1.1. Modell	5
2.1.2. Inferenz	6
2.1.3. Training	6
2.2. Graphische Modelle	10
2.2.1. Gerichtete graphische Modelle	10
2.2.2. Ungerichtete graphische Modelle	11
3. Implementierung	21
3.1. Benutzung	21
4. Evaluation	23
4.1. Datensatz	23
4.2. Experimente	24
4.2.1. Das Maß für die Testleistung	24
4.2.2. Parameter für das Training lernen	25
4.2.3. Generalisierung des Trainings	27
4.2.4. Vergleich der Inferenzarten	27
4.2.5. Generalisierung der Inferenz	28
4.3. Geschwindigkeit	29
4.4. Diskussion	30
5. Zusammenfassung und Ausblick	38
A. Anhang	39
Literaturverzeichnis	45
Abbildungsverzeichnis	48

Tabellenverzeichnis	49
Algorithmenverzeichnis	49

1. Einleitung

Ziel dieser Arbeit ist es, eine Methode für die semantische Segmentierung historischer Landkarten zu entwickeln, die für den Aufbau eines geographischen Informationssystems verwendet werden soll.

Geographische Informationssysteme (GIS) dienen zur Erfassung, Analyse und Visualisierung geographischer Objekte und ihrer räumlichen Zusammenhänge. Sie werden für die Planung und Verwaltung von natürlichen Ressourcen und Infrastrukturen, im Verkehrswesen, der Kartographie und anderen Bereichen verwendet [Bra90]. Insbesondere die Analyse von Siedlungsgebiet und Bausubstanz hat wichtige Anwendungen wie die Planung von Verkehrswegen und Infrastruktur oder die Erstellung von Gefahrenkarten z. B. bei Naturkatastrophen [MHH09].

Auch aus historischen Karten lassen sich vielfältige interessante Informationen gewinnen, die beispielsweise bei der Landschaftsplanung, Naturschutzplanung oder für langfristiges Monitoring der Landnutzung eingesetzt werden [NW02].

Diese Arbeit entstand in enger Zusammenarbeit mit dem Leibniz-Institut für ökologische Raumentwicklung (IÖR). Dort werden für den Aufbau eines geographischen Informationssystems Geodaten erfasst und modelliert. Eine Anwendung ist der IÖR-Monitor¹, der die Entwicklung von Siedlung, Freiraum, Bevölkerung, Naturschutz und Verkehr für Deutschland visualisiert. Dafür werden hauptsächlich die Daten aus dem Amtlichen Topographisch-Kartographischen Informationssystem (ATKIS) verwendet. In Zukunft sollen aber auch ältere topographische Karten integriert werden [Mei09].

Bei diesen Karten handelt es sich um *Messtischblätter*. Das sind topographische Karten im Maßstab 1 : 25.000, daher auch die Bezeichnung TK25. Sie wurden ab etwa 1875 bis 1945 für ganz Deutschland erstellt.

Höchaufgelöste Scans dieser Landkarten müssen dazu georeferenziert, d. h. mit geographischen Informationen versehen werden sowie bestimmte Gebietsarten darin lokalisiert werden. Bislang werden nach dem Trial-and-Error-Verfahren Bildverarbeitungsmethoden mit verschiedenen Filtern verwendet [HRHM11]. Dazu ist viel Handarbeit nötig. Deshalb sollte nun eine Lösung entwickelt werden, die Teile dieser Arbeit zumindest unterstützt oder sogar ganz automatisiert.

Außerdem werden bisher die einzelnen graphischen Elemente, die das zu analysierende Gebiet ausmachen, unabhängig voneinander identifiziert. Beispielsweise werden, um Siedlungsgebiete zu finden, Filter für Schraffur, Filter für Gebäude, Filter für Wege usw. angewandt und die Ergebnisse kombiniert. Hier dagegen sollen, als Neuerung in diesem Anwendungsgebiet, komplexe, zusammengesetzte Elemente wie Siedlungsgebiet erkannt werden.

¹<http://www.ioer-monitor.de/>

Im Rest des Kapitels werden die Anforderungen an die praktische Lösung formuliert und ein Überblick über Bildsegmentierung gegeben. In Kapitel 2 folgt die Beschreibung der Theorie hinter der hier verwendeten Methode und in Kapitel 3 etwas zur praktischen Umsetzung. In Kapitel 4 wird diese evaluiert und Kapitel 5 schließt die Arbeit mit einer Zusammenfassung und Diskussion ab.

1.1. Anforderungen an die Software

Es soll eine eigenständig lauffähige Bibliothek für die überwachte Segmentierung von topographischen Karten entwickelt werden, die möglichst einfach und ohne viel Vorwissen über Computer Vision benutzbar ist. Sie soll Bilder in „interessante“ und „uninteressante“ Regionen unterteilen. In den hier verwendeten Testdaten waren Siedlungen diese interessanten Gebiete, die im Bild lokalisiert werden sollten. Eventuell sollen später auch andere Arten von Regionen lokalisiert werden.

Die Anforderungen an die Software waren:

- Bearbeitung großer monochromer digitaler Bilder
- im Idealfall soll anhand weniger von Hand segmentierter Bilder nur einmal ein Klassifikator gelernt werden und die Inferenz mit allen anderen Karten effizient erfolgen
- in C++ geschrieben und mit möglichst wenigen Abhängigkeiten nativ unter Windows und Linux lauffähig
- Python-Wrapper, um die Software in den vorhandenen Workflow integrieren zu können
- eventuell Training mit OpenMP parallelisierbar

1.2. Einordnung

Bildsegmentierung gehört zu den ältesten und am meisten untersuchten Problemen in der Computer Vision [Sze10, S. 269]. Dementsprechend haben sich viele verschiedene Vorgehensweisen entwickelt. Angewandt wird Bildsegmentierung z. B. als Vorverarbeitungsschritt bei *Content-based image retrieval* [SWS⁺00], Videokompression [IPV00], Objekterkennung [BKH⁰⁰] oder bei bildgebenden Verfahren in der Medizin [PXP00].

Bei der Segmentierung soll ein Bild in Regionen unterteilt werden, die (in den meisten Anwendungsfällen) möglichst die im Bild vorhandenen Objekte abdecken sollen. Da jedem Pixel anhand seiner Lage und seinen Merkmalen eine Region zugeordnet werden soll, entspricht Segmentierung einer Klassifikationsaufgabe im Maschinellen Lernen. Man kann Bildsegmentierung daher, ebenso wie Klassifikation, grob einteilen in überwacht, unüberwacht und semiüberwacht.

1.2.1. Überwachte Segmentierung

Es ist eine Menge von Klassen vorgegeben, die i. A. realen Objekten entsprechen und es stehen Trainingsdaten in Form von bereits segmentierten Bildern zur Verfügung. Aus diesen Trainingsbildern wird ein Modell gelernt, das beschreibt, auf welche Weise neue Bilder segmentiert werden sollen.

Ein beliebtes Framework für überwachte Segmentierung² ist, einerseits Pixel anhand ihrer Merkmale individuell zu klassifizieren und dann in einem zweiten Schritt die räumlichen Zusammenhänge der Labels zu modellieren.

Für die Klassifikation bietet das Maschinelle Lernen viele Methoden, z. B. *Logistische Regression* (z. B. in [RM03]) oder Entscheidungsbäume [NRB⁺11]. In den letzten Jahren wurden sehr erfolgreich Neuronale Netze eingesetzt, siehe [LSR⁺15] für ein aktuelles Beispiel. In dieser Arbeit werden *Random Forests* verwendet.

Die räumlichen Zusammenhänge lassen sich mit *Markov Random Fields* oder *Conditional Random Fields* modellieren. Auch für die Inferenz in diesen Modellen wurden viele verschiedene Methoden entwickelt, z. B. ICM [Bes86], Simulated Annealing [Bar89], Graph Cut [BVZ01] oder Loopy Belief Propagation [YFW00].

1.2.2. Unüberwachte Segmentierung

Hier gibt es keine vorgegebenen Klassen, sondern das System muss diese ohne Trainingsbilder selbst finden. Dazu werden Pixel anhand ihrer Ähnlichkeit, Nähe, Textur oder anderen Merkmalen gruppiert.

Es gibt kantenorientierte (z. B. [KWT88]), regionenorientierte [FH04] und pixelorientierte [CM02] Segmentierungsverfahren.

1.2.3. Semiüberwachte Segmentierung

Hier wird (oft interaktiv) vom Benutzer mit breiten Strichen oder einem umschließenden Rechteck ein Objekt in einem Bild markiert. Aus den Pixeln in der Nähe der Striche wird ein Modell gelernt, mit dem die übrigen Pixel klassifiziert werden. In [RKB04] wird das iterativ fortgesetzt bis die fertige Segmentierung erreicht ist.

²Die gleichen Methoden werden aber auch erfolgreich für andere Anwendungen in der Computer Vision eingesetzt, z. B. Bildwiederherstellung [Bes86] oder Stereo matching [Bar89].

2. Grundlagen

Bei der vorliegenden Aufgabe der semantischen Segmentierung von Landkarten soll jedem Pixel eines Eingabebildes genau eine der beiden Klassen *interessantes Gebiet* (*IG*) und *uninteressantes Gebiet* (*UG*) zugeordnet werden. Mit *interessantes Gebiet* ist das Gebiet gemeint, das im Bild lokalisiert werden soll und mit *uninteressantes Gebiet* alles, was nicht dazu gehört. Im Rest der Arbeit wird immer davon ausgegangen, dass diese interessanten Gebiete Siedlungen sind, auch wenn die Methoden natürlich auch auf andere Arten von Gebieten anwendbar sind.

Da bekannt ist, dass realistische Siedlungen nicht einzelne Pixel umfassen, die beliebig über eine Karte verstreut sind, sondern dass sie größere, zusammenhängende Flächen ergeben, ist es sinnvoll anzunehmen, dass ein Pixel tendenziell Siedlung ist, wenn viele andere Pixel in der Nähe Siedlung sind und umgekehrt. Dieser Sachverhalt soll bei der Segmentierung beachtet werden.

Man kann die vorliegende Aufgabe innerhalb des *Maschinellen Lernens* als *Structured-Prediction*-Problem betrachten [SM11, S. 267 ff.]. Structured Prediction ist eine Erweiterung der *Klassifikation*. Das Ziel bei einer Klassifikationsaufgabe ist, basierend auf beobachteten Merkmalen, einer Variablen einen Wert aus einer Menge von Klassen zuzuordnen. Bei Structured Prediction dagegen soll für viele Variablen die Klasse vorausgesagt werden, wobei außerdem bestimmte Beziehungen zwischen den Belegungen dieser Variablen beachtet werden müssen. Die Motivation dahinter ist, dass in vielen Domänen die Variablen nicht unabhängig voneinander existieren, sondern es gibt zusätzliches domänenspezifisches Wissen über die Verteilung der Variablenbelegungen.

Eine klassische Anwendung für Structured Prediction ist *Part-of-speech Tagging*, ein Problem der Verarbeitung natürlicher Sprachen. Hier ist die Aufgabe, jedem Wort in einem geschriebenen Satz die Wortart zuzuordnen. Variablen wären hier die einzelnen Wörter, Klassen die Menge der Wortarten und domänenspezifische Beziehungen wären hier beispielsweise, dass in einem deutschen Satz relativ selten zwei Substantive aufeinanderfolgen.

Zur Lösung eines Structured-Prediction-Problems kann man eine Kombination aus Klassifikation und *Graphischen Modellen* verwenden. Graphische Modelle sind eine Familie von Methoden zur Modellierung von multivariaten Daten und Beziehungen zwischen ihnen. Für die vorliegende Aufgabe der semantischen Segmentierung wurden Random Forests für die Klassifikation und Conditional Random Fields als Graphisches Modell verwendet. Die beiden Methoden werden im Folgenden beschrieben.

2.1. Random Forests

Random Forests [Bre01] (in der Literatur oft auch als *Decision Forests* bezeichnet) sind eine gut untersuchte Klassifikationsmethode, die im Allgemeinen einfach und schnell ist und gute Ergebnisse liefert. Sie ist nicht auf lineare Klassifikation beschränkt, sondern kann beliebig komplexe Funktionen lernen. Außerdem sind Random Forests in der Lage, nicht nur die absolute Zugehörigkeit zu einer Klasse vorherzusagen, sondern auch den Grad der Sicherheit dieser Zuordnung [CS13, S. 30].

Es gibt je nach Anwendungsfall viele, leicht verschiedene Varianten von Random Forests. Hier werden Modell, Inferenz und Training von Random Forests für die Klassifikation von Pixeln in Bildern in genau zwei Klassen beschrieben. Die Beschreibung folgt im Großen und Ganzen [CS13, S. 25 ff.].

Ziel ist, einen Random Forest aus bereits segmentierten Trainingsbildern zu lernen, der anschließend in zuvor noch nicht gezeigten Bildern für jedes Pixel den Grad der Zugehörigkeit zu den Klassen *interessantes Gebiet* (*IG*) oder *uninteressantes Gebiet* (*UG*) vorhersagt. Das ganze passiert aufgrund der Merkmale des Pixels. Merkmale sind der Grauwert des Pixels selbst, sowie die Grauwerte seiner Nachbarpixel innerhalb eines festen Radius'.

2.1.1. Modell

Ein *Random Forest* ist eine Menge von t unabhängig voneinander generierten *Entscheidungsbäumen*.

Ein *Entscheidungsbau* ist ein binärer Baum mit der maximalen Tiefe d . Jeder innere Knoten ist ein *Testknoten* i , der eine binäre Testfunktion $test_i: \{Pixel\} \rightarrow \{L, R\}$ repräsentiert. Diese Testfunktionen bildet also jedes Pixel entweder auf L oder R ab. Jeder Blattknoten b eines Entscheidungsbäums enthält eine Wahrscheinlichkeitsverteilung $p_b(k)$ der Klassen $\{IG, UG\}$.

Die Testfunktionen, die Pixeln aufgrund ihrer Merkmale entweder L oder R zuordnen, können beliebig komplizierte Funktionen sein. Eine ganz simple Variante wäre die Wahl

$$test_i(p) = \begin{cases} L & \text{wenn } I(p) < s_i \\ R & \text{sonst} \end{cases} \quad (2.1)$$

d. h. die Funktion ergibt L genau dann, wenn die Helligkeit $I(p)$ des Eingabepixels kleiner als ein Schwellwert s_i ist, der dieser Testfunktion zugeordnet ist.

Weil aber alleine die Helligkeit nicht ausreicht, ein einzelnes Pixel zu klassifizieren, sollen auch Pixel in dessen unmittelbarer Umgebung betrachtet werden. Deswegen wird in dieser Arbeit eine einfache und beliebte Variante (siehe z. B. [GL13, S. 149]) verwendet:

$$test_i(p) = \begin{cases} L & \text{wenn } I(p + o_i) - I(p + u_i) < s_i \\ R & \text{sonst} \end{cases} \quad (2.2)$$

Hier wird getestet, ob die Differenz der Helligkeitswerte zweier Nachbarpixel (repräsentiert durch die Offsets o_i und u_i) kleiner oder größer als ein Schwellwert ist. Die Offsets und der Differenz-Schwellwert s_i sind in jedem Testknoten anders.

2.1.2. Inferenz

Ein Entscheidungsbaum bildet Pixel auf eine Wahrscheinlichkeitsverteilung der Klassen ab. Ein Random Forest kombiniert einfach die Resultate seiner Entscheidungsbäume.

Die Inferenz mit einem Entscheidungsbaum läuft so ab, dass beginnend mit dessen Wurzelknoten die durch ihn repräsentierte Testfunktion auf das Eingabepixel angewandt wird. Wenn sie L ergibt, wird das Pixel an die Wurzel des linken Teilbaums weitergereicht, ansonsten an die Wurzel des rechten Teilbaums. Dort wird das Pixel auf die gleiche Weise behandelt. Das geht so lange, bis ein Blattknoten erreicht ist. Die Wahrscheinlichkeitsverteilung an diesem Knoten wird als Ergebnis der Inferenz zurückgegeben. Sie gibt eine Schätzung für die Wahrscheinlichkeit an, dass das Eingabepixel zu IG oder UG gehört.

Bei der Inferenz mit einem Random Forest wird die Inferenz mit jedem darin enthaltenen Entscheidungsbaum durchgeführt und das Ergebnis auf irgendeine Weise gemittelt. Gebräuchliche Methoden dafür sind Addition oder Multiplikation mit anschließender Normalisierung: Sei $p_i(k)$ die resultierende Verteilung des Entscheidungsbaums i und t die Anzahl der Entscheidungsbäume. Dann ist das Ergebnis des Random Forests

$$p(k) = \frac{1}{t} \sum_{i=1}^t p_i(k) \quad (2.3)$$

oder

$$p(k) = \frac{\prod_{i=1}^t p_i(k)}{\sum_{k'} \prod_{i=1}^t p_i(k')} \quad (2.4)$$

In dieser Arbeit wird Addition (2.3) verwendet.

Random Forests kann man daher als einfache Variante des Ensemblelernens betrachten, weil hier mehrere Entscheidungsbäume gelernt und kombiniert werden, um die Leistung zu verbessern. Diese Art des Ensemblelernens wird als *Bagging* bezeichnet [Bre96].

Die Vorgehensweise bei der Inferenz ist in Algorithmus 1 auf Seite 7 zusammengefasst.

2.1.3. Training

Generiert werden die Entscheidungsbäume aus Trainingsbildern, also aus einer Menge von Pixeln, von denen die Klasse IG oder UG bereits bekannt ist. Man hofft, dass die so erzeugten Entscheidungsbäume dann auch Pixel, die sich nicht in der Trainingsmenge befanden, korrekt klassifizieren.

Das Training kann über mehrere Parameter gesteuert werden:

Algorithmus 1 Inferenz in Random Forests

```

function RANDOM-FOREST-INFERENZ(pixel, forest)
    p  $\leftarrow$  leere Wahrscheinlichkeitsverteilung
    for all i  $\in$  forest do
        pi  $\leftarrow$  Entscheidungsbaum-Inferenz(pixel, i)
        p  $\leftarrow$  p + pi
    end for
    p  $\leftarrow$  Normalisieren(p)
    return p
end function

function ENTSCHEIDUNGSBAUM-INFERENZ(pixel, baum)
    knoten  $\leftarrow$  Wurzelknoten(baum)
    if knoten ist Blattknoten then
        p  $\leftarrow$  Verteilung(knoten)
        return p
    end if
    if Testfunktion(knoten)(pixel) = L then
        return Entscheidungsbaum-Inferenz(pixel, Linker-Teilbaum(knoten))
    else
        return Entscheidungsbaum-Inferenz(pixel, Rechter-Teilbaum(knoten))
    end if
end function

```

- die Anzahl t der Entscheidungsbäume in einem Random Forest
- die maximale Baumtiefe d
- die Anzahl p der Samples beim Generieren eines Testknotens

Die Auswirkungen davon werden in Abschnitt 4.2.2 untersucht.

Die Bäume eines Random Forests werden unabhängig voneinander je auf einem Teil der Trainingsdaten trainiert, der aus der Gesamtmenge gesampelt wird:

- Da die vorliegenden Kartenbilder wesentlich weniger Siedlungs- als Nicht-Siedlungsgebiete enthalten, werden alle Siedlungspixel für das Training verwendet
- außerdem alle Nicht-Siedlungspixel innerhalb eines Radius' von 15 Pixeln um Siedlungen
- von den übrigen Nicht-Siedlungspixeln werden so viele zufällig gewählt, dass am Ende gleich viele Siedlungs- wie Nichtsiedlungspixel in der Trainingsmenge sind.

Beim Lernen eines Entscheidungsbaums werden der Reihe nach von der Wurzel bis zu den Blättern Testknoten erzeugt, die so gewählt sind, dass sie die an diesem Knoten

ankommenden Trainingsdaten möglichst gut aufteilen. „Gut aufteilen“ bedeutet hier, dass die Testfunktion für die meisten oder gar alle IG -Pixel aus diesen Trainingsdaten L und für die meisten UG -Pixel R ergibt oder umgekehrt. Ist die Maximaltiefe d erreicht oder teilt ein Testknoten alle ankommenden Trainingsbeispiele perfekt auf, werden Blattknoten erzeugt, denen die Klassenverteilung der an diesem Blattknoten ankommende Trainingsbeispiele als (empirische) Wahrscheinlichkeitsverteilung zugeordnet wird.

Auf diese Weise entstehen im Idealfall Entscheidungsbäume, die einem Pixel eine der beiden Klassen mit hoher Sicherheit zuordnen.

Die Auswahl eines Testknotens erfolgt, indem zufällig p potentielle Testfunktionen erzeugt werden und diejenige übernommen wird, die die von der Wurzel aus an diesem Knoten ankommenden Trainingsbeispiele am besten aufteilt. Sei $G(p_1, p_2)$ ein Maß für die Gleichverteilung einer Verteilung mit zwei möglichen Ereignissen. Dann ist ein mögliches Maß für die Güte der Aufteilung eines Knotens die Differenz der Gleichverteilung der ankommenden Trainingsbeispiele und der erwarteten Gleichverteilung an den beiden Ausgängen des Knotens:

$$G\left(\frac{I}{I+U}, \frac{U}{I+U}\right) - \sum_{i \in \{R,L\}} \frac{I_i + U_i}{I+U} \cdot G\left(\frac{I_i}{I_i + U_i}, \frac{U_i}{I_i + U_i}\right). \quad (2.5)$$

Hier sind I und U die Anzahl der eingehenden IG - bzw. UG -Trainingsbeispiele und I_R, I_L, U_R, U_L analog die Anzahl der nach L bzw. R ausgehenden Trainingsbeispiele.

Eine mögliche Wahl für G ist die Entropie H einer Verteilung. Motivieren lässt sich das mit informationstheoretischen Überlegungen: Die Vorhersage eines Ereignisses v , das mit Wahrscheinlichkeit $p(v)$ eintritt hat einen Informationsgehalt von $-\log_2 p(v)$ bit [SW49]. Der erwartete Informationsgehalt bei mehreren Ereignissen v_1, \dots, v_n ist demzufolge

$$H(v_1, \dots, v_n) = \sum_{i=1}^n p(v_i) \cdot (-\log_2 p(v_i)) = -\sum_{i=1}^n p(v_i) \cdot \log_2 p(v_i). \quad (2.6)$$

Das bezeichnet man als die *Entropie* einer Verteilung.

Bevor ein neues Pixel von einem Testknoten bearbeitet wird, werden $H\left(\frac{I}{I+U}, \frac{U}{I+U}\right)$ bit Information benötigt, um es korrekt zu klassifizieren. Hier dient $\left(\frac{I}{I+U}, \frac{U}{I+U}\right)$ als Schätzung für die Verteilung für ein noch nicht klassifiziertes Pixel. Hat der Test für das Pixel den Ausgang L , werden noch $H\left(\frac{I_L}{I_L+U_L}, \frac{U_L}{I_L+U_L}\right)$ bit benötigt, analog für R . Im Mittel ergibt das $\sum_{i \in \{R,L\}} \frac{I_i + U_i}{I+U} \cdot H\left(\frac{I_i}{I_i + U_i}, \frac{U_i}{I_i + U_i}\right)$ bit Information, die nach dem Test gebraucht werden, um das Pixel zu klassifizieren, da $\frac{I_i + U_i}{I+U}$ eine Schätzung dafür ist, dass der Test i ergibt. Die Schätzung für den durchschnittlichen Informationsgewinn durch den Testknoten ist also identisch mit dem Term (2.5) mit $G = H$.

Algorithmus 2 fasst das Training von Random Forests als Pseudoquelltext zusammen.

Algorithmus 2 Training von Random Forests

```

function RANDOM-FOREST-TRAINING(trainingsbeispiele, t, d, p)
    forest  $\leftarrow \emptyset$ 
    loop t mal
        tb  $\leftarrow$  Sampeln(trainingsbeispiele)
        baum  $\leftarrow$  Entscheidungsbaum-Training(tb, d, p, 0)
        forest  $\leftarrow$  forest  $\cup \{\text{baum}\}$ 
    end loop
    return forest
end function

function ENTSCHEIDUNGSBAUM-TRAINING(tb, d, p, aktuelle_tiefe)
    if aktuelle_tiefe = d or tb enthält Beispiele aus genau einer Klasse then
        return Blattknoten-Erstellen(tb)
    else
        potentielle_testknoten  $\leftarrow \bigcup_{i=0}^p \{\text{zufälliger Testknoten}\}$ 
        neuer_testknoten  $\leftarrow \operatorname{argmax}_{\text{knoten} \in \text{potentielle\_testknoten}} \text{Güte}(\text{knoten})$   $\triangleright (2.5)$ 
        tbL, tbR  $\leftarrow$  Aufteilen(tb, neuer_testknoten)
        baumL  $\leftarrow$  Entscheidungsbaum-Training(tbL, d, p, aktuelle_tiefe + 1)
        füge baumL als linken Kindknoten zu neuer_testknoten hinzu

        baumR  $\leftarrow$  Entscheidungsbaum-Training(tbR, d, p, aktuelle_tiefe + 1)
        füge baumR als rechten Kindknoten zu neuer_testknoten hinzu

        return neuer_testknoten
    end if
end function

```

2.2. Graphische Modelle

Ein gelernter Random Forest kann bereits jetzt zur Segmentierung eines Bildes genutzt werden: Für jedes Pixel wird Algorithmus 1 ausgeführt und man erhält eine Schätzung der Wahrscheinlichkeit, dass es sich um ein *IG*-Pixel handelt. Ist dieser Wert höher als ein Schwellwert, (z. B. der feste Wert 0,5), ordnet man dem Pixel *IG* zu, ansonsten *UG*. Das führt aber im Allgemeinen und besonders bei den hier betrachteten Landkarten zu schlechten Ergebnissen (siehe Abbildung 4.9a auf Seite 33 für ein Beispiel). Das liegt daran, dass unser Vorwissen, dass *IG*-Pixel größere, zusammenhängende Gebiete bilden, außer Acht gelassen wird. Dieses Vorwissen soll mit einem Graphischen Modell modelliert werden, deshalb schließt sich an die Random-Forest-Inferenz als zweiter Schritt die Inferenz mit einem *Conditional Random Field* an, das die Ausgabe des Random Forests verwendet.

Ein *Graphisches Modell* stellt die Wahrscheinlichkeitsverteilung einer Menge Y von Zufallsvariablen als Produkt einer Menge von *Faktoren* (auch als *Faktorfunktion* oder *Potentialfunktionen* bezeichnet) dar statt beispielsweise als Tabelle der gemeinsamen Verteilung. Eine solche Tabelle hätte bei binären Zufallsvariablen $2^{|Y|}$ Einträge und wäre bei realistischen Problemgrößen damit unbenutzbar groß.

Der Begriff „Graphisches Modell“ kommt daher, dass sich durch die Faktorisierung die Struktur von Variablen und bedingten Abhängigkeiten zwischen ihnen für eine Domäne recht kompakt und intuitiv verständlich als Graph visualisieren lässt.

Zur Notation im Rest dieser Arbeit: Y bezeichne eine Menge von Zufallsvariablen und $Y_i \in Y$ die i te Variable. Die Domäne von Y_i sei \mathcal{Y}_i und $\mathcal{Y} = \times_{i=1}^{|Y|} \mathcal{Y}_i$. Eine Belegung der Variablen Y_i sei mit y_i und eine Belegung aller Variablen Y mit y bezeichnet. Gegebenenfalls gibt es auch die Variablenmenge X , die analog notiert wird.

Es lassen sich grob gerichtete von ungerichteten Graphischen Modellen unterscheiden.

2.2.1. Gerichtete graphische Modelle

Sie werden auch als *Bayessche Netze* bezeichnet.

Sei $G = (Y, E)$ ein gerichteter, kreisloser Graph über die Variablen Y mit $E \subset (Y \times Y)$. Ein *Gerichtetes graphisches Modell* beschreibt dann die Verteilung $p(Y)$ als Produkt von gegebenen *lokalen bedingten Verteilungen*:

$$p(y) = \prod_{i=1}^{|Y|} p(y_i \mid y_{\text{Vorg}_G(i)}). \quad (2.7)$$

Hier ist $y_{\text{Vorg}_G(i)}$ die Belegung der Vorgängerknoten des Knotens i in G . Wenn der Knoten i keinen Vorgänger hat, ist mit $p(y_i \mid y_{\text{Vorg}_G(i)})$ die unbedingte Wahrscheinlichkeit $p(y_i)$ gemeint.

Haben die Knoten in Y höchstens k Vorgänger, dann muss man für eine lokale Verteilung $p(y_i \mid y_{\text{Vorg}_G(i)})$ bei binären Variablen 2^k Werte speichern. Für das ganze Netz sind also höchstens $|Y| \cdot 2^k$ Zahlen erforderlich, was bei realistischen Netzen viel

weniger ist als die $2^{|Y|}$ für die vollständige gemeinsame Verteilung, da im Allgemeinen $k \ll |Y|$.

Bayessche Netze können verwendet werden, wo Variablen in einem Ursache-Wirkungs-Zusammenhang stehen, z. B. in [HBHH98]. Eine wichtige Spezialisierung von Bayesschen Netzen sind *Hidden-Markov-Modelle*, mit denen sich z. B. zeitliche Veränderungen modellieren lassen. Bayessche Netze werden oft für generative Modelle verwendet [SM11, S. 277].

2.2.2. Ungerichtete graphische Modelle

Sie werden auch als *Markov Random Fields* bezeichnet. Die Beschreibung hier folgt lose [SM11, S. 273 f.].

Sei $F = \{f_\phi : \mathcal{Y}_f \rightarrow \mathbb{R}_+ \}_{\phi=1}^m$ eine Menge von *Faktorfunktionen*. Eine Faktorfunktion f bildet jeweils eine Belegung y_f einer Teilmenge $Y_f \subseteq Y$ von Variablen auf einen Wert ≥ 0 ab. Ein *Ungerichtetes graphisches Modell* ist dann die Menge aller Wahrscheinlichkeitsverteilungen, die als

$$p(y) = \frac{1}{Z} \cdot \prod_{f \in F} f(y_f) \quad (2.8)$$

geschrieben werden können. Hier ist

$$Z = \sum_{y \in \mathcal{Y}} \prod_{f \in F} f(y_f) \quad (2.9)$$

die Normalisierungskonstante oder Partitionsfunktion, damit der Term $p(y)$ summiert über alle möglichen Belegungen y insgesamt 1 ergibt.

Im Gegensatz zu den lokalen Verteilungen in Bayesschen Netzen müssen die Faktorfunktionen nicht unbedingt eine Wahrscheinlichkeitsverteilung über die Y_f ergeben, sondern können ein beliebiges Maß von Kompatibilität einer Belegung sein. Sie sollten also beim Entwurf des Modells so gewählt werden, dass eine wahrscheinliche oder anderweitig gewünschte Belegung einen hohen Wert erzielt. Da eine Faktorfunktion f auch eine bedingte Wahrscheinlichkeitsverteilung sein kann, ist ersichtlich, dass gerichtete ein Spezialfall von ungerichteten graphischen Modellen sind.

Auch wenn jede Faktorfunktion prinzipiell eine beliebige Funktion (mit Wertebereich \mathbb{R}_+) sein kann, gibt es in der Praxis oft nur wenige verschiedene *Faktortypen* (auch als *Clique templates* bezeichnet), die die gleiche Funktion repräsentieren, nur auf unterschiedlichen Variablen operieren.

Günstig im Hinblick auf die Inferenz ist es, Faktorfunktionen der Form $f(y_f) = \exp(-E_f(y_f))$ zu verwenden. Die *Energiefunktionen* $E_f : \mathcal{Y}_f \mapsto \mathbb{R}$ kann man sich als „Strafe“ für unpassende Belegungen y vorstellen.

Statt Energiefunktionen beim Modellieren fest vorzugeben, kann man sie auch linear mit Gewichten ausstatten, die dann anhand von Trainingsbeispielen gelernt werden können. In [NRB⁺11] werden außerdem die Faktortypen gelernt. [SM11, S. 331 ff.] gibt einen Überblick über Lernmethoden.

Es gibt zwei verschiedene, wenn auch verwandte, Möglichkeiten, die Faktorisierung einer Verteilung $p(y)$ als Graph darzustellen:

Markov-Graph Sei $G = (Y, E)$ mit $E \subseteq (Y \times Y)$ ein ungerichteter Graph über den Variablen Y . Die Kanten repräsentieren bedingte Abhängigkeiten zwischen Variablen: Für zwei Variablen $Y_a, Y_b \in Y$ gilt $p(y_a | y_{N_G(a)}) = p(y_a | y_{N_G(a)}, y_b)$. Hier ist $y_{N_G(a)}$ die Belegung der mit Y_a benachbarten Variablen. Das bedeutet also, dass die Verteilung einer Variable Y_a nur von der Belegung ihrer Nachbarn in G abhängt und unabhängig von allen anderen ist (solange die Belegung der Nachbarn bekannt ist). Eine Faktorisierung wie in Gleichung (2.8) erreicht man, indem man für jede *maximale Clique* eine Faktorfunktion für alle Variablen in dieser Clique erstellt. Eine *Clique* ist eine untereinander vollverbundene Teilmenge von Y , d. h. eine Menge $C \subseteq Y$ von Variablen, sodass für alle $Y_a, Y_b \in C$ gilt, dass $(Y_a, Y_b) \in E$. Eine *maximale Clique* ist eine, die keine Teilmenge einer anderen Clique ist.

Faktorgraph Im Faktorgraph sind Faktoren als weitere Knoten und mit den betreffenden Variablen verbunden dargestellt. Ein *Faktorgraph* ist ein bipartiter Graph (Y, F, E) , wobei Y Variablen und F Faktoren sind und die Kanten $E \subseteq (F \times Y)$ einen Faktorknoten mit dessen Eingabevervariablen verbinden.

Markov-Graphen sind i. A. besser für die Modellierung eines Problems geeignet, während Faktorgraphen praktischer für die Beschreibung von Inferenzalgorithmen sind.

Conditional Random Fields

In den meisten Anwendungsfällen von Graphischen Modellen, insbesondere bei Structured Prediction, unterscheidet man bei den verwendeten Variablen zwischen mehreren Arten, die verschiedene Zwecke erfüllen:

- *Eingabevervariablen*, im Folgenden, wie in der Machine-Learning-Literatur üblich, mit der Menge X bezeichnet. Die Belegungen sind zum Zeitpunkt der Inferenz bekannt, z. B. aus Beobachtungen.
- *Ausgabevervariablen*, hier weiterhin mit Y bezeichnet. Die Belegungen sollen (bei gegebenem X) vorhergesagt werden.
- In manchen Anwendungen gibt es noch *latente (verborgene) Variablen*, deren Belegung unbekannt, aber für die Anwendung uninteressant ist. Sie werden ggf. zum Modellieren benötigt. Latente Variablen werden in dieser Arbeit nicht weiter behandelt.

Das Modell kann also als

$$p(x, y) = \frac{1}{Z} \cdot \prod_{f \in F} f(x_f, y_f) \quad (2.10)$$

geschrieben werden. Es modelliert die gemeinsame Verteilung von X und Y . Diese Art von Modellen wird als *generativ* bezeichnet. Im Gegensatz dazu modellieren *diskriminative* Modelle $p(y | x)$, was üblicherweise der aus Anwendersicht interessante Wert ist. Diese Modelle sind manchmal intuitiver, weil in generativen Modellen auch (eventuell implizit) die Verteilung der Eingabevervariablen modelliert werden muss, was nicht immer sinnvoll ist. Generative Modelle haben dafür den Vorteil, dass sie recht einfach mit latenten Variablen, und Eingabevervariablen, für die keine Beobachtungen zur Verfügung stehen, zurechtkommen. Für weitere Eigenschaften und Unterschiede von generativen und diskriminativen Modellen siehe [JN02].

Eine diskriminative Variante von Markov Random Fields sind *Conditional Random Fields (CRF)*: Seien X Eingabe- und Y Ausgabevervariablen und G ein Faktorgraph über $X \cup Y$. Dann ist (X, Y) ein *Conditional Random Field*, wenn für jede Belegung x von X (d. i. eine Beobachtung) die Verteilung $p(y | x)$ wie folgt mit G faktorisiert:

$$p(y | x) = \frac{1}{Z(x)} \cdot \prod_{f \in F} f(x_f, y_f). \quad (2.11)$$

Hier ist

$$Z(x) = \sum_{y \in \mathcal{Y}} \prod_{f \in F} f(x_f, y_f) \quad (2.12)$$

also im Unterschied zu Gleichung (2.9) wird nicht über die Belegungen für X summiert, weil sie fest vorgegeben ist.

Modell

Im Folgenden wird beschrieben, wie die vorliegende Aufgabe der Bildsegmentierung mit einem CRF modelliert wird.

Für jedes Pixel im zu segmentierenden Bild gibt es je eine Eingabe- und eine Ausgabevervariable. Der Wertebereich \mathcal{Y}_i der Ausgabevervariablen ist $\{IG, UG\}$ und der der Eingabevervariablen $\mathcal{X}_i = [0; 1]$ für alle i . Die Belegung x_i einer Eingabevervariablen ist die durch den Random Forest geschätzte Wahrscheinlichkeit $p(x_i = IG)$, dass es sich bei Pixel i um ein *IG*-Pixel handelt.

Es gibt zwei Faktortypen:

Datenterm (oder Unary Potentials) Die Faktoren F_1 , die je eine Eingabe- mit einer Ausgabevervariablen verknüpfen. Die Gleichung der $f \in F_1$ ist

$$f(x_f, y_f) = \begin{cases} x_f & \text{wenn } y_f = IG \\ 1 - x_f & \text{wenn } y_f = UG \end{cases} \quad (2.13)$$

Regularisierungsterm (oder Pairwise Potentials) Die Faktoren F_2 zwischen je zwei Ausgabevervariablen zweier benachbarter Pixel. Die Funktionen ergeben 1, wenn die beiden Belegungen übereinstimmen, ansonsten einen konstanten Wert $c \in [0; 1]$. Dieser Wert wird per Kreuzvalidierung mit Trainingsdaten gelernt. Es gilt also

für alle $f \in F_2$

$$f(x_f, y_{f,1}, y_{f,2}) = \begin{cases} 1 & \text{wenn } y_{f,1} = y_{f,2} \\ c & \text{sonst} \end{cases} \quad (2.14)$$

Intuitiv bewirken die beiden Faktortypen, dass bei der Berechnung der günstigsten Belegung y^* zwei Kriterien miteinander konkurrieren: Der Datenterm bewirkt, dass die AusgabevARIABLEN tendenziell den Zustand einnehmen, den der Random Forest vorgibt und der Regularisierungsterm, dass benachbarte Variablen einander angleichen. Je kleiner c , desto mehr treten die Faktoren F_2 in den Vordergrund, wodurch das Segmentierungsergebnis glatter und einheitlicher aussieht. Bei $c = 0$ würden alle Y den gleichen Wert annehmen, bei $c = 1$ würde dagegen gar keine Glättung stattfinden und jedes Y_i den Wert IG annehmen, wenn das korrespondierende $x_i \geq 0,5$.

Inferenz

Natürlich soll, sobald eine Beobachtung x vorliegt, mithilfe des Modells aus dem vorigen Abschnitt die beste Belegung der AusgabevARIABLEN Y berechnet werden, um das endgültige Segmentierungsergebnis zu erhalten. Dabei gibt es mehrere Möglichkeiten für eine „beste“ Belegung:

Maximum-A-Posteriori-Inferenz (MAP) Die beste Belegung y^* ist die mit der höchsten Wahrscheinlichkeit für eine gegebene Beobachtung x :

$$y^* = \operatorname{argmax}_{y \in \mathcal{Y}} p(y | x). \quad (2.15)$$

Maximum-Marginal-Inferenz (MMI) (auch als *Probabilistic Inference* bezeichnet). Die beste Belegung einer einzelnen Variablen y_i^* ist die mit der höchsten marginalen Wahrscheinlichkeit unter der Beobachtung x :

$$y_i^* = \operatorname{argmax}_{y_i \in \mathcal{Y}_i} p(y_i | x). \quad (2.16)$$

Motivieren lassen sich beide Inferenzarten mit Überlegungen aus der statistischen Entscheidungstheorie: Sei $p(y | x)$ die berechnete Wahrscheinlichkeit für y bei gegebenem x , von der wir annehmen, dass sie die wahre Wahrscheinlichkeit ausreichend gut annähert. Sei $\gamma: \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ eine Gewinnfunktion, wobei $\gamma(y, y')$ den Gewinn für die Vorhersage y bei wahrer Belegung y' ergibt. Sei außerdem $f: \mathcal{X} \mapsto \mathcal{Y}$ eine Entscheidungsfunktion, die ein y aus einem gegebenen x vorhersagt.

Der erwartete Gewinn Γ bei einer Beobachtung x , einer Entscheidung f und einer Gewinnfunktion γ ist

$$\Gamma(x, f, \gamma) = \sum_{y \in \mathcal{Y}} p(y | x) \cdot \gamma(f(x), y). \quad (2.17)$$

Eine mögliche sinnvolle Gewinnfunktion ist einfach die binäre Gewinnfunktion

$$\gamma_b(y, y') = \begin{cases} 1 & y = y' \\ 0 & \text{sonst} \end{cases}. \quad (2.18)$$

Der erwartete Gewinn für $\gamma = \gamma_b$ beträgt:

$$\Gamma(x, f, \gamma_b) = \sum_{y \in \mathcal{Y}} p(y | x) \cdot \begin{cases} 1 & f(x) = y \\ 0 & \text{sonst} \end{cases} \quad (2.19)$$

$$= \sum_{y \in \mathcal{Y}} \begin{cases} p(y | x) & f(x) = y \\ 0 & \text{sonst} \end{cases} \quad (2.20)$$

$$= p(f(x) | x) \quad (2.21)$$

Der Term (2.21) und damit der erwartete Gewinn wird also maximiert für $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} p(y | x)$, was der MAP-Inferenz entspricht.

Eine ebenfalls sinnvolle Gewinnfunktion ist die Hamming-Gewinnfunktion

$$\gamma_h(y, y') = \frac{1}{|Y|} \cdot \sum_{i=1}^{|Y|} \begin{cases} 1 & y_i = y'_i \\ 0 & \text{sonst} \end{cases}. \quad (2.22)$$

Im Gegensatz zur binären Gewinnfunktion, bei der nur eine komplett richtige Vorhersage Gewinn ergibt, ist der Gewinn hier proportional zur Anzahl der richtig vorhergesagten Variablen. Mit dieser Gewinnfunktion ist der erwartete Gewinn

$$\Gamma(x, f, \gamma_h) = \sum_{y \in \mathcal{Y}} p(y | x) \cdot \frac{1}{|Y|} \cdot \sum_{i=1}^{|Y|} \begin{cases} 1 & f(x)_i = y_i \\ 0 & \text{sonst} \end{cases} \quad (2.23)$$

$$= \frac{1}{|Y|} \cdot \sum_{y \in \mathcal{Y}} p(y | x) \cdot \sum_{i=1}^{|Y|} \begin{cases} 1 & f(x)_i = y_i \\ 0 & \text{sonst} \end{cases} \quad (2.24)$$

$$= \frac{1}{|Y|} \cdot \sum_{i=1}^{|Y|} \sum_{y \in \mathcal{Y}} p(y | x) \cdot \begin{cases} 1 & f(x)_i = y_i \\ 0 & \text{sonst} \end{cases} \quad (2.25)$$

$$= \frac{1}{|Y|} \cdot \sum_{i=1}^{|Y|} p(Y_i = f(x)_i | x). \quad (2.26)$$

Der erwartete Gewinn wird hier maximiert mit $f(x)_i = \operatorname{argmax}_{y_i \in \mathcal{Y}_i} p(y_i | x)$, also der Maximum-Marginal-Inferenz.

Nach der Inferenzart richtet sich die Wahl des Inferenz-Algorithmus'. Beides auf naive Weise auszurechnen und zu maximieren ist für nichttriviale Problemgrößen nicht machbar, da es die Aufzählung von allen $2^{|Y|}$ möglichen Belegungen von Y erfordern würde. Tatsächlich sind beide Inferenzarten NP-vollständig für allgemeine

Graphen [Shi94]. Durch Beschränkung auf bestimmte Arten von Faktorgraphen und Faktorfunktionen bzw. durch Approximation statt exakter Inferenz lassen sich aber beide Probleme effizient lösen.

MAP-Inferenz mit Graph Cut Um $p(y | x)$ zu maximieren, ist es nicht nötig, die Partitionsfunktion $Z(x)$ zu berechnen. Man kann die Inferenz auf ein Energie-Minimierungs-Problem reduzieren. Die hier verwendeten Faktorfunktionen, als Energiefunktionen formuliert, sind:

$$E_f(x_f, y_f) = \begin{cases} -\log x_f & \text{wenn } y_f = IG \\ -\log(1 - x_f) & \text{wenn } y_f = UG \end{cases} \quad (2.27)$$

für alle $f \in F_1$ und

$$E_f(x_f, y_{f,1}, y_{f,2}) = \begin{cases} 0 & \text{wenn } y_{f,1} = y_{f,2} \\ e & \text{sonst} \end{cases} \quad (2.28)$$

für alle $f \in F_2$. Hier ist $e = -\log c$ mit $e \geq 0$ die Regularisierungskonstante.

Damit ergibt sich

$$y^* = \operatorname{argmax}_{y \in \mathcal{Y}} \frac{1}{Z(x)} \cdot \prod_{f \in F} \exp(-E_f(x_f, y_f)) \quad (2.29)$$

$$= \operatorname{argmax}_{y \in \mathcal{Y}} \prod_{f \in F} \exp(-E_f(x_f, y_f)) \quad (2.30)$$

$$= \operatorname{argmax}_{y \in \mathcal{Y}} \exp \left(- \sum_{f \in F} E_f(x_f, y_f) \right) \quad (2.31)$$

$$= \operatorname{argmax}_{y \in \mathcal{Y}} - \sum_{f \in F} E_f(x_f, y_f) \quad (2.32)$$

$$= \operatorname{argmin}_{y \in \mathcal{Y}} \sum_{f \in F} E_f(x_f, y_f). \quad (2.33)$$

Diese Energieminimierung kann wiederum auf das *Min-Cut-Problem* für Graphen reduziert werden, wenn das CRF bestimmte Bedingungen erfüllt [NL11, S. 242]:

- die Ausgabevariablen Y müssen binär sein, was hier der Fall ist, denn es gibt genau zwei Klassen
- die Energiefunktionen müssen *regulär* sein. Welche Kombinationen von Energiefunktionen genau als regulär gelten, wird in [KZ04] definiert. Im vorliegenden Fall, wo es lediglich unäre (F_1) und binäre (F_2) Faktoren gibt, sind die Energiefunktionen *regulär*, wenn $E_f(x_f, y_f) \geq 0$ für alle $f \in F_1$ gilt und für alle binären Faktoren $f \in F_2$, dass

$$E_f(x_f, y_{f,1}, y_{f,2}) = 0 \quad (2.34)$$

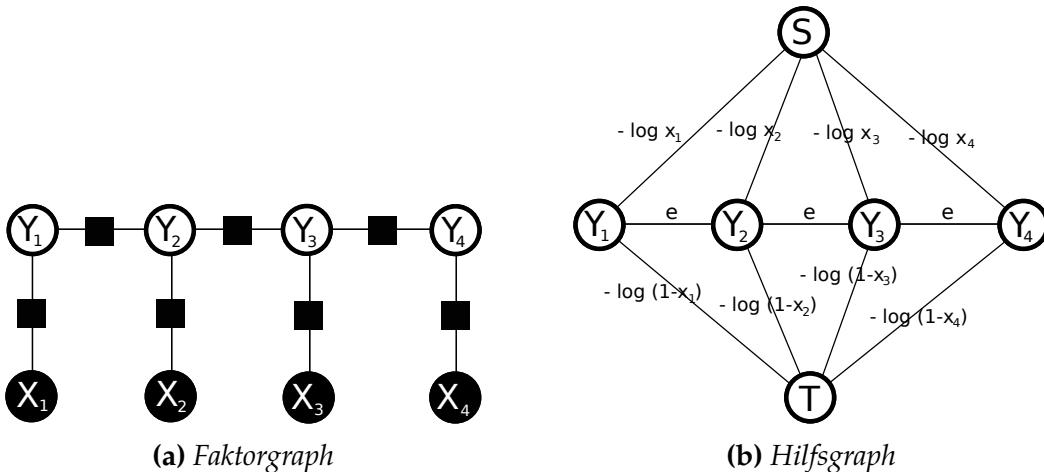


Abbildung 2.1: Ein Beispiel-Faktorgraph mit daraus abgeleitetem Hilfsgraphen. Der Übersichtlichkeit halber in 1D. Im tatsächlich verwendeten Modell bilden die Y ein zweidimensionales Gitter.

wenn $y_{f,1} = y_{f,2}$ und

$$E_f(x_f, y_{f,1}, y_{f,2}) = E_f(x_f, y_{f,2}, y_{f,1}) \geq 0 \quad (2.35)$$

bei nicht übereinstimmenden Belegungen.

Diese Bedingungen sind durch die Wahl der Energiefunktionen (2.27) und (2.28) erfüllt.

Aus dem CRF mit den angegebenen Energiefunktionen kann nun ein *Min-Cut-Problem* generiert werden, indem ein Hilfsgraph $G^H = (V^H, E^H)$ wie folgt konstruiert wird:

- $V^H = Y \cup \{S, T\}$, d. h. Knoten in G^H sind (wie im Faktorgraph) die Variablen Y , sowie die zusätzlichen Knoten S und T .
- Die Kanten E^H sind ungerichtet und gewichtet. Für jeden binären Faktor im Faktorgraph, der zwei Knoten Y_i und Y_j verbindet, ist die Kante $(Y_i, Y_j, E_f(x_f, y_i = IG, y_j = UG))$ in E^H . Das Gewicht der Kanten ist also die Energie für den Fall, dass die beiden betreffenden Variablen verschiedene Belegungen haben.

Außerdem gibt es für jede Variable Y_i eine Kante zum Knoten S , die mit $E_f(x_f, y_i = IG)$ gewichtet ist und eine Kante zu T , die mit $E_f(x_f, y_i = UG)$ gewichtet ist.

Abbildung 2.1 zeigt einen Faktorgraphen und den daraus abgeleiteten Hilfsgraphen.

Das *Min-Cut-Problem* besteht darin, den Graphen G^H durch Eliminieren von Kanten so in zwei Teilgraphen zu zerschneiden, dass S in einem und T im anderen Teil bleibt und die Summe der Gewichte der eliminierten Kanten minimal ist. Aus diesem minimalen Schnitt kann schließlich die Lösung für das ursprüngliche Energieminimierungsproblem abgelesen werden: Die Variablen Y_i , die sich im Teilgraphen mit S

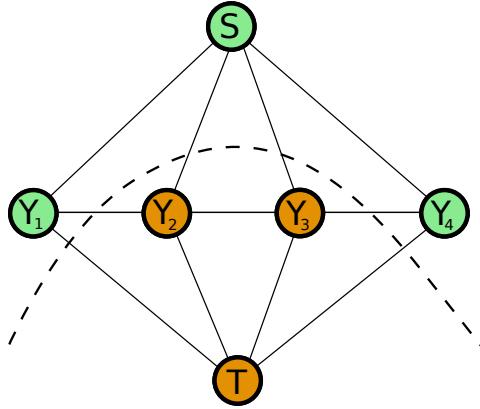


Abbildung 2.2: Hilfsgraph aus Abbildung 2.1 mit Schnitt. Im 2D-Fall kann man sich statt einer Schnittlinie eine Schnittfläche zwischen den Variablen vorstellen.

befinden, haben den Zustand $y_i = IG$, die anderen $y_i = UG$. Abbildung 2.2 zeigt einen Schnitt und die Zuordnung für den Graphen in Abbildung 2.1:

Für das effiziente Lösen eines Min-Cut-Problems existieren zahlreiche Algorithmen. Der in der Implementierung zu dieser Arbeit verwendete Algorithmus stammt aus [BK04]. Er beruht auf einer weiteren Reduktion des Problems, und zwar auf ein äquivalentes *Max-Flow-Problem*. Das ist die Aufgabe, in einem gerichteten, gewichteten Graphen den Fluss von einem Quell- zu einem Zielknoten zu maximieren. Aus einer Lösung dafür kann auch der minimale Schnitt abgeleitet werden [FF56]. Der verwendete Algorithmus hat die Worst-Case-Komplexität $\mathcal{O}(|V^H|^2 \cdot |E^H| \cdot |C|)$ [BK04, S. 9], wobei $|C|$ die Kosten für den minimalen Schnitt sind.

Maximum-Marginal-Inferenz mit Gibbs-Sampling Statt zu versuchen, eine exakte Lösung zu finden, besteht auch die Möglichkeit, die Lösung nur zu approximieren, dafür auf effiziente Weise. Ein Verfahren dafür ist das *Gibbs-Sampling*.

Es ist ein Monte-Carlo-Verfahren, denn dabei werden Stichproben aus der Verteilung $p(y | x)$ generiert, ohne diese Verteilung explizit zu berechnen. Diese Stichproben können für die Maximum-Marginal-Inferenz benutzt werden: Sei y^1, y^2, \dots, y^N eine Menge von Samples, deren Verteilung $p(y | x)$ ist. Sei außerdem $h(y)$ eine beliebige Funktion $h: \mathcal{Y} \mapsto \mathbb{R}$. Dann ist eine Approximation des Erwartungswerts von $h(y)$

$$\sum_y p(y | x) \cdot h(y) \approx \frac{1}{N} \cdot \sum_{j=1}^N h(y^j). \quad (2.36)$$

Für die Wahl

$$h_i(y) = \begin{cases} 1 & y_i = IG \\ 0 & \text{sonst} \end{cases} \quad (2.37)$$

ergibt (2.36) genau die Approximation der marginalen Wahrscheinlichkeit für die

Variable Y_i

$$\sum_y p(y \mid x) \cdot \begin{cases} 1 & y_i = IG \\ 0 & \text{sonst} \end{cases} = p(y_i = IG \mid x) \approx \frac{1}{N} \cdot \sum_{j=1}^N \begin{cases} 1 & y_i^j = IG \\ 0 & \text{sonst} \end{cases}. \quad (2.38)$$

Analog lässt sich für jedes $Y_i \in Y$ auch $p(y_i = UG \mid x)$ berechnen und damit das Maximum dieser beiden marginalen Wahrscheinlichkeiten für die Maximum-Marginal-Inferenz.

Eine Methode, eine Menge von Samples zu erzeugen, ist *Markov Chain Monte Carlo* (MCMC). Dabei werden Samples nicht unabhängig voneinander erzeugt, sondern mit einer Markov-Kette, deren Zustände Belegungen y von Y sind und die Übergangswahrscheinlichkeiten so gewählt sind, dass die Wahrscheinlichkeit einer Belegung y gegen $p(y \mid x)$ geht. Ein einfacher Algorithmus dafür ist das *Gibbs-Sampling*. Er ist in Algorithmus 3 dargestellt.

Algorithmus 3 Gibbs-Sampling

```

function GIBBS-SAMPLING( $x, N$ )
     $y \leftarrow$  zufällige Anfangsbelegung
     $zähler \leftarrow 0$                                  $\triangleright$  Vektor mit den gleichen Dimensionen wie  $y$ . Zählt
                                                 $\triangleright$  für jedes  $Y_i$ , wie oft  $IG$  gesampelt wurde.
    loop  $N$  mal
        for all  $Y_i \in Y$  do
             $y_i \leftarrow$  Sampeln aus  $p(y_i \mid y_{\setminus i}, x)$            $\triangleright$  siehe (2.39)
            if  $y_i = IG$  then
                 $zähler_i \leftarrow zähler_i + 1$ 
            end if
        end for
    end loop

    for all  $Y_i \in Y$  do
        if  $zähler_i \geq N/2$  then
             $y_i^* \leftarrow IG$ 
        else
             $y_i^* \leftarrow UG$ 
        end if
    end for
    return  $y^*$ 
end function

```

In jedem Schritt wird jede Variable individuell gesampelt, und zwar aus der bedingten Wahrscheinlichkeit $p(y_i \mid y_{\setminus i}, x)$ für die vorherigen bekannten Werte von $y_{\setminus i}$. Dabei bezeichne $y_{\setminus i}$ eine Belegung von $Y \setminus Y_i$, also von allen Y außer Y_i .

Die Berechnung von $p(y_i \mid y_{\setminus i}, x)$ ist wesentlich einfacher als die Berechnung von $p(y \mid x)$, weil $p(y_i \mid y_{\setminus i}, x) = p(y_i \mid y_{N_G(i)}, x)$, d. h. es hängt nur von den höchstens vier

Variablen ab, mit denen Y_i im Faktorgraph G benachbart ist. Es gilt also

$$p(y_i | y_{\setminus i}, x) = \frac{\prod_{j \in N_G(i)} f_i(y_i, y_j, x)}{\sum_{y_i \in \{IG, UG\}} \prod_{j \in N_G(i)} f_i(y_i, y_j, x)}. \quad (2.39)$$

Um zu zeigen, dass die Markov-Kette, die mit den Übergangswahrscheinlichkeiten aus (2.39) gebildet wird, tatsächlich Samples aus der Verteilung $p(y | x)$ produziert, muss man zeigen, dass $p(y | x)$ die *stationäre Verteilung* dieser Markov-Kette ist. Der Beweis folgt [RN04, S. 635 f.]. Jede Markov-Kette hat genau eine stationäre Verteilung, wenn sie *ergodisch* ist [Pak69]. Eine Bedingung für Ergodizität ist, dass in der Markov-Kette jeder Zustand von jedem aus erreicht werden kann, was hier erfüllt ist, wenn man, wie in Algorithmus 3 angedeutet, in jedem der N Schritte alle Variablen (in beliebiger Reihenfolge) einmal neu sampelt.

Es bezeichne $q(y \rightarrow y')$ die Übergangswahrscheinlichkeit (2.39) für ein beliebiges i und $\pi(y)$ eine Verteilung.

Die Verteilung $\pi^*(y)$ ist die *stationäre Verteilung*, gdw.

$$\pi^*(y') = \sum_y \pi^*(y) \cdot q(y \rightarrow y'). \quad (2.40)$$

Um zu zeigen, dass $\pi^*(y) = p(y | x)$ für ein gegebenes x gilt, kann man zuerst zeigen, dass q die Bedingung des *detaillierten Ausgleichs* für $p(y | x)$ erfüllt:

$$p(y | x) \cdot q(y \rightarrow y') = p(y' | x) \cdot q(y' \rightarrow y) \quad \forall y, y', \quad (2.41)$$

denn es gilt

$$p(y | x) \cdot q(y \rightarrow y') = p(y | x) \cdot p(y'_i | y_{\setminus i}, x) \quad (2.42)$$

$$= p(y_i, y_{\setminus i} | x) \cdot p(y'_i | y_{\setminus i}, x) \quad (2.43)$$

$$= p(y_i | y_{\setminus i}, x) \cdot p(y_{\setminus i} | x) \cdot p(y'_i | y_{\setminus i}, x) \quad (2.44)$$

$$= p(y_i | y_{\setminus i}, x) \cdot p(y'_i, y_{\setminus i} | x) \quad (2.45)$$

$$= p(y' | x) \cdot q(y' \rightarrow y). \quad (2.46)$$

Wenn man in Gleichung (2.41) über alle y summiert, erhält man:

$$\sum_y p(y | x) \cdot q(y \rightarrow y') = \sum_y p(y' | x) \cdot q(y' \rightarrow y) \quad (2.47)$$

$$= p(y' | x) \cdot \sum_y q(y' \rightarrow y) \quad (2.48)$$

$$= p(y' | x). \quad (2.49)$$

Das zeigt, dass $p(y | x)$ die stationäre Verteilung in einer Markov-Kette mit der Übergangswahrscheinlichkeit q ist.

Die Worst-Case-Komplexität ist $\mathcal{O}(N \cdot |Y|)$.

3. Implementierung

Die in Kapitel 2 vorgestellte Kombination aus Random Forest und Conditional Random Field wurde im Rahmen dieser Arbeit in C++ implementiert, unter Zuhilfenahme der Bibliotheken

- *CImg* [Tsc12] für alle Bildverarbeitungs-Funktionen
- *SimpleJSON*¹ für das Speichern und Laden der Random Forests und
- *maxflow* [BK04] für die Inferenz mittels Graph Cut.

Auf der CD, die dieser Arbeit beiliegt, befinden sich alle Quelltexte sowie Testdaten und -skripte, die für die Evaluation verwendet wurden.

Falls der verwendete Compiler OpenMP unterstützt, wird das Training parallelisiert. OpenMP² ist eine Spezifikation von Compilerdirektiven für paralleles Rechnen mittels Shared Memory. Es wird u. a. von GCC und Microsoft Visual Studio (nur bestimmte Versionen) unterstützt. Wenn es aktiviert ist, wird jeder Entscheidungsbaum in einem eigenen Thread trainiert.

3.1. Benutzung

Die Software wurde *Lakaseg* getauft (kurz für *Landkarten segmentieren*) und kann wahlweise als Kommandozeilenprogramm oder als Bibliothek kompiliert werden. Für letzteren Fall steht ein Python-Modul `lakaseg.py` zur Verfügung, mit dem man einfach die Funktionen der Bibliothek aufrufen kann.

Abbildung 3.1 zeigt exemplarisch die Benutzung des Moduls in Python.

Die Bilder mit den Labels müssen genau zwei verschiedene Grauwerte > 0 besitzen. Der hellere davon steht für das „interessante“ Gebiet, der dunklere für den Rest. Pixel mit dem Grauwert 0 werden beim Training ignoriert.

¹<https://github.com/MJPA/SimpleJSON/>

²<http://openmp.org/wp/>

```
import lakaseg

# aus den angegebenen Trainingsbildern (mit Ground truth) einen
# Random Forest lernen und in der Datei "random_forest.json" speichern
lakaseg.trainieren([("karte1.png", "labels1.png"), ("karte2.png", "labels2.png")],
                    "random_forest.json",
                    forest_size = 10,
                    max_tree_depth = 5,
                    testobject_tries = 500,
                    window_size = 5)

# mit dem gelernten Random Forest eine andere Karte segmentieren und das
# Ergebnis in "ergebnis.png" speichern
lakaseg.segmentieren("karte.png", "random_forest.json", "ergebnis.png",
                      edge_weight = 8.5,
                      inference_method = "maxflow")
```

Abbildung 3.1.: Die Benutzung des Programms in Python

4. Evaluation

Um das hier beschriebene Verfahren zu evaluieren und um für die praktische Anwendung sinnvolle Parameter für das Programm herauszufinden, wurden verschiedene Tests mit Karten, die mir vom IÖR zur Verfügung gestellt wurden, durchgeführt.

4.1. Datensatz

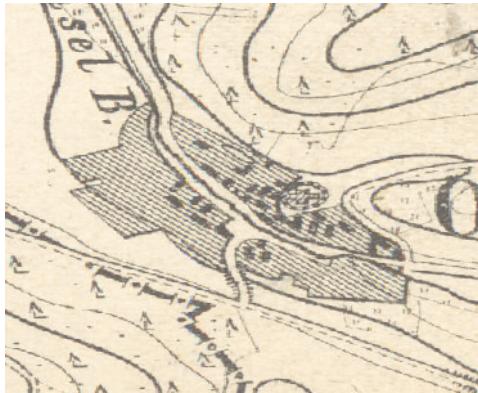
Bei den Testbildern handelt es sich um Ausschnitte von eingescannten Messtischblättern. Die Bilder sind von unterschiedlichem Format und Größe. Die durchschnittlichen Maße sind etwa 2000×1500 Pixel. Allerdings soll das Programm später auch mit deutlich größeren Bildern genutzt werden.

Die Bilder sind Scans von weißem bis gelblichem, teilweise leicht verschmutztem Papier, auf das mit dunkler Tinte gezeichnet und gedruckt wurde. Lediglich Gewässer sind in einigen Karten blau. Aber da diese keine Textur haben und damit ohnehin keine Verwechslungsgefahr mit Siedlungsgebieten besteht, können alle Bilder vor der Bearbeitung problemlos in Grauwertbilder umgewandelt werden.

Siedlungen sind mit engen, parallelen Strichen markiert. Die Striche gehen entweder schräg von links oben nach rechts unten oder von rechts oben nach links unten. Der genaue Winkel ist unterschiedlich, auch innerhalb einer Karte. Außerdem sind innerhalb oder auch neben der Schraffur Gebäude als dunkle Punkte eingezeichnet. Siedlungen sind von hellen Straßen und Wegen durchzogen, die auch zur Siedlung zählen sollen. Insgesamt sehen Siedlungsgebiete einigermaßen einheitlich aus, haben aber sehr unterschiedliche Form und Größe. Abbildung 4.1 zeigt exemplarisch eine Siedlung.

Nicht-Siedlungen sind dagegen sehr uneinheitlich, da es in Messtischblättern viele verschiedene Linien- und Flächenelemente gibt. Sie können viel weißen Raum enthalten, aber auch Wege oder Waldgebiete, die emblematisch mit Bäumen markiert sind. Es gibt auch Gebiete (Dämme oder Sumpfgebiete), die ebenfalls schraffiert sind, aber auf andere Weise (dichter oder weniger dicht, verschiedene Strichdicke, andere Orientierung) als Siedlungsgebiete. Manche Gebiete sind aber selbst für das menschliche Auge schwer zu identifizieren. Hier liegt also Verwechslungsgefahr mit Siedlungsgebieten vor. Außerdem sind über allen geographischen Daten Informationen wie Städtenamen, Grenz- oder Höhenlinien eingezeichnet. Abbildung 4.2 zeigt einen Kartenausschnitt ohne Siedlung. Die Abbildungen A.5 bis A.6 zeigen mögliche Linien- und Flächenelemente von Messtischblättern.

In allen Bildern ist deutlich mehr Nicht-Siedlungsgebiet als Siedlungsgebiet vorhanden.



(a) Kartenausschnitt



(b) Ground truth für diesen Ausschnitt.
Hell bedeutet Siedlung.

Abbildung 4.1.: Beispiel für Siedlungsgebiet

Die Auflösung der Scans ist unterschiedlich, was es eventuell problematisch macht, Siedlungen korrekt zu erkennen, da der Abstand der parallelen Linien, die Siedlungsgebiet darstellen, dadurch unterschiedlich ist.

4.2. Experimente

Insgesamt stehen nur drei gelabelte Testbilder zur Verfügung, die auch etwas unterschiedliches Aussehen haben (leicht verschiedene Auflösung, andere Orientierung der Schraffuren, unterschiedliche Flächenelemente in der Umgebung der Siedlungen). Es wäre sinnvoll, mit allen drei Arten zu trainieren und zu testen. Daher wurden diese drei Bilder für die Experimente jeweils in zwei oder drei Stücken geteilt und in verschiedenen Kombinationen getestet.

4.2.1. Das Maß für die Testleistung

Um die Güte eines Segmentierungsergebnisses zu messen, könnte man es einfach mit der Ground truth vergleichen und die Anzahl der richtig klassifizierten Pixel zählen. Das wäre aber oft nicht sehr aussagekräftig, weil die meisten Testbilder nur relativ wenige Siedlungspixel enthalten. Ein Klassifikator würde also bei manchen Bildern schon gute Ergebnisse erzielen, wenn er einfach jedes Pixel als Nichtsiedlung klassifiziert.

Besser geeignet sind *Precision* und *Recall*. Diese Maße wurden ursprünglich für das *Information Retrieval* entwickelt, sind aber auch hier anwendbar: Weil das Ziel ist, Siedlungspixel im Bild zu finden, kann man falsch-positiv klassifizierte (*FP*) Pixel zählen (d. h. fälschlicherweise als Siedlung klassifiziert) und analog falsch-negative (*FN*) sowie richtig-positive (*TP*) und richtig-negative (*TN*).

Die *Precision p* gibt an, wieviele derjenigen Pixel, die als Siedlung klassifiziert wurden,

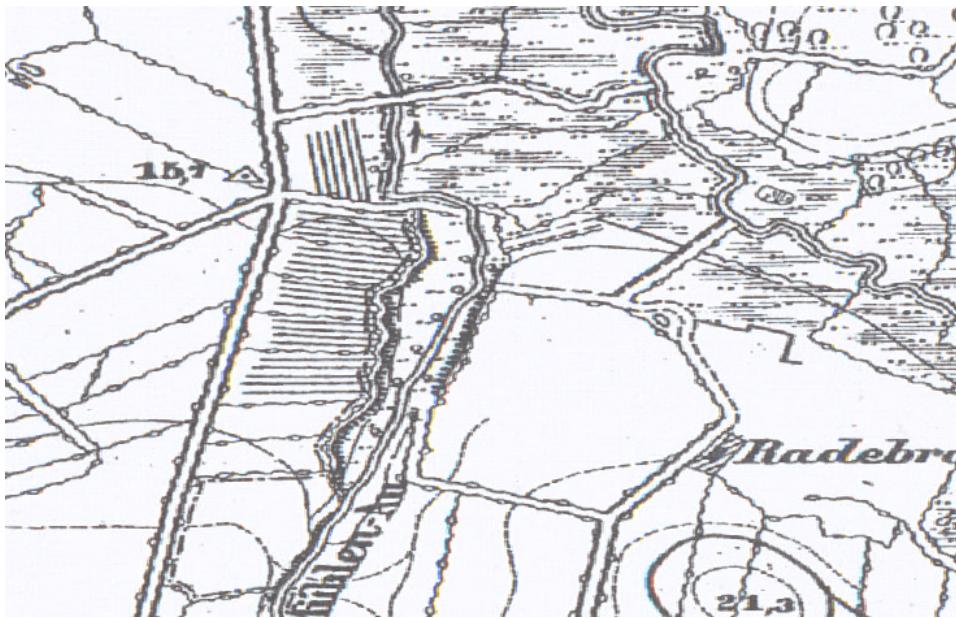


Abbildung 4.2.: Beispiel für Nicht-Siedlungsgebiet

tatsächlich Siedlung sind:

$$p = \frac{TP}{TP + FP}. \quad (4.1)$$

Der *Recall* r gibt an, wieviele der tatsächlichen Siedlungspixel gefunden wurden, also auch als Siedlung klassifiziert wurden:

$$r = \frac{TP}{TP + FN}. \quad (4.2)$$

Das Ziel für einen Klassifikator ist, sowohl p als auch r zu maximieren.

Um die Leistung von Parameterkonfigurationen miteinander vergleichen zu können, werden die Werte p und r zu einem Wert zusammengefasst, und zwar zum *F-Maß*. Das ist der gewichtete harmonische Mittelwert von p und r :

$$F = \frac{1}{\alpha \cdot p^{-1} + (1 - \alpha) \cdot r^{-1}}. \quad (4.3)$$

Mit dem Parameter α kann man p oder r stärker gewichten. Da es hier keinen Grund gibt, eins der beiden hervorzuheben, ist hier $\alpha = 0,5$.

4.2.2. Parameter für das Training lernen

Es gibt vier Parameter, mit denen man das Training des Random Forests steuern kann:

1. die Anzahl t der Entscheidungsbäume im Random Forest
2. die maximale Tiefe d der Entscheidungsbäume

Parameter	Wert
Anzahl der Bäume t	7
Maximale Tiefe d	3
Anzahl der Samples p	600
Fensterbreite w	9

Tabelle 4.1: Optimale Trainingsparameter für die hier verwendeten Trainingsdaten

3. die Anzahl der Samples p beim Generieren eines Testknotens in den Entscheidungsbäumen
4. die Breite w des Fensters um ein Pixel, dessen Inhalt in einem Testknoten verwendet wird.

Gute Werte für diese Parameter sollen per Kreuzvalidierung herausgefunden werden: Mit drei Bildern, und zwar je einem Teilstück der drei Karten, wurde ein Random Forest trainiert, mit dem die übrigen Teilstücke getestet wurden. Das ganze wurde für alle zwölf Kombinationen von Trainings-Teilstücken gemacht und die Testleistung gemittelt.

Da bei jedem dieser Parameter ein höherer Wert die Anzahl der Freiheitsgrade in einem Random Forest erhöht, ist zu erwarten, dass die Leistung auf den Testdaten mit dem Parameterwert erst steigt und bei höheren Werten wieder fällt, weil bei kleinen Werten Unteranpassung und bei großen Werten Überanpassung stattfindet. Es dürfte allerdings nicht ausreichen, den optimalen Wert für jeden Parameter unabhängig von den anderen Parametern zu suchen, weil die beste Wahl eines Parameters auch die beste der anderen Parameter beeinflussen dürfen, z. B. sollte man bei einem großen Fenster w auch für p einen hohen Wert wählen, weil mehr Freiheitsgrade für die Testfunktion bedeuten, dass auch mehr Samples ausprobiert werden müssen. Deshalb wurde beim Bestimmen der besten Parameter so vorgegangen, dass für drei festgelegte Parameter der vierte variiert wurde und derjenige mit der höchsten Testleistung bestimmt wurde. Dieser Wert wurde festgesetzt und auf die gleiche Weise mit den anderen drei Parametern verfahren. Das ganze hat sich so lange wiederholt, bis ein Fixpunkt mit den besten Parametern gefunden wurde.

Für die Inferenz wurde dabei der Maxflow-Algorithmus verwendet. Die Regularisierungskonstante e (siehe (2.28)), die einstellt, wie sehr das Ergebnis durch das CRF geglättet wird, betrug 5,0.

Tabelle 4.1 zeigt das Ergebnis. Die Abbildungen 4.3 bis 4.6 zeigen die Testleistung für verschiedene Werte der Parameter, wobei die restlichen Parameter den Wert aus Tabelle 4.1 haben.

Etwas unerwartet ist, dass schon drei Entscheidungsknoten ausreichen, um die Pixel in den Testdaten ausreichend gut zu klassifizieren. Bei tieferen Entscheidungsbäumen tritt bereits Überanpassung auf. Man sieht, dass die anderen Parameter (innerhalb vernünftiger Grenzen) keine besonders großen Auswirkungen haben.

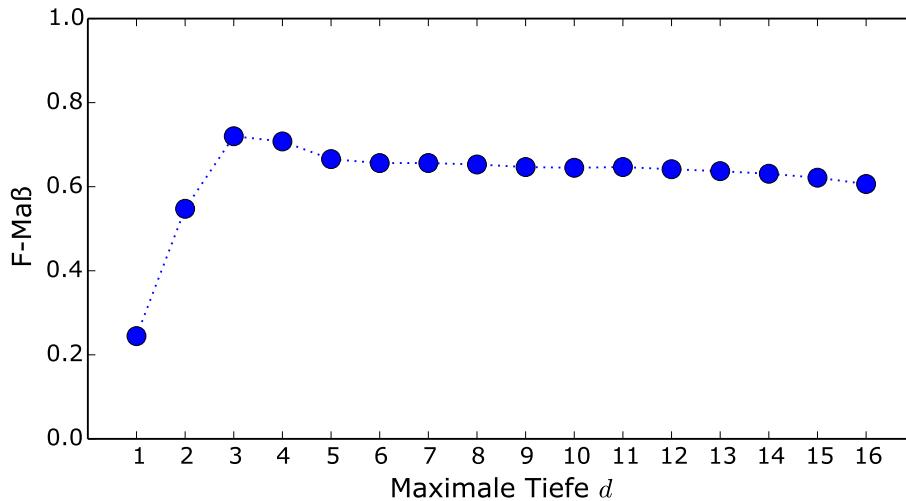


Abbildung 4.3.: Testleistung abhängig von der maximalen Tiefe

4.2.3. Generalisierung des Trainings

Dann wurde getestet, wie gut ein trainierter Random Forest auf andere Bilder generalisiert. Dazu wurden Random Forests auf m zufällig ausgewählten Pixeln (je zur Hälfte Siedlungspixel und Nicht-Siedlungspixel) aus einem der Testbild-Teile trainiert und die Testleistung auf dem anderen Teil desselben Bildes sowie auf Bildern anderer Typs gemessen. Das ganze wurde für jedes m 50 Mal durchgeführt und die Leistung gemittelt.

Trainiert wurde mit dem Bild in Abbildung A.1. Es wurde ausgewählt, weil es eine sehr typische Karte ist¹.

Die verwendeten Parameter für das Training sind die in Tabelle 4.1 angegebenen. Für die Inferenz wurde wieder der Maxflow-Algorithmus mit $e = 5,0$ verwendet.

Abbildung 4.7 zeigt das Ergebnis. Man erkennt, dass die Leistung wesentlich besser ist, wenn der Random Forest auf dem gleichen Kartentyp getestet wird, auf dem er trainiert wurde. Außerdem wird schon ab etwa 1000 Trainingspixeln die Leistung nicht mehr wesentlich besser. Das legt nahe, dass für die praktische Anwendung die Qualität der Trainingsdaten wichtiger ist als die Quantität.

4.2.4. Vergleich der Inferenzarten

Weiterhin wurde die beiden Inferenzmethoden verglichen und wie sich die Regularisierungskonstante e bei der Inferenz auswirkt. Wie in Abschnitt 4.2.2 wurden alle zwölf Kombinationen von Trainings- und Testbildern verwendet, um die Testleistung zu ermitteln. Diesmal wurde mit den festen Parametern aus Tabelle 4.1 trainiert und der Inferenzalgorithmus und der Wert e variiert. Beim Gibbs-Algorithmus wurde

¹Mündliche Mitteilung von IÖR-Mitarbeitern

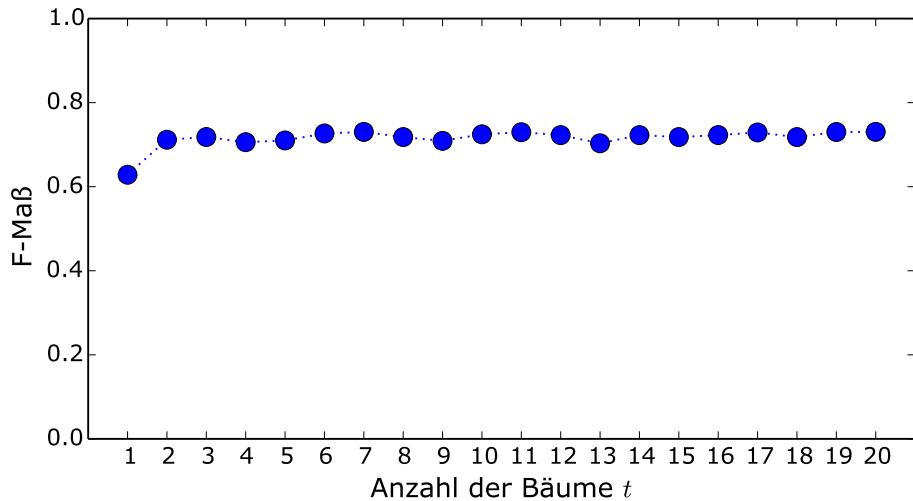


Abbildung 4.4.: Testleistung abhängig von der Anzahl der Entscheidungsbäume

außerdem die Anzahl der Schritte N variiert. Abbildung 4.8 zeigt das Ergebnis.

Der Maxflow-Algorithmus zeigt bei $e = 5$ die beste Leistung. Die Wahl von e hat einen sehr großen Einfluss auf die Güte der Segmentierung. Bei $e = 50$ ist das Ergebnis so geglättet, dass allen Pixeln Nichtsiedlung zugeordnet wurde.

Beim Gibbs-Algorithmus dagegen ändert sich die Leistung bei $e \geq 4$ kaum, dafür hängt sie sehr von N ab. Das weist darauf hin, dass der Sampling-Algorithmus selbst nach einigen tausend Schritten noch nicht konvergiert ist.

Abbildung 4.9 zeigt die Wirkung des Parameters e anhand des Bildes aus Abbildung A.1. Es wurde die Maxflow-Methode verwendet. Abbildung 4.10 zeigt für den Gibbs-Algorithmus die Wirkung von verschiedenen Werten für N mit $e = 5$.

4.2.5. Generalisierung der Inferenz

Da, zumindest beim Maxflow-Algorithmus, die Testleistung stark mit der Wahl von e schwankt, wurde nun noch getestet, ob und wie stark der optimale Wert für e bei verschiedenen Kartentypen variiert. Deshalb wurde für jeden der drei Kartentypen alle Kombinationen von Trainings- und Inferenzbild für verschiedene Werte von e getestet.

Das Ergebnis ist in Abbildung 4.11 zu sehen. Da hier jeweils mit dem gleichen Typ trainiert und getestet wurde, ist, zumindest bei den Kartentypen 1 und 3, die Leistung besser als in allen bisherigen Tests. Es zeigt sich, dass günstige Werte für e nicht unbedingt von einem Kartentyp auf einen anderen übertragbar sind. Die Schlussfolgerung ist also wieder, dass man in der Praxis jeden Kartentyp gesondert bearbeiten sollte und dass man verschiedene Werte für e ausprobieren sollte, denn die Testleistung kann sehr stark variieren.

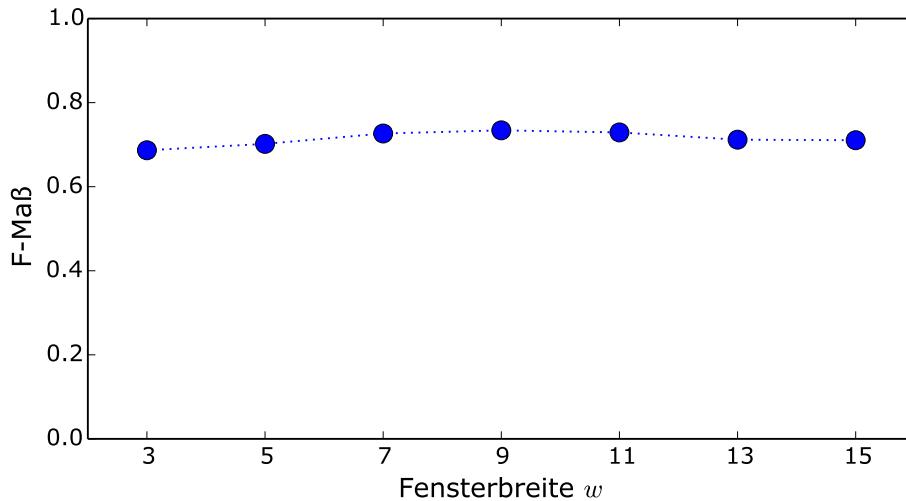


Abbildung 4.5.: Testleistung abhängig von der Fensterbreite

4.3. Geschwindigkeit

Die Zeit, die das Programm für Training und Inferenz braucht, ist in Abbildung 4.12 bis 4.16 dargestellt. Dargestellt ist die durchschnittliche Dauer über die zwölf Kombinationen von Trainings- und Inferenzbildern, wie in Abschnitt 4.2.2 beschrieben. Jeweils ein Parameter wurde variiert und die restlichen mit den Werten aus Tabelle 4.1 belegt. Die Inferenzmethode war auch hier *Maxflow* mit $e = 5,0$, wenn nicht anders angegeben.

Gemessen wurde die Zeit auf einem Intel Core i7-3770 ($8 \times 3,4$ GHz) mit 20 GiB RAM. Das Betriebssystem war Linux (Ubuntu 14.04).

Mit den Parametern in Tabelle 4.1 dauerte das Training auf den hier verwendeten Bildern im Durchschnitt ca. 11 Sekunden pro Random Forest und die Inferenz ca. 1 Sekunde pro Bild.

In Abbildung 4.13 ist zu sehen, dass die Trainingszeit bei $t = 9$ und $t = 17$ einen kleinen Sprung macht, was daran liegt, dass der Testcomputer acht Prozessorkerne hat und das Lernen von bis zu acht Entscheidungsbäumen daher parallel erfolgen kann. Allerdings ist der Speedup offensichtlich fast vernachlässigbar gering. Beim Versuch, stattdessen das Sampeln der Testknoten zu parallelisieren, war der Speedup noch kleiner. Hier besteht also Optimierungspotenzial.

Abbildung 4.15 zeigt, dass bei kleinem p die Inferenz durchschnittlich deutlich länger dauert als sonst. Der Grund ist, dass bei diesen Werten die Entscheidungsbäume schlechte Unary Potentials erzeugt, die sehr uneinheitlich über das Bild verteilt sind. Dabei braucht der Maxflow-Algorithmus länger, um die Lösung zu finden.

Abbildung 4.16 zeigt, dass die Inferenz per Gibbs-Sampling bei $N = 10.000$ im Durchschnitt fast 4 Minuten pro Bild braucht. Diese Methode ist also wesentlich langsamer als Maxflow, trotz der geringeren theoretischen Worst-Case-Komplexität.

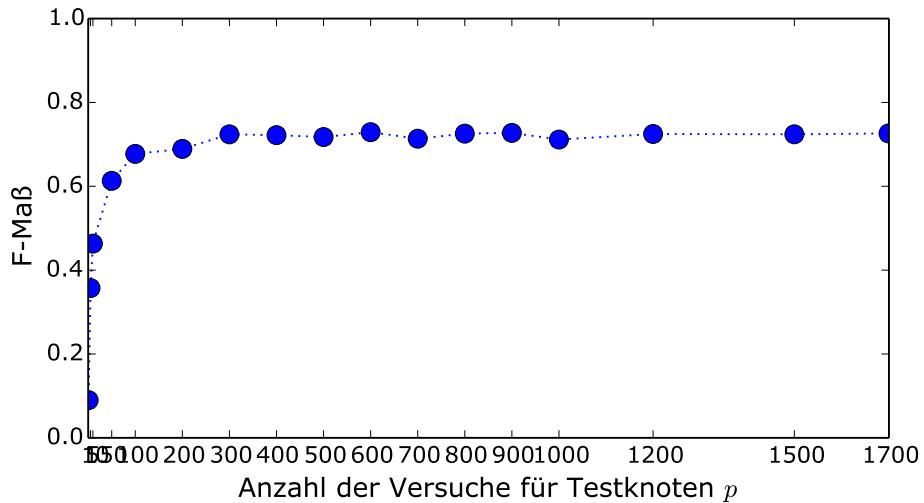


Abbildung 4.6.: Testleistung abhängig von der Anzahl der Versuche beim Sampeln der Testfunktionen

4.4. Diskussion

Die Abbildungen A.1 und A.2 zeigen zur Illustration die Ausgabe des Programms für zwei Kartenstücke. Die Ground truth dafür ist in Abbildung A.3 und Abbildung A.4 zu sehen. Gelernt wurde jeweils auf allen zur Verfügung stehenden Karten außer auf dem abgefragten Kartenstück. Die Trainingsparameter waren die in Tabelle 4.1 und für die Inferenz wurde Maxflow mit $e = 5,0$ verwendet. Bei Abbildung A.1 handelt es sich um eine eher einfache, aber typische Karte. Im Großen und Ganzen stimmt das Ergebnis mit dem tatsächlichen Siedlungsgebiet überein, bis auf Unsauberkeiten im Süden der Siedlung. In Abbildung A.2 dagegen ist das Ergebnis weniger gut: es werden Feuchtwiesen und Dämme fälschlicherweise als Siedlung erkannt, kleine Siedlungen werden andererseits übersehen. Hier wäre also in der Praxis noch Handarbeit nötig.

Aus den Tests lassen sich einige Empfehlungen für die praktische Anwendung ableiten:

- Die Parameter für das Lernen der Random Forests sind recht robust und können aus Tabelle 4.1 übernommen werden.
- Der Wert von e hat dagegen einen großen Einfluss auf das Ergebnis. Hier lohnt es sich, verschiedene Werte auszuprobieren.
- Es empfiehlt sich, für jeden Kartentyp gesondert einen Random Forest zu lernen. Dazu scheinen schon wenige Trainingsdaten auszureichen.
- Die Zeit für die Inferenz mit dem Maxflow-Algorithmus hängt natürlich von der Bildgröße ab, ist aber wohl ausreichend für die praktische Anwendung. Die Inferenz mit Gibbs-Sampling ist dagegen zu langsam.

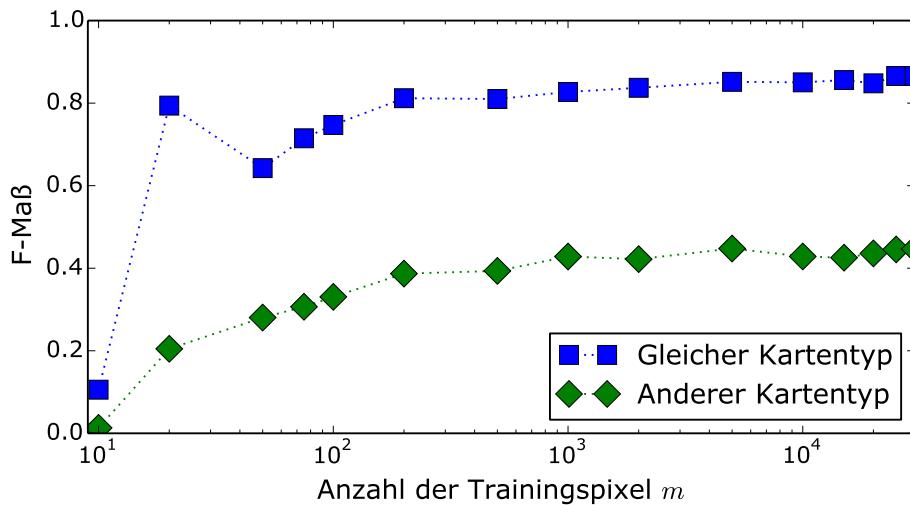


Abbildung 4.7.: Testleistung abhängig von der Anzahl der Trainingspixel

Ob die Qualität der Segmentierung ausreicht, hängt vermutlich von der konkreten Aufgabe ab, die mit dem Programm gelöst werden soll. Eine Eigenschaft des Glättens mit einem CRF und einem konstanten Regularisierungsterm ist, dass mit steigendem e die Gebiete runder und glatter werden. Deswegen hat das Verfahren Schwierigkeiten mit länglichen oder ausgefransten Gebieten, wie man in Abbildung 4.9 sehen kann.

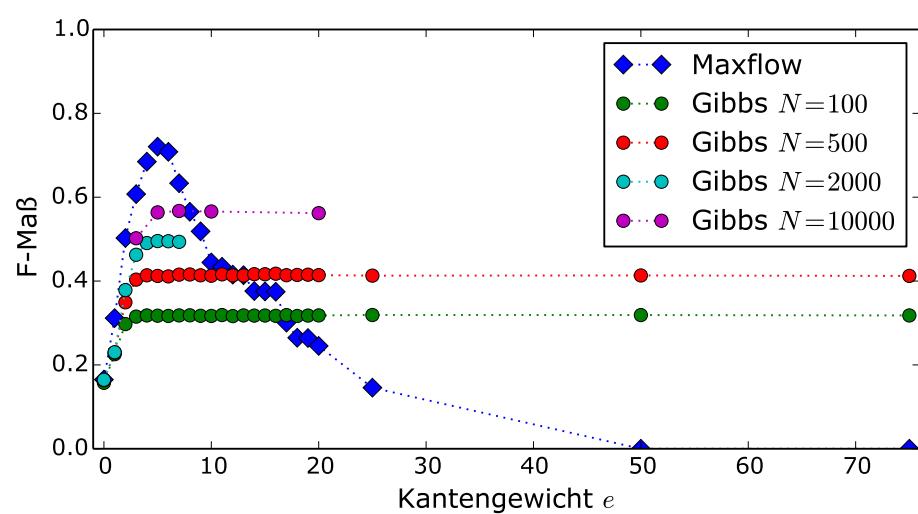


Abbildung 4.8.: Testleistung für verschiedene Werte von e

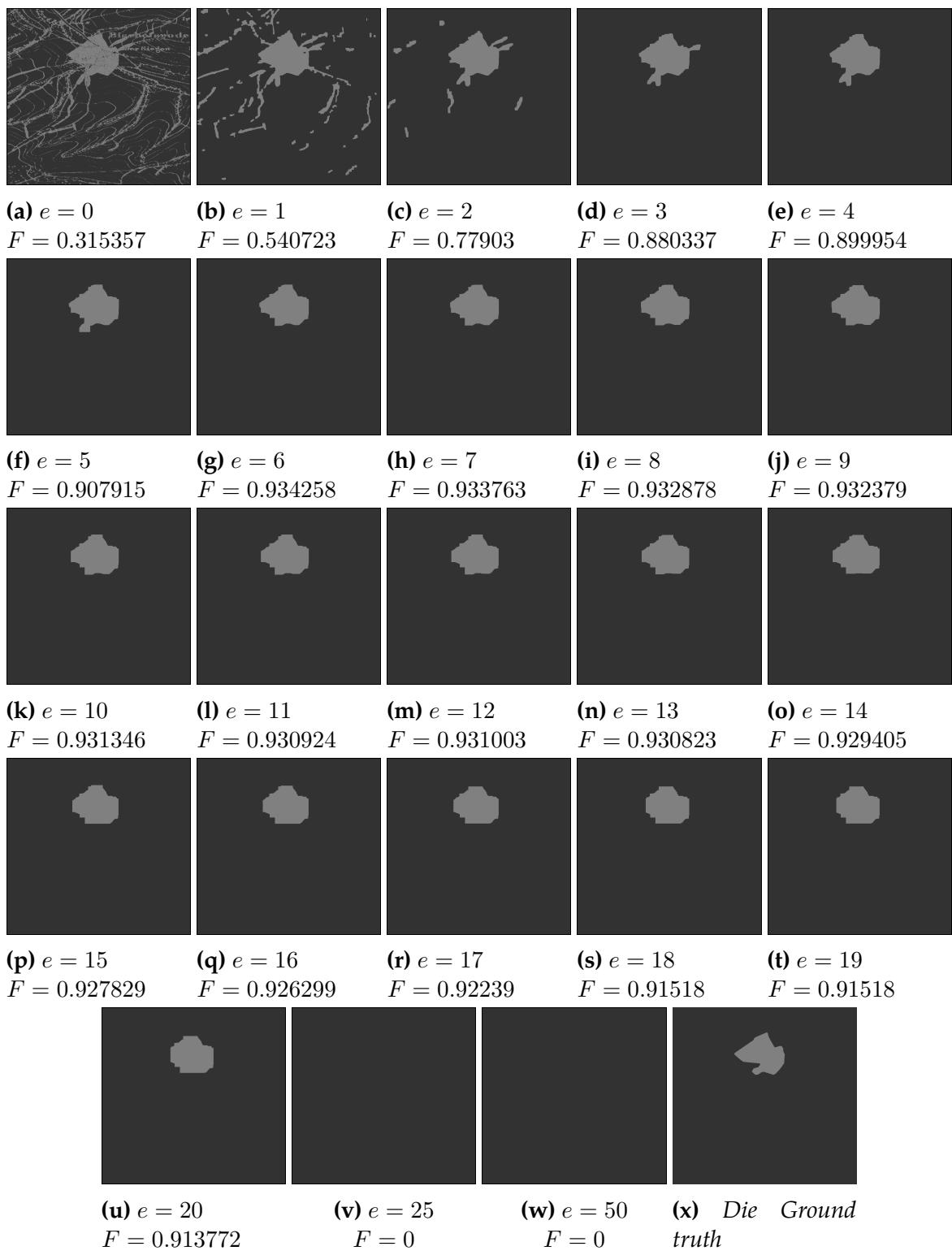


Abbildung 4.9.: Beispiel für die Wirkung der Regularisierungskonstante e bei der Inferenz mit Maxflow

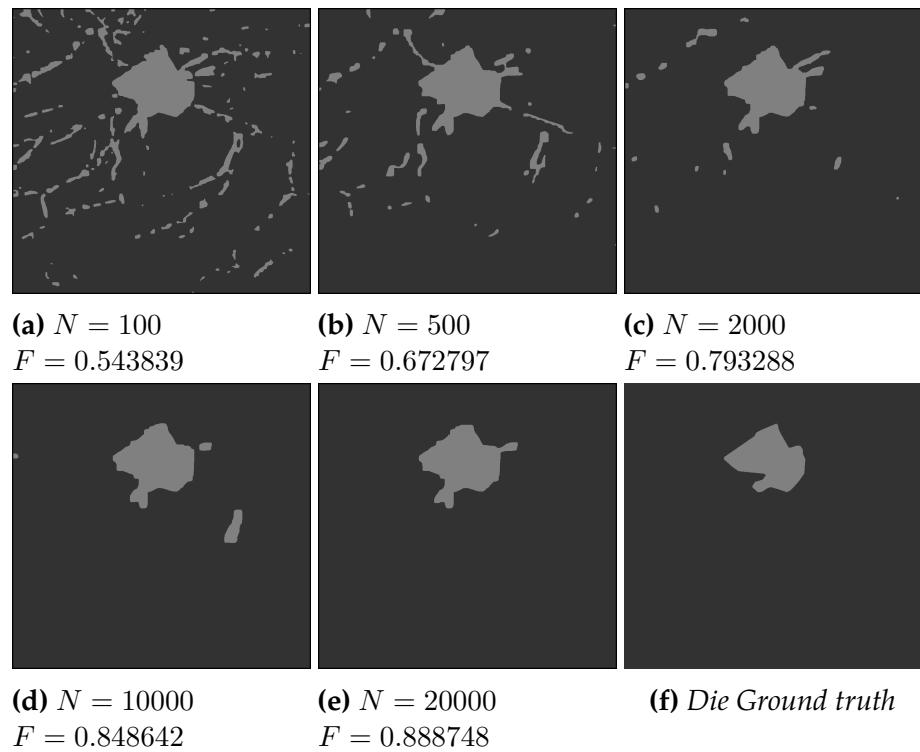


Abbildung 4.10.: Beispiel für die Wirkung der Anzahl der Samplingschritte beim Gibbs-Sampling

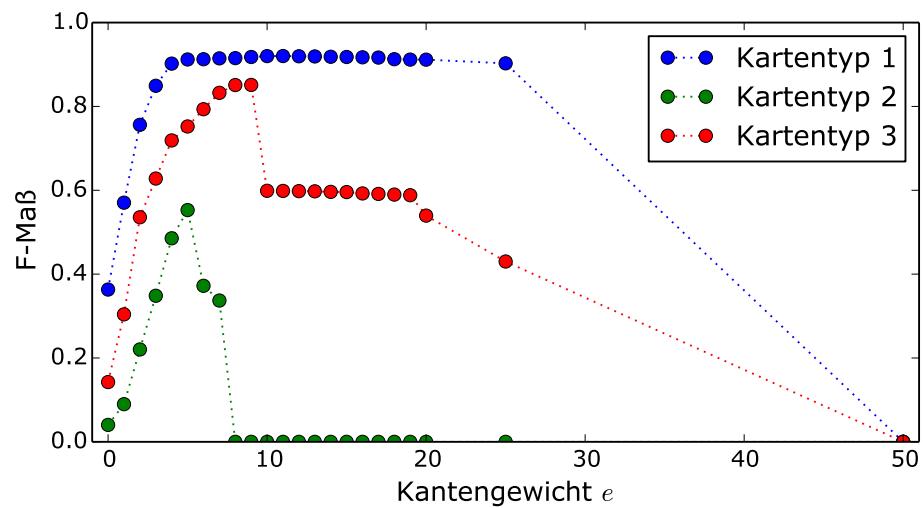


Abbildung 4.11.: Testleistung für verschiedene Werte von e

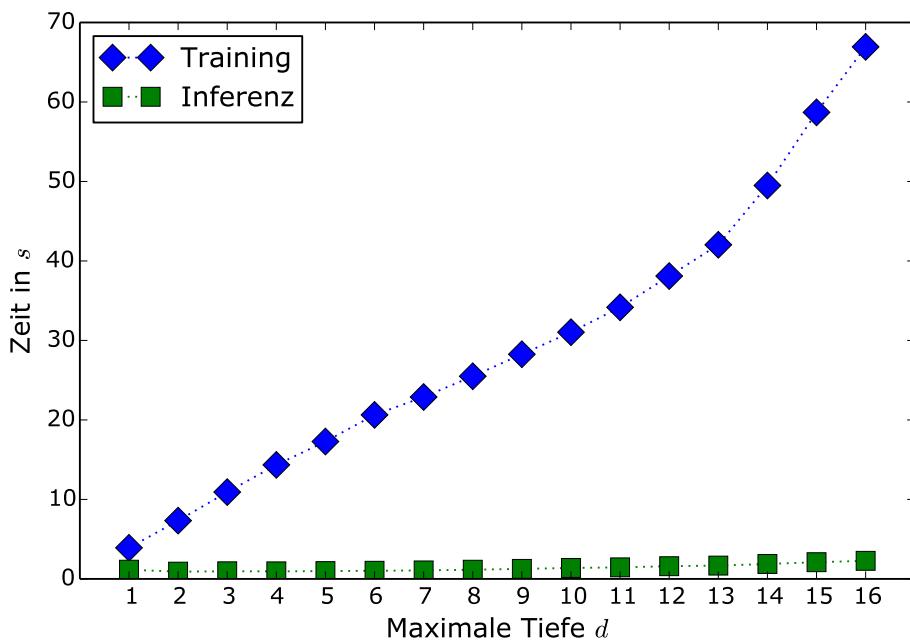


Abbildung 4.12.: Durchschnittliche Trainings- und Inferenzdauer abhängig von der maximalen Tiefe

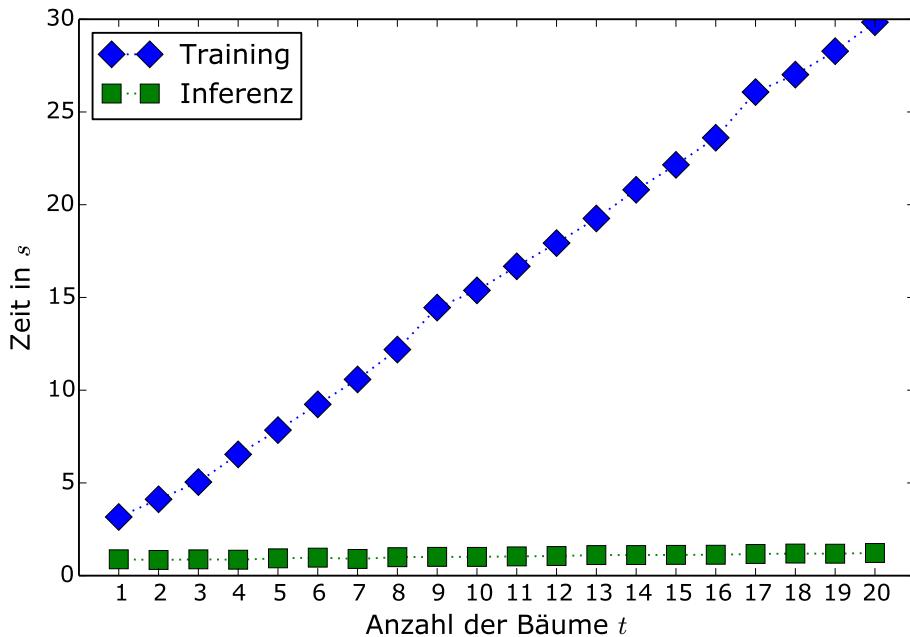


Abbildung 4.13.: Durchschnittliche Trainings- und Inferenzdauer abhängig von der Anzahl der Entscheidungsbäume

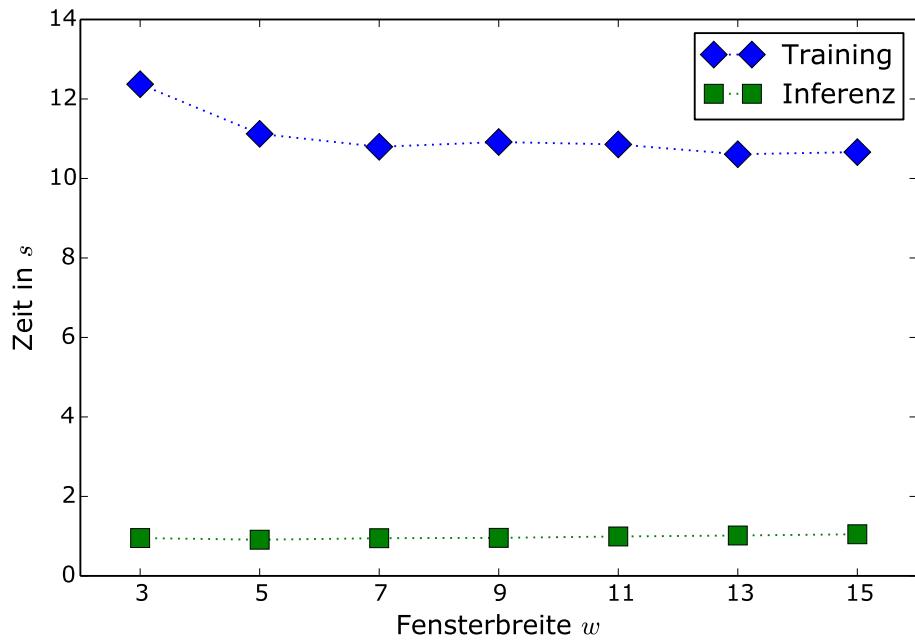


Abbildung 4.14.: Durchschnittliche Trainings- und Inferenzdauer abhängig von der Fensterbreite

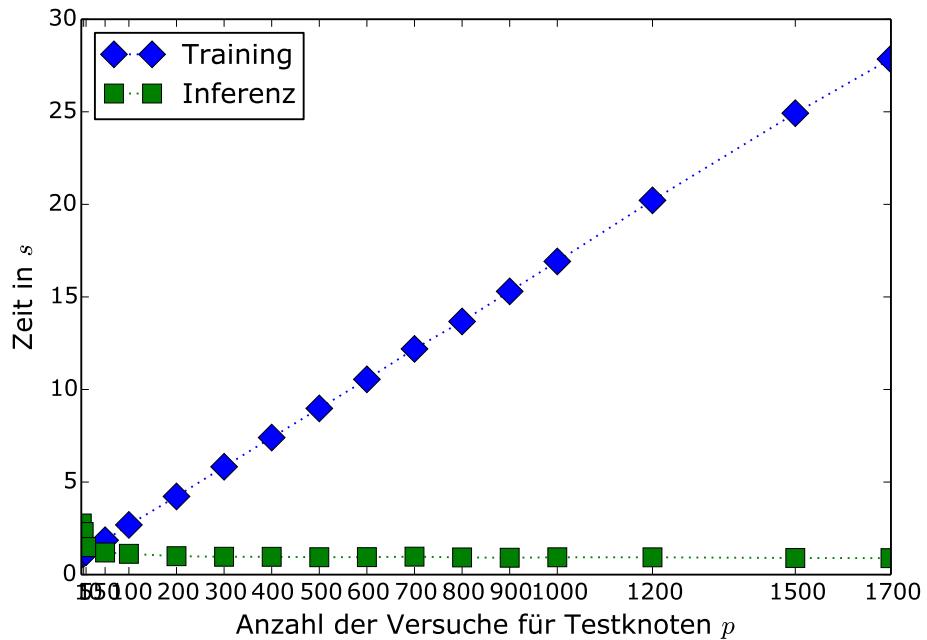


Abbildung 4.15.: Durchschnittliche Trainings- und Inferenzdauer abhängig von der Anzahl der Versuche beim Sampeln der Testfunktionen

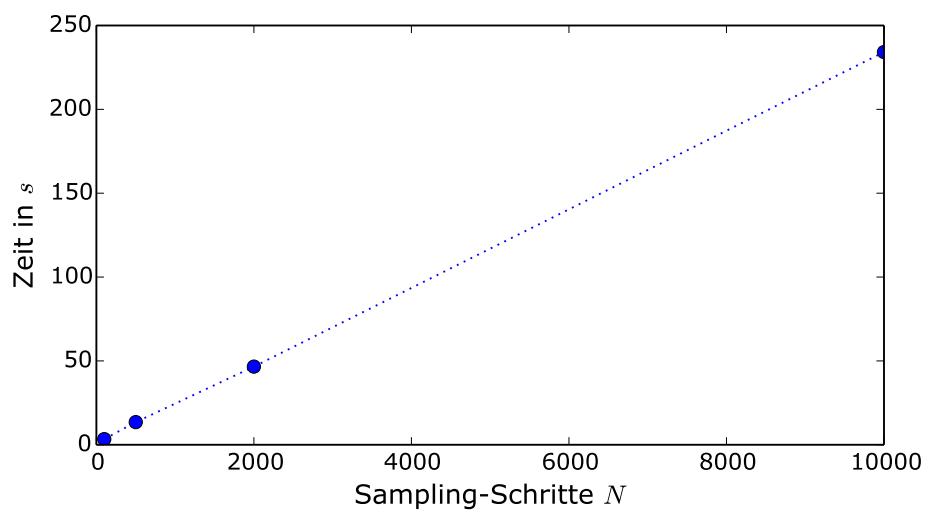


Abbildung 4.16.: Inferenzdauer abhängig von der Anzahl der Schritte beim Gibbs-Sampling

5. Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Methode zur semantischen Segmentierung von historischen Landkarten, basierend auf einer Kombination von Random Forests und Conditional Random Fields, vorgestellt. Die Methode wurde unter Verwendung zweier Inferenzalgorithmen implementiert und getestet. Dabei wurde festgestellt, dass es mit Einschränkungen geeignet ist, Messtischblätter automatisch in Siedlungs- und Nichtsiedlungsgebiete einzuteilen.

Probleme und Grenzen des Systems sind:

- Bei komplizierteren Kartenbildern, die beispielsweise viel Sumpfgebiet enthalten, ist immer noch Handarbeit erforderlich, um z. B. falsch-positive Gebiete zu beseitigen.
- Auch bei einfachen Karten hat die hier vorgestellte Methode die unerwünschte Tendenz, gefundene Siedlungsgebiete so zu glätten, dass Ausbuchtungen verschwinden. Eine Lösung könnte hier sein, im Regularisierungsterm keine Konstanten zu verwenden wie in (2.14), sondern die Funktion vom Bildinhalt abhängig zu machen. Dazu könnte man z. B. einen weiteren Random Forest lernen, der jeweils für zwei benachbarte Pixel die Wahrscheinlichkeit $p(y_1 = y_2 | x)$ dafür ausgibt, dass diese Pixel die gleiche Klasse haben. Dann könnte (2.14) so aussehen:

$$f(x_f, y_{f,1}, y_{f,2}) = \begin{cases} p(y_{f,1} = y_{f,2} | x_f) & \text{wenn } y_{f,1} = y_{f,2} \\ 1 - p(y_{f,1} = y_{f,2} | x_f) & \text{sonst} \end{cases}. \quad (5.1)$$

- Während die Maxflow-Methode eine akzeptable Geschwindigkeit hat, ist die Inferenz per Gibbs-Sampling so langsam, dass sie für die Praxis wohl nicht in Frage kommt. Hier besteht die Möglichkeit, bei Bedarf andere Methoden für Maximum-Marginal-Inferenz zu verwenden, z. B. *Loopy Belief Propagation*.
- Das Parallelisieren des Trainings mit OpenMP hat nur sehr geringe Auswirkungen. Das zu verbessern könnte die Trainingsdauer beträchtlich verkürzen.

A. Anhang

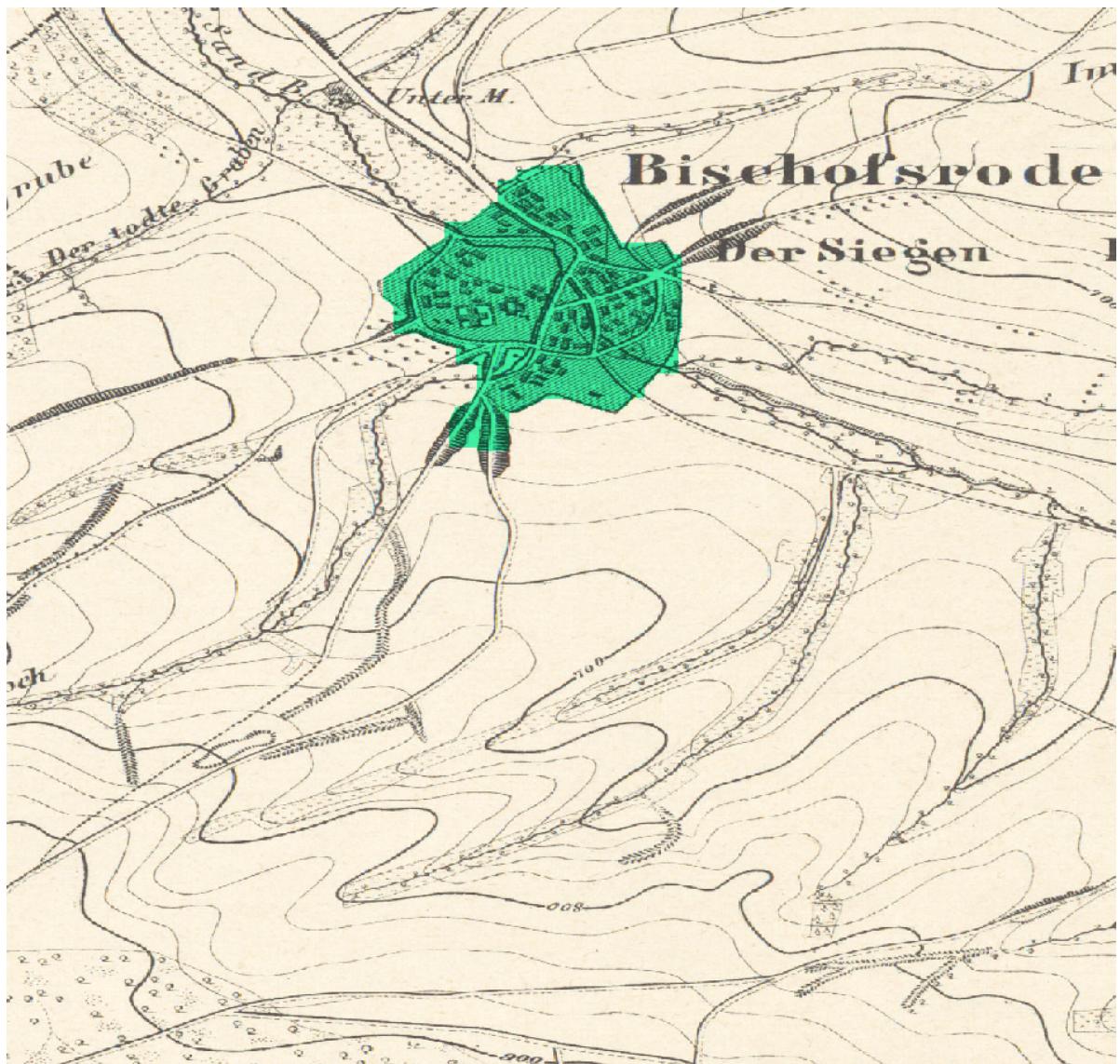


Abbildung A.1.: Beispiel-Inferenz mit einer eher einfachen Karte. Abgebildet ist die Originalkarte und türkis darübergelegt das Ergebnis der Inferenz.

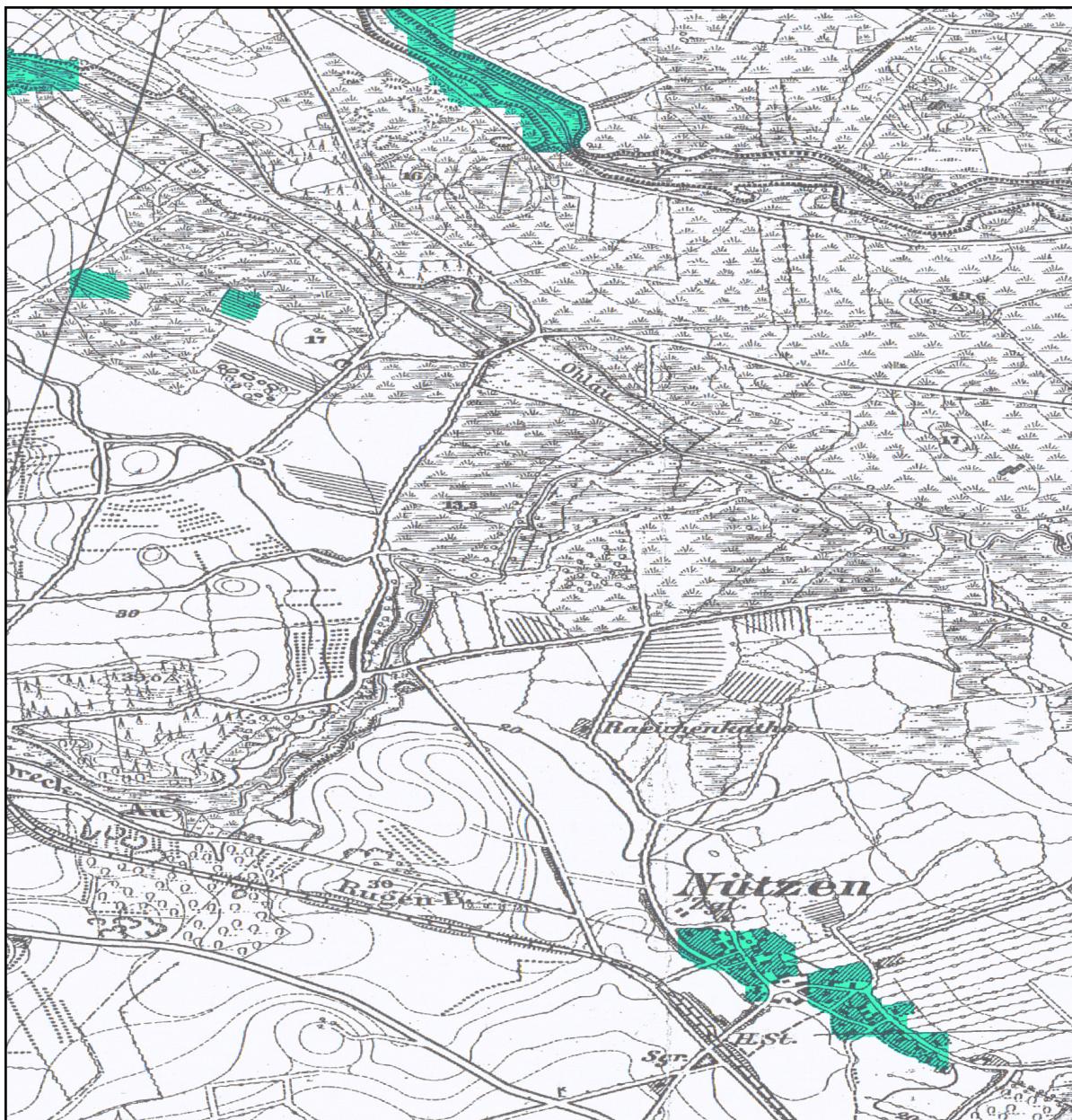


Abbildung A.2: Beispiel-Inferenz mit einer eher schwierigen Karte. Abgebildet ist die Originalkarte und türkis darübergelegt das Ergebnis der Inferenz.

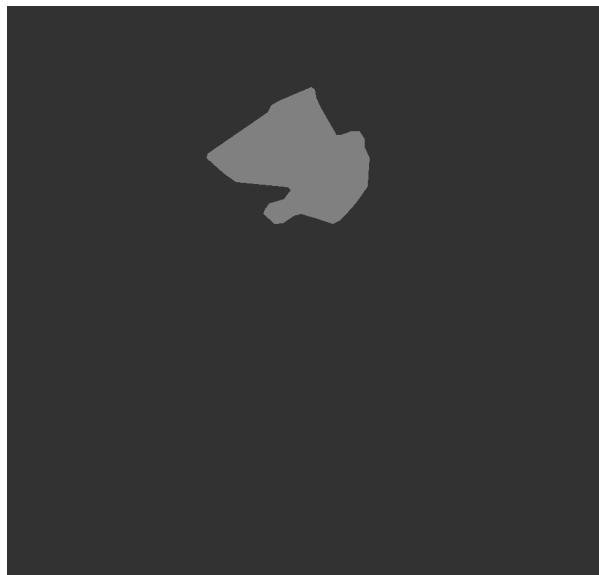


Abbildung A.3.: Die *Ground truth* für Abbildung A.1

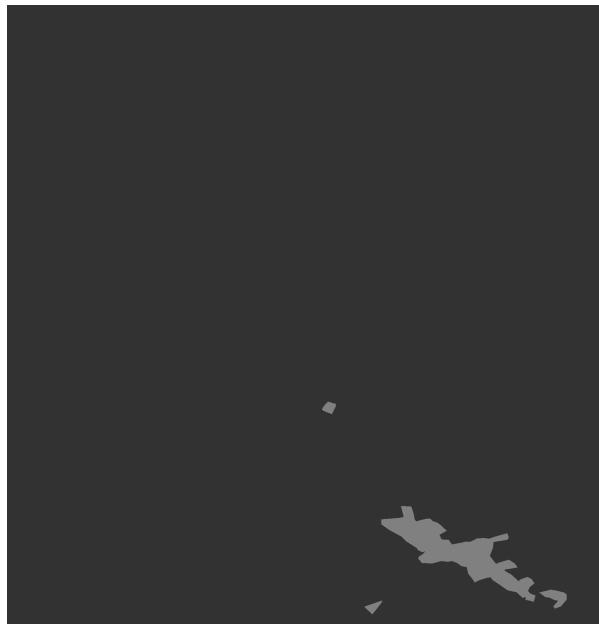


Abbildung A.4.: Die *Ground truth* für Abbildung A.2

Zeichenerklärung:

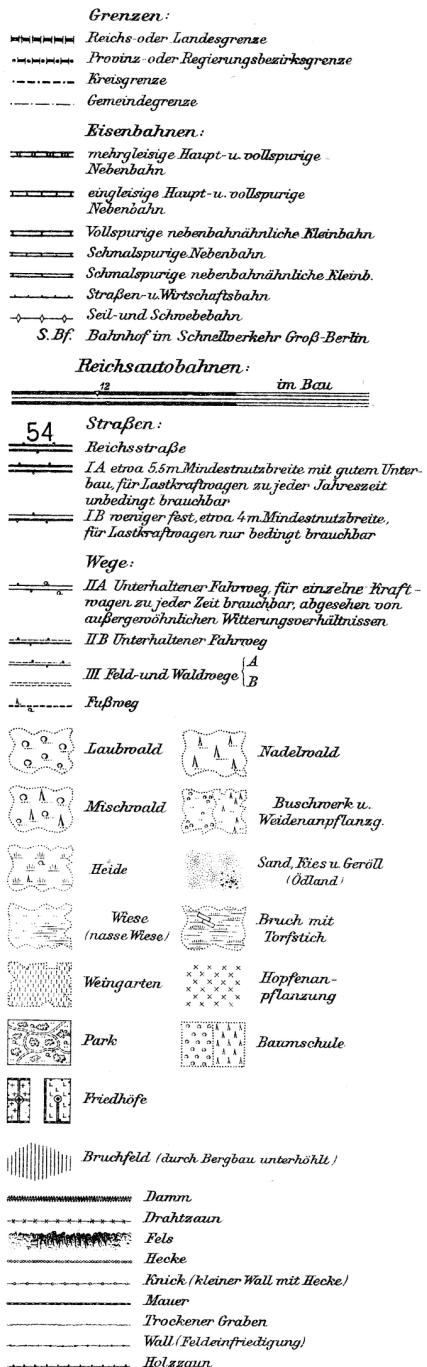


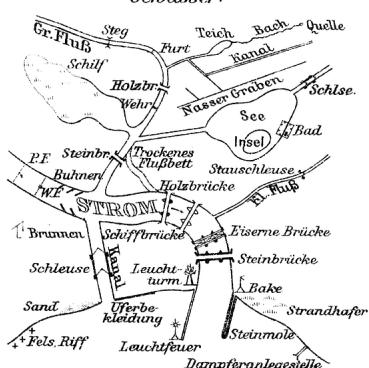
Abbildung A.5.: Legende eines Messtischblatts von 1933

	<i>Denkmal</i>
	<i>Einzelgrab</i>
	<i>Erratischer Block</i>
	<i>Försterei, Waldwärter</i>
	<i>Funkstelle</i>
	<i>Luftfahrtfeuer</i>
	<i>Grenzstein, säule</i>
	<i>Grube, Steinbruch</i>
	<i>Heiligenbild, Kapelle</i>
	<i>Hervorragender Baum</i>
	<i>Höhenpunkt</i>
	<i>Hünenstein, Hünengrab</i>
	<i>Kilometerstein</i>
	<i>Kirche</i>
	<i>Meilenstein</i>
	<i>Nio. Punkte</i>
	<i>Pegel</i>
	<i>Ruine</i>
	<i>Bergwerk, im Betrieb u. verlassen</i>
	<i>Schlacht, Gefechtsfeld</i>
	<i>Schornstein, frei u. im Haus</i>
	<i>Treibbake</i>
	<i>Trig. Punkt</i>
	<i>Turm</i>
	<i>Warte</i>
	<i>Wassermühle</i>
	<i>Wegweiser</i>
	<i>Windrad</i>
	<i>Windmühle (ehem.)</i>

Abkürzungen:

<i>Abl.</i>	<i>Ablage</i>	<i>Kol.</i>	<i>Kolonie</i>
<i>And. St.</i>	<i>Anlegestelle</i>	<i>K. D.</i>	<i>Kulturgechichtl. Denkm.</i>
<i>Bf.</i>	<i>Bahnhof</i>	<i>(M.)</i>	<i>Mühle, meistbar</i>
<i>B.W.</i>	<i>Bahnwärter</i>	<i>N.D.</i>	<i>Naturdenkmal</i>
<i>Chs.</i>	<i>Chausseehaus</i>	<i>N.S.G.</i>	<i>Naturschutzgebiet</i>
<i>D.M.</i>	<i>Dampfmühle</i>	<i>Ö.M.</i>	<i>Ölmühle</i>
<i>Dom.</i>	<i>Domäne</i>	<i>Pav.</i>	<i>Pavillon</i>
<i>Ehr.Fdhf.</i>	<i>Ehrenfriedhof</i>	<i>Sch.</i>	<i>Scheune</i>
	<i>für Krieger</i>	<i>(S.)</i>	<i>Schornstein, meistbar</i>
<i>Fbr.</i>	<i>Fabrik</i>	<i>Schp.</i>	<i>Schuppen</i>
<i>II.</i>	<i>Hütte</i>	<i>St.</i>	<i>Stall</i>
<i>Hp.</i>	<i>Haldepunkt</i>	<i>S.W.</i>	<i>Sägenwerk</i>
<i>Jg. Hb.</i>	<i>Jugendherberge</i>	<i>To.</i>	<i>Teerofen</i>
<i>K.O.</i>	<i>Kalkofen</i>	<i>Vro.</i>	<i>Vorrwerk</i>
<i>kbhf.</i>	<i>Kleinbahnhof</i>	<i>Ww.</i>	<i>Weihrauch</i>
<i>(K.)</i>	<i>Kirche, meistbar</i>	<i>Whs.</i>	<i>Wirtshaus</i>
<i>Kr.</i>	<i>Krug</i>	<i>Zgl.</i>	<i>Ziegelei</i>

Gewässer:



Höhenlinien:



Die Höhen sind in Metern über Normal-Null angegeben.

Abbildung A.6.: Legende eines Messtischblatts von 1933 (Fortsetzung)

Literaturverzeichnis

- [Bar89] BARNARD, STEPHEN T: *Stochastic stereo matching over scale*. International Journal of Computer Vision, 3(1):17–32, 1989.
- [Bes86] BESAG, JULIAN: *On the statistical analysis of dirty pictures*. Journal of the Royal Statistical Society. Series B (Methodological), Seiten 259–302, 1986.
- [BK04] BOYKOV, YURI und VLADIMIR KOLMOGOROV: *An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 26(9):1124–1137, 2004.
- [BKHS00] BANDLOW, THORSTEN, MICHAEL KLUPSCH, ROBERT HANEK und THORSTEN SCHMITT: *Fast image segmentation, object recognition and localization in a robocup scenario*. In: *RoboCup-99: Robot Soccer World Cup III*, Seiten 174–185. Springer, 2000.
- [Bra90] BRASSEL, KURT: *Geographische Informationssysteme: eine Einleitung*. Geographica helvetica, 45(4):143–144, 1990.
- [Bre96] BREIMAN, LEO: *Bagging Predictors*. In: *Machine Learning*, Band 24, Seiten 123–140, 1996.
- [Bre01] BREIMAN, LEO: *Random forests*. Machine learning, 45(1):5–32, 2001.
- [BVZ01] BOYKOV, YURI, OLGA VEKSLER und RAMIN ZABIH: *Fast approximate energy minimization via graph cuts*. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 23(11):1222–1239, 2001.
- [CM02] COMANICIU, DORIN und PETER MEER: *Mean shift: a robust approach toward feature space analysis*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(5):603–619, 2002.
- [CS13] CRIMINISI, ANTONIO und JAMIE SHOTTON: *Decision Forests for Computer Vision and Medical Image Analysis*. Springer Science & Business Media, 2013.
- [FF56] FORD, LESTER R und DELBERT R FULKERSON: *Maximal flow through a network*. Canadian journal of Mathematics, 8(3):399–404, 1956.
- [FH04] FELZENSZWALB, PEDRO F. und DANIEL P. HUTTENLOCHER: *Efficient Graph-Based Image Segmentation*. International Journal of Computer Vision, 59:167–181, 2004.

- [GL13] GALL, JÜRGEN und VICTOR LEMPITSKY: *Class-specific hough forests for object detection*. In: *Decision forests for computer vision and medical image analysis*, Seiten 143–157. Springer, 2013.
- [HBHH98] HORVITZ, E.J., J.S. BREESE, D. HECKERMAN und D. HOVEL: *The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users*. 1998.
- [HRHM11] HEROLD, HENDRIK, PATRIC ROEHM, ROBERT HECHT und GOTTHARD MEINEL: *Automatically Georeferenced Maps as a Source for High Resolution Urban Growth Analyses*. In: *Proceedings of the ICA 25th International Cartographic Conference, July*, Seiten 3–8, 2011.
- [IPV00] IKONOMAKIS, NICOLAOS, KONSTANTINOS N PLATANIOTIS und ANASTASIOS N VENETSANOPoulos: *Color image segmentation for multimedia applications*. Journal of Intelligent & Robotic Systems, 28(1):5–20, 2000.
- [JN02] JORDAN, M.I. und A.Y. Ng: *On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes*. 2002.
- [KWT88] KASS, M., A. WITKIN und D. TERZOPoulos: *Snakes: Active Contour Models*. International Journal of Computer Vision, 1(4):321–331, 1988.
- [KZ04] KOLMOGOROV, VLADIMIR und RAMIN ZABIN: *What energy functions can be minimized via graph cuts?* IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(2):147–159, 2004.
- [LSR⁺15] LIN, GUOSHENG, CHUNHUA SHEN, IAN REID et al.: *Efficient piecewise training of deep structured models for semantic segmentation*. arXiv preprint arXiv:1504.01013, 2015.
- [Mei09] MEINEL, GOTTHARD: *Konzept eines Monitors der Siedlungs- und Freiraumentwicklung auf Grundlage von Geobasisdaten*. Flächennutzungsmonitoring. Konzepte, Indikatoren, Statistik, Seiten 177–194, 2009.
- [MHH09] MEINEL, GOTTHARD, ROBERT HECHT und HENDRIK HEROLD: *Analyzing building stock using topographic maps and GIS*. Building Research & Information, 37(5-6):468–482, 2009.
- [NL11] NOWOZIN, SEBASTIAN und CHRISTOPH H. LAMPERT: *Structured Learning and Prediction in Computer Vision*. Foundations and Trends® in Computer Graphics and Vision, 6(3–4):185–365, 2011.
- [NRB⁺11] NOWOZIN, SEBASTIAN, CARSTEN ROTHER, SHAI BAGON, TOBY SHARP, BANGPENG YAO und PUSHMEET KOHLI: *Decision tree fields*. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*, Seiten 1668–1675. IEEE, 2011.

- [NW02] NEUBERT, MARCO und ULRICH WALZ: *Auswertung historischer Kartenwerke für ein Landschaftsmonitoring*. Angewandte Geographische Informationsverarbeitung, 14:396–402, 2002.
- [Pak69] PAKES, AG: *Some conditions for ergodicity and recurrence of Markov chains*. Operations Research, 17(6):1058–1061, 1969.
- [PXP00] PHAM, DZUNG L., CHENYANG XU und JERRY L. PRINCE: *Current Methods in Medical Image Segmentation*. Annual Review of Biomedical Engineering, 2(1):315–337, 2000. PMID: 11701515.
- [RKB04] ROTHER, CARSTEN, VLADIMIR KOLMOGOROV und ANDREW BLAKE: "GrabCut": *Interactive Foreground Extraction Using Iterated Graph Cuts*. ACM Trans. Graph., 23(3):309–314, August 2004.
- [RM03] REN, XIAOFENG und JITENDRA MALIK: *Learning a classification model for segmentation*. In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, Seiten 10–17. IEEE, 2003.
- [RN04] RUSSELL, STUART und PETER NORVIG: *Künstliche Intelligenz: Ein moderner Ansatz*. Pearson, 2 Auflage, 2004.
- [Shi94] SHIMONY, SOLOMON EYAL: *Finding MAPs for belief networks is NP-hard*. Artificial Intelligence, 68(2):399–410, 1994.
- [SM11] SUTTON, CHARLES und ANDREW McCALLUM: *An Introduction to Conditional Random Fields*. Machine Learning, 4(4):267–373, 2011.
- [SW49] SHANNON, CLAUDE E. und WARREN WEAVER: *The Mathematical Theory of Information*. 1949.
- [SWS⁺⁰⁰] SMEULDERS, ARNOLD W. M., M. WORRING, S. SANTINI, A. GUPTA und R. JAIN: *Content-based image retrieval at the end of the early years*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(12):1349–1380, Dezember 2000.
- [Sze10] SZELISKI, RICHARD: *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., 1. Auflage, 2010.
- [Tsc12] TSCHUMPERLÉ, DAVID: *The CImg Library*. In: *IPOL 2012 Meeting on Image Processing Libraries*, Seiten 4–pp, 2012.
- [YFW00] YEDIDIA, JONATHAN S, WILLIAM T FREEMAN und YAIR WEISS: *Generalized belief propagation*. In: *NIPS*, Band 13, Seiten 689–695, 2000.

Abbildungsverzeichnis

2.1. Ein Beispiel-Faktorgraph mit daraus abgeleitetem Hilfsgraphen	17
2.2. Hilfsgraph mit Schnitt	18
3.1. Die Benutzung des Programms in Python	22
4.1. Beispiel für Siedlungsgebiet	24
4.2. Beispiel für Nicht-Siedlungsgebiet	25
4.3. Testleistung abhängig von der maximalen Tiefe	27
4.4. Testleistung abhängig von der Anzahl der Entscheidungsbäume	28
4.5. Testleistung abhängig von der Fensterbreite	29
4.6. Testleistung abhängig von der Anzahl der Versuche beim Sampeln der Testfunktionen	30
4.7. Testleistung abhängig von der Anzahl der Trainingspixel	31
4.8. Testleistung für verschiedene Werte von e	32
4.9. Beispiel für die Wirkung der Regularisierungskonstante e bei der Inferenz mit Maxflow	33
4.10. Beispiel für die Wirkung der Anzahl der Samplingschritte beim Gibbs-Sampling	34
4.11. Testleistung für verschiedene Werte von e	34
4.12. Durchschnittliche Trainings- und Inferenzdauer abhängig von der maximalen Tiefe	35
4.13. Durchschnittliche Trainings- und Inferenzdauer abhängig von der Anzahl der Entscheidungsbäume	35
4.14. Durchschnittliche Trainings- und Inferenzdauer abhängig von der Fensterbreite	36
4.15. Durchschnittliche Trainings- und Inferenzdauer abhängig von der Anzahl der Versuche beim Sampeln der Testfunktionen	36
4.16. Inferenzdauer abhängig von der Anzahl der Schritte beim Gibbs-Sampling	37
A.1. Beispiel-Inferenz	40
A.2. Eine weitere Beispiel-Inferenz	41
A.3. Die Ground truth für Abbildung A.1	42
A.4. Die Ground truth für Abbildung A.2	42
A.5. Legende eines Messtischblatts von 1933	43
A.6. Legende eines Messtischblatts von 1933 (Fortsetzung)	44

Tabellenverzeichnis

4.1. Optimale Trainingsparameter für die hier verwendeten Trainingsdaten 26

Algorithmenverzeichnis

1.	Inferenz in Random Forests	7
2.	Training von Random Forests	9
3.	Gibbs-Sampling	19