HAW HAMBURG

Masterarbeit

Daniel Osterholz

Evaluierung von spezialisierten und generischen Modellen zur Identifikation von städtischen Einzelbäumen mittels Transfer Learning

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

Daniel Osterholz

Evaluierung von spezialisierten und generischen Modellen zur Identifikation von städtischen Einzelbäumen mittels Transfer Learning

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Clemen
Zweitgutachter: Prof. Dr. Marina Tropmann-Frick

Eingereicht am: 29. Oktober 2024

Daniel Osterholz

Thema der Arbeit

Evaluierung von spezialisierten und generischen Modellen zur Identifikation von städtischen Einzelbäumen mittels Transfer Learning

Stichworte

DL, DNN, Transfer Learning, Fernerkundung, Satellitenbilder, Baumkronensegmentierung, urbane Wälder, SAM, DeepForest, CNN, RetinaNet, Instanzsegmentierung, RGB-Bilder, LiDAR, Feinabstimmung, Modellleistung, Bildverarbeitung, Nachhaltigkeit, SDG 15

Kurzzusammenfassung

Die anhaltende Urbanisierung und der Klimawandel erhöhen die Dringlichkeit, eine zuverlässige und funktionsfähige Methode zur Überwachung und Pflege urbaner Wäldern zu entwickeln. Eine präzise automatisierte Möglichkeit der Segmentierung von Baumkronen wird immer wichtiger, um diese Ziele zu erreichen. Diese Arbeit untersucht die Leistungsfähigkeit generischer, leicht zugänglicher, Modelle (Segment Anything Model (SAM)) und deren Anpassung an spezifische Domänen, wie die Erkennung von Baumkronen in urbanen Wäldern. Zur Bewertung der Leistungsfähigkeit wird ein auf diese Domäne spezialisiertes Modell (DeepForest (DF)) mit einem fein abgestimmten generischen Modell verglichen. SAM wurde mithilfe von Transfer Learning auf die Segmentierung von Baumkronen in urbanen Wäldern auf RGB-Satellitenbildern fein abgestimmt und anschließend in verschiedenen Schwierigkeitskategorien sowie in unterschiedlichen Floren DF gegenübergestellt. Hierfür wurden Experimente in verschiedenen Großstädten weltweit durchgeführt, um die Generalisierbarkeit der Modelle zu evaluieren. Die Ergebnisse zeigen, dass das feinabgestimmte SAM in vielen Fällen vergleichbare oder sogar bessere Ergebnisse als das spezialisierte DF erzielte. Diese Arbeit trägt zur Bewertung der Leistungsfähigkeit von Basismodellen und deren Einsatzmöglichkeiten in der Fernerkundung bei. Die gewonnenen Erkenntnisse unterstützen Stadtplaner und Umweltbehörden bei der automatisierten Überwachung und Pflege städtischer Grünflächen, was wiederum den Zielen der nachhaltigen Stadtentwicklung entspricht.

Daniel Osterholz

Title of Thesis

Evaluation of specialized and generic models for the identification of individual urban trees using transfer learning

Keywords

DL, DNN, transfer learning, remote sensing, satellite images, tree canopy segmentation, urban forests, SAM, DeepForest, CNN, RetinaNet, instance segmentation, RGB images, LiDAR, fine-tuning, model performance, image processing, sustainability, SDG 15

Abstract

Ongoing urbanisation and climate change increase the urgency to develop a reliable and functional method for monitoring and maintaining urban forests. A precise automated canopy segmentation capability is becoming increasingly important to achieve these goals. This thesis investigates the performance of generic, easily accessible, models (Segment Anything Model (SAM)) and their adaptation to specific domains, such as tree canopy detection in urban forests. To evaluate the performance, a model specialised for this domain (DeepForest (DF)) is compared with a finely tuned generic model. SAM was fine-tuned using transfer learning on the segmentation of tree crowns in urban forests on RGB satellite images and then compared to DF in different difficulty categories and in different floras. For this purpose, experiments were conducted in various large cities worldwide to evaluate the generalisability of the models. The results show that in many cases the fine-tuned SAM achieved comparable or even better results than the specialised DF. This work contributes to the evaluation of the performance of base models and their possible applications in remote sensing. The knowledge gained supports urban planners and environmental authorities in the automated monitoring and maintenance of urban green spaces, which in turn corresponds to the goals of sustainable urban development.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Hintergrund und Motivation

Seit Jahren findet in vielen Ländern der Welt eine starke Urbanisierung statt[20, 8, 46]. Dieser Wandel kann als einer der bedeutendsten Veränderungsprozesse im globalen Maßstab betrachtet werden[14]. Die Ballungsgebiete wachsen weiter, wodurch auch die Belastungen auf die vorhandene Infrastruktur zunimmt[48, 35, 36, 39]. Zu dieser Infrastruktur gehören auch die urbanen Wälder, welche aus Grünanlagen und den darin befindlichen Bäumen bestehen. Diese haben einen erheblichen Einfluss auf die Lebensqualität der Einwohner, was sich in vielfältiger Weise zeigt: von der Verbesserung des Mikroklimas und der Reduzierung von Lärm bis hin zur positiven Beeinflussung der Gesundheit der Stadtbewohner[13, 9, 52, 57]. Die genaue Überwachung von Einzelbäumen in urbanen Wäldern stellt daher eine wichtige Aufgabe für Stadtplaner, Umweltbehörden und andere Stakeholder dar.

Eine möglichst präzise Segmentierung von Baumkronen versetzt die genannten Stakeholder in die Lage, den Bestand und die Gesundheit der Stadtbäume zu bewerten, zu überwachen und ggf. Maßnahmen zu ergreifen, um die Straßensicherheit zu gewährleisten. Die so ermöglichte Pflege urbaner Wälder trägt unmittelbar zu der Erreichung der Sustainable Development Goals (SDGs) bei, insbesondere des Ziels „Leben an Land“ (SDG 15), welches sich mit dem Schutz, der Wiederherstellung und der nachhaltigen Nutzung von Landökosystemen und Wäldern beschäftigt[24]. Diese Überwachung und anschließende Pflege gewinnen angesichts des aktuellen Klimawandels zunehmend an Bedeutung, da urbane Ökosysteme hochkomplexe und zerbrechliche Konstrukte darstellen. Es ist kaum vorhersehbar, welche Auswirkungen eine erneute Hitzewelle, insbesondere in den großen Städten dieser Welt, auf diese empfindlichen Ökosysteme haben wird und wie lange die urbanen Wälder solchen Belastungen standhalten können.

Die Erfassung solcher Wälder wird vielerorts manuell vorgenommen, was angesichts des Fachkräftemangels und notwendiger Expertise kaum zu bewältigen ist[38, 15]. Gerade

kleine Städte können diesen Kraftakt nur bedingt oder gar nicht erfüllen. Die Segmentierung der Wälder mithilfe geeigneter Technologie ist somit naheliegend, stellt jedoch gerade in städtischen Gebieten hohe Anforderungen an die verwendeten Modelle. Anders als in klassischen Wäldern, bei denen die Herausforderung vor allem darin besteht, die Baumkronen in einem relativ homogenen Umfeld zu trennen, bringen urbane Wälder mit ihrer Vielzahl unterschiedlichster Objekte und Strukturen ganz eigene Probleme mit sich. In dieser heterogenen Umgebung muss ein Modell in der Lage sein, die Bäume zuverlässig und präzise zu identifizieren – etwas, das mit herkömmlichen Clusteralgorithmen oder schwellenwertbasierten Ansätzen nicht oder nur ungenügend bewerkstelligt werden kann. Moderne Bildsegmentierungstechniken, unter Verwendung von tiefen neuronalen Netzen, zeigen hier große Fortschritte bei der zuverlässigen Segmentierung von Bäumen in urbanen Wäldern. Dennoch bleibt die Segmentierung weiterhin eine Herausforderung der aktuellen Zeit[23, 59, 34].

Die letzten Jahre hat sich im Bereich der Bildsegmentierung durch den Einsatz von CNNs eine deutliche Verbesserung gezeigt[34, 25]. Diese Modelle, insbesondere spezialisierte Ansätze wie DeepForest[54], wurden für die Bewältigung der Aufgabe der Segmentierung von Baumkronen entwickelt und werden erfolgreich in verschiedenen Waldökosystemen eingesetzt[4, 44]. DeepForest verwendet hochauflösende Satelliten- und Luftbilder sowie LiDAR-Daten, um die Baumkronen zu segmentieren und einzelne Bauminstanzen zu erkennen[54]. Jedoch ist es ressourcenintensiv, für jede Fachdomäne ein eigenes, darauf spezialisiertes Modell zu trainieren. Daher stellt sich die Frage, ob ein fein abgestimmtes Basismodell ähnliche oder sogar bessere Ergebnisse liefern kann.

Der Ansatz, ein Basismodell für eine spezialisierte Domäne durch Feinabstimmung anzupassen, wird gerade in den letzten Jahren immer häufiger umgesetzt, wie das vermehrte Auftreten von Basismodellen zeigt. Mittels Transfer Learning wird ein vortrainiertes Modell, welches auf einer breiten Datenbasis trainiert wurde, für spezifische Aufgaben wie die Baumkronensegmentierung fein abgestimmt. Durch dieses Vorgehen lassen sich auch mit kleineren Datensätzen für spezifische Aufgaben gute Ergebnisse erzielen. Die Idee hierbei ist, die starke allgemeine Datenbasis um einen spezialisierten Anteil zu ergänzen[55]. Das Segment Anything Model (SAM) ist ein Beispiel für ein solches generisches Basismodell, welches durch die Feinabstimmung auf andere Aufgaben angepasst werden kann. Diese Feinabstimmung kann, wie in dieser Arbeit beschrieben, auch die Transformation von semantischer Segmentierung hin zu Instanzsegmentierung umfassen[27].

Die Motivation dieser Arbeit ist es, den tatsächlichen Nutzen von generischen Modellen gegenüber einem Spezialmodell zu zeigen. Hierfür wurde die Segmentierung von Baumkronen auf hochauflösenden RGB-Satellitenbildern in urbanen Wäldern gewählt. Diese Domäne beinhaltet sehr komplexe Aufgabenstellungen für Segmentierungsalgorithmen und Modelle. Das Ziel dieser Arbeit ist es, experimentell zu zeigen, dass fein abgestimmte Basismodelle selbst in anspruchsvollen Domänen spezialisierten Modellen mindestens ebenbürtig sind. Es wird erwartet, dass mithilfe einer verhältnismäßig kleinen Datenmenge zur Feinabstimmung von SAM die Resultate mindestens ebenbürtig gegenüber den Resultaten von DeepForest sind. Eine der Herausforderungen des experimentellen Designs besteht darin, die Leistungsfähigkeit auch in verschiedenen Floren und Städten zu testen, um die Aussagekraft der Ergebnisse zu stärken, ohne dass für jede Stadt Modellanpassungen notwendig sind.

Die Implementierung des Rahmenwerks und der Methoden zur Untersuchung der Leistungsfähigkeit von Basismodellen gegenüber Spezialmodellen leistet einen wichtigen Beitrag für die oben beschriebenen Stakeholder bei der Automatisierung und Analyse urbaner Wälder. Die dadurch ermöglichte einfache und zuverlässige Erkennung von Bauminstanzen in verschiedenen Städten unter Verwendung frei verfügbarer Bilder und Daten trägt zur barrierefreien und gut zugänglichen Domäne der Fernerkundung bei. Dies wird es weiteren interessierten Forschern und neuen Stakeholdern ermöglichen, städtische Floren zu analysieren und so den Erhalt von Grünflächen und urbanen Wäldern zu unterstützen, was wiederum den Zielen der nachhaltigen Stadtentwicklung und dem SDG 15 „Leben an Land “ entspricht.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist es, die Ergebnisse eines bekannten generischen Basismodells mit den Segmentierungsergebnissen eines Spezialmodells zu vergleichen. Dies wird im Bereich der Segmentierung von Baumkronen in urbanen Wäldern implementiert, bei dem die Ergebnisse des generischen Modells (SAM) evaluiert und mit einem spezialisierten Modell (DeepForest) verglichen werden. Konkret wird untersucht, ob SAM durch Feinabstimmung auf urbane Baumkronensegmente vergleichbare oder sogar bessere Ergebnisse erzielen kann als DeepForest, ein Spezialmodell, welches speziell für die Segmentierung von Baumkronen auf RGB-Satellitenbildern entwickelt wurde.

Eine Herausforderung wird es sein, innerhalb der urbanen Umgebung die Baumkronen sicher zu identifizieren, da sich in ihrer unmittelbaren Nähe weitere Objekte befinden. Diese komplexen Umgebungsbedingungen werden sowohl das fein abgestimmte Basismodell als auch das speziell für diese Umgebung trainierte Modell an ihre Grenzen bringen[58].

Das SAM unterstützt durch seine Architektur eine Vielzahl von Segmentierungsaufgaben. Es ist in seiner Originalversion jedoch nicht in der Lage, Instanzen zu segmentieren. Es ist lediglich in der Lage, Bildpixel voneinander zu trennen und zu entscheiden, welche der Pixel ggf. zu einem Objekt zusammenzufassen sind. Es ist weder fähig, Instanzen zu identifizieren, noch ist es eindeutig (Bildpixel können zu verschiedenen Objekten gehören) und es besitzt auch keine Labels, welche Klassen suggerieren. Daher muss eine Erweiterung des SAM implementiert werden, welches ohne Nutzereingaben in der Lage ist, die klassenlosen Segmente zu unterscheiden und sie als *Baum* oder *kein Baum* zu identifizieren. Dies erfolgt mithilfe eines kleinen spezialisierten Datensatzes, welcher aus einer Anzahl von Baumkronen auf RGB-Satellitenbildern besteht.

Neben der Untersuchung der Segmentierungsqualität in urbanen Wäldern, wird zudem die Generalisierbarkeit der Modelle in unterschiedlichen geographischen Ausdehnungen untersucht. Hierfür werden geographisch weit auseinanderliegende Gebiete mit unterschiedlichen Floren in die Experimente mit aufgenommen. Auf diese Weise soll gezeigt werden, dass eine Feinabstimmung nicht nur innerhalb des gleichen geographischen Bereichs stattfinden muss wie der verwendete Trainingsdatensatz. Die gewählten Städte sind Hamburg (hier auch der Trainingsdatensatz), Kapstadt, San Francisco und Tokio. Es wird erwartet, dass auch in den Bereichen, in denen keine Feinabstimmung vorgenommen wurde, eine verbesserte Leistung erzielt wird. Diese Erwartung basiert auf der Tatsache, dass auf RGB-Bildern eine Flora maximal durch ihre Form unterschieden werden kann. Daher wird vermutet, dass die Unterschiede in den Formen der Baumkronen in unterschiedlichen Teilen der Erde nicht ausschlaggebend genug sind, um das Segmentierungsergebnis signifikant zu verschlechtern.

Durch die Implementierung der Arbeit und die Durchführung der Experimente soll ein Beitrag dazu geleistet werden, ob es sinnvoll und zielführend ist, an der Implementierung und dem Training von Basismodellen festzuhalten oder ob die Nutzung von Spezialmodellen das sinnvollere Herangehen ist.

1.3 Forschungsfrage und Hypothese

Die zentralen Forschungsfragen dieser Arbeit lauten: Kann ein generisches, fein abgestimmtes Basismodell (SAM) mindestens die Leistungsfähigkeit eines spezialisierten Modells (DeepForest) in der Segmentierung von Einzelbäumen in urbanen Wäldern erreichen?

Hypothesen

- Hypothese 1 (**H1**): Ein fein abgestimmtes Basismodell ist mindestens genauso leistungsfähig wie ein spezialisiertes Modell im Bereich der Segmentierung von Baumkronen in urbanen Wäldern.
- Hypothese 2 (**H2**): Die Feinabstimmung eines Basismodells zur Identifikation von Baumkronen in urbanen Wäldern mittels RGB-Bildern sollte einen positiven Einfluss auf die Segmentierungsergebnisse in einer anderen Flora haben, da auf RGB-Bildern eine Flora nicht zwingend von einer anderen zu unterscheiden ist. Die Ergebnisse eines Basismodells in einer unbekannten Flora sollten zumindest vergleichbar mit denen eines spezialisierten Modells sein.

1.4 Forschungslücke

Obwohl in der aktuellen Forschung immer wieder neue Basismodelle vorgestellt werden, gibt es nur spärliche Literatur zur tatsächlichen Leistungsfähigkeit fein abgestimmter Basismodelle im Verhältnis zu ihren spezialisierten Pendants. Bei den Vergleichen der Leistungsfähigkeit unterschiedlicher Modelle wird zudem häufig ausschließlich über die *Accuracy* gearbeitet, welche potenziell zu Fehlschlüssen führen kann, da sie kein umfassendes Bild der Leistungsfähigkeit bietet. Diese Arbeit zielt darauf ab, diese Lücke zu schließen und die Anwendungsmöglichkeiten von Basismodellen im Bereich der urbanen Baumerkennung zu erweitern. Sie soll somit einen Indikator dafür bieten, inwieweit die Leistungsfähigkeit eines fein abgestimmten Basismodells mit der eines spezialisierten Modells vergleichbar ist.

2 Methoden

Im Abschnitt „Methoden “ werden alle verwendeten Algorithmen und Modelle, die nicht eigenständig implementiert wurden, genau beschrieben. Diese werden ausführlich erläutert, um die Erkenntnisse aus der Implementierung optimal nutzen zu können. Darüber hinaus wird ein Einblick in verwandte Arbeiten gegeben und der Unterschied zu dieser Arbeit aufgezeigt.

2.1 Verwandte Arbeiten

Das Forschungsgebiet der Erkennung von Bäumen, auch in urbanen Wäldern, ist sehr aktiv. In aktuellen Veröffentlichungen werden spezialisierte Modelle sowie fein abgestimmte generische Modelle genutzt, um gute Segmentierungsergebnisse zu erhalten. Die nachfolgenden Arbeiten geben einen kleinen Einblick in aktuelle Veröffentlichungen zu diesen Themen, erheben jedoch keinen Anspruch auf Vollständigkeit.

Titel	**Autoren**	**Veröffentlichung**	**PDF**
GeoSAM: Fine-tuning SAM...	Sultan R. I. et al.	19.11.2023	PDF
Tree-GPT: Modular Large...	Du S. et al.	07.10.2023	PDF
Segment Anything in H...	Ke L. et al.	02.06.2023	PDF
Individual Tree-Crown...	Beloiu M. et al.	06.03.2023	PDF
Tree Detection and Species...	Sivanandam P. et al.	12.08.2022	PDF

Tabelle 2.1: Veröffentlichungen über Baumerkennung und Segmentierungsmodelle

GeoSAM: Fine-tuning SAM with sparse and dense visual prompting for automated segmentation of mobility infrastructure: haben das SAM für geospezifische Daten angepasst, um die verbesserte Segmentierfähigkeit auf Satellitenbildern zu ermöglichen[49].

Tree-GPT: Modular Large Language Model Expert System for Forest Remote Sensing Image Understanding and Interactive Analysis: hat einen modularen Modellansatz gewählt, welcher große LLM Modelle wie GPT in die Segmentierung von Bäumen integriert. Dies stellt einen Versuch dar, ein Basismodell für spezielle Domänen fein abzustimmen[11].

Segment Anything in High Quality: ist ein Modell, welches die ursprüngliche hier verwendete Version des SAM noch für hochauflösende Bilder optimieren soll. Die Autoren sind dieselben wie die Autoren des hier referenzierten SAM-Papers[25].

Individual tree-crown detection and species identification in heterogeneous forests using aerial RGB imagery and deep learning: fokussiert sich auf die Segmentierung von Baumkronen als Instanzen in städtischen Bereichen und kommt somit in dem gleichen Einsatzgebiet wie diese Arbeit zum Einsatz. Das hier genutzte Modell ist ein speziell für diesen Bereich konzipiertes Modell und fällt somit in die Definition eines Spezialmodells[5].

Tree Detection and Species Classification in a Mixed Species Forest Using Unoccupied Aircraft System (UAS) RGB and Multispectral Imagery: gehen einen Schritt weiter und untersuchen die Möglichkeit der Segmentierung von Bauminstanzen und der Unterscheidung der Arten. Diese Arbeit setzt somit voraus, dass Bauminstanzen solide und zuverlässig segmentiert werden können[44].

Alle in diesem Abschnitt beschriebenen Veröffentlichungen befassen sich mit der Herausforderung der Erkennung von Bäumen. Es zeigt klar, dass es sich hierbei um einen aktiven Forschungszweig handelt und es noch keine abschließende Lösung der Herausforderungen der Segmentierung von Bäumen gibt. Diese Arbeit versucht die Forschungslücke zu füllen, welche erst einmal klären soll, ob es sinnvoll ist, ein Basismodell einem Spezialmodell vorzuziehen. In den oben genannten Veröffentlichungen wird diese Frage nicht behandelt und es wird sich für die eine oder andere Modellart entschieden.

2.2 DeepForest (Spezial-Modell)

DeepForest wurde 2020 von einer Forschungsgruppe zur Erkennung und Segmentierung von Baumkronen in Luftaufnahmen vorgestellt[54]. Es ist in der Lage, Instanzen von Baumkronen auf Luftbildaufnahmen und Satellitenbildern zu segmentieren. Bei dem Modell handelt es sich um ein spezielles RetinaNet, welches mit einer großen Menge an hochauflösenden Bildmaterialien trainiert wurde. Innerhalb des Modells werden unterschiedliche Ansätze an Algorithmen genutzt, um bekannte Probleme von sehr tiefen neuronalen Netzen abzuschwächen bzw. zu lösen. Hierzu gehören Skip Connections, die Fokal-Loss Funktion und weitere Techniken. Nachfolgend werden alle Komponenten sowie die Datengrundlage des Modells genau erläutert.

Datengrundlage und Training Als Datengrundlage für das DeepForest Modell dienten hochauflösende RGB Luftaufnahmen von Baumkronen. Für die Erstellung der Segmente (Baumkronen) wurde eine Mischung aus algorithmisch generierten und manuell beschrifteten Baumkronen genutzt. Die algorithmisch erstellten Baumkronen wurden mithilfe von LiDAR-Daten[29] erstellt und stammen aus 22 unterschiedlichen Waldgebieten. Um die Genauigkeit der Vorhersagen weiter zu verbessern, wurden 10.000 manuell beschriftete Baumkronen aus sechs weiteren Wäldern für das Training verwendet.

2.2.1 Architektur

Das RetinaNet ist ein Netz zur Objekterkennung, das durch seine Architektur dazu geeignet ist, um Klassenungleichgewichte mithilfe der Fokal-Loss Funktion auszugleichen. Es besteht aus einem Backbone-Netz (Abschn.: 2.2.2), einem Feature-Pyramid-Netz (Abschn.: 2.2.3) sowie zwei spezialisierten Teilnetzen. Das erste Teilnetz ist für die Klassifizierung zuständig (Abschn.: 2.2.4), während das zweite für die Optimierung der Ankeroxen verantwortlich ist(Abschn.: 2.2.5).

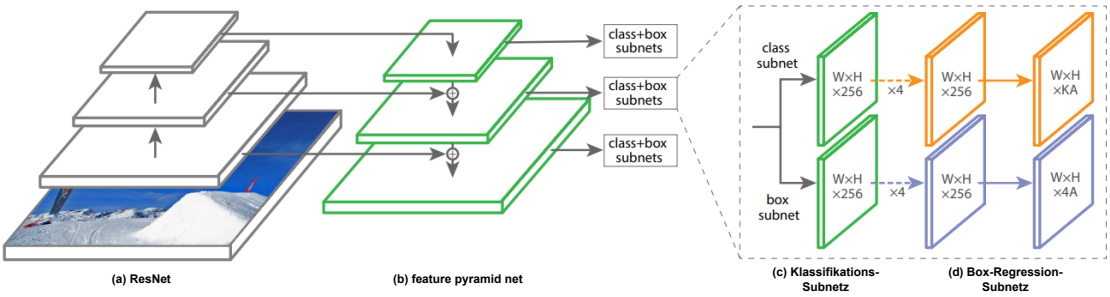

Abbildung 2.1: Die schematische Darstellung eines RetinaNet[32]

Auf dem Bild (Abb.: 2.1) ist die Netzarchitektur eines typischen RetinaNet, wie im Paper [32] beschrieben, zu sehen. Es ist speziell für die Identifikation von Segmenten auf Bildern konzipiert. Die Architektur besteht von links nach rechts aus folgenden Einzelkomponenten[30, pp. 223-239]:

(a) Ein ResNet50-Netz dient als Backbone des RetinaNet und nimmt das Eingabebild auf. Es extrahiert auf verschiedenen Ebenen unterschiedlich granulare Merkmale und gibt sie an die nächste Schicht weiter. Zusätzlich zu der Verarbeitung im Backbone werden die Merkmalskarten auf bestimmten Schichten an ein FPN weitergegeben.

(b) Ein Feature-Pyramid-Netzwerk (FPN) empfängt auf bestimmten definierten Ebenen Merkmalskarten sowie das finale Ergebnis des Backbone ResNet50. Es integriert die Informationen aus einem „top-down“ Pfad mit den Verbindungen aus früheren Schichten, um Merkmalskarten mit Informationen aus verschiedenen Auflösungsebenen zu erzeugen.

(c) Das Klassifikations-Subnetz besteht aus vier 3×3 Convolutional-Schichten mit jeweils 256 Kanälen, gefolgt von einer ReLU-Aktivierung. Anschließend folgt eine fünfte Convolutional-Schicht, welche die Klassenvorhersagen für die verschiedenen Ankerboxen ausgibt. Für jede Ankerbox wird pro Klasse eine Sigmoid-Aktivierung durchgeführt, um eine Multi-Label-Klassifikation zu ermöglichen. Die Ausgabegröße beträgt $W \times H \times KA$, wobei A die Anzahl der Ankerboxen pro Position und K die Anzahl der Klassen ist.

(d) Das Box-Regression-Subnetz besteht ebenfalls aus vier 3×3 Convolutional-Schichten mit jeweils 256 Kanälen und ReLU-Aktivierungen, hierbei jedoch gefolgt von einer

Convolutional-Schicht, welche die Box-Offsets $(\Delta x, \Delta y, \Delta w, \Delta h)$ für jede Ankerbox vorhersagt. Die Ausgabegröße beträgt $W \times H \times 4A$, wobei A die Anzahl der Ankerboxen pro Position ist.

2.2.2 ResNet50

Das ResNet50 fungiert als Backbone-Netz des RetinaNet. Es ist eine spezielle Form eines neuronalen Faltungsnetzes (CNN[7]) und wird als Residualnetz (ResNet[28]) bezeichnet. Ein ResNet50 besteht aus 50 Schichten und fällt damit in die Kategorie der tiefen neuronalen Netze (mehr als 2 versteckte Schichten). Mithilfe der tiefen Architektur werden Merkmale unterschiedlicher Auflösung aus dem Eingabebild extrahiert. Der Vorteil der Nutzung eines ResNet als Backbone liegt in der Fähigkeit des Netzes, trotz der sehr tiefen Architektur das Problem des verschwindenden Gradienten durch Einführung der Residualblöcke (Abbschn.: 2.2.2) zu vermindern oder sogar zu umgehen.

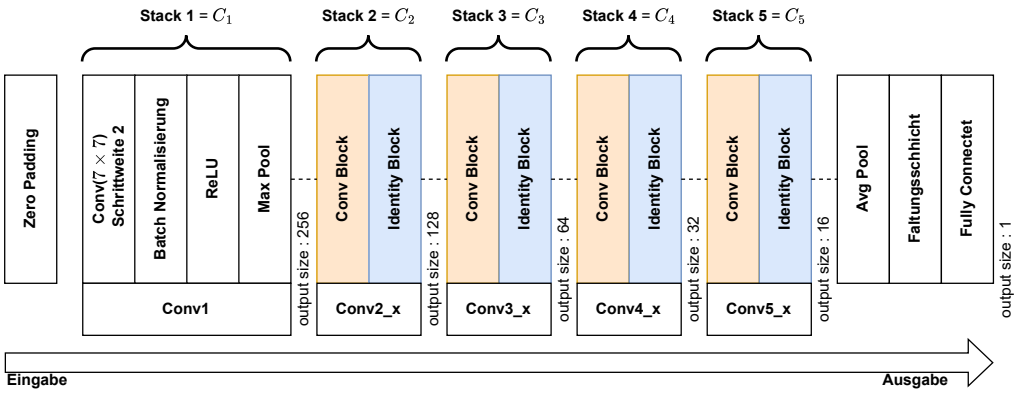

Abbildung 2.2: ResNet Backbone (angelehnt an [18])

In der Abb.: 2.2 ist eine abstrakte Darstellung eines Residualnetzes zu sehen. Die Eingabe des Netzes ist ein Bild I, welches durch eine Sequenz von Verarbeitungsschritten in Merkmalskarten unterschiedlicher Auflösung überführt wird. Jede Schicht hat dabei eine bestimmte Aufgabe, diese sind wie folgt zusammenzufassen:

Auf das Bild I wird Zero Padding [1] angewendet. Bei einem Bild I der Dimension $H \times W$ fügt das Zero Padding an allen Seiten von I eine bestimmte Anzahl von Zeilen und Spalten mit Nullen hinzu. Damit hat I nun die Dimension $(H + 2P) \times (W + 2P)$, wobei P die Paddinggröße ist. Der Zweck des Paddings ist es, 1. Die Größe der Merkmalskarten nach

einer Faltung zu erhalten 2. Die Kanteninformationen zu bewahren 3. Eine zu schnelle Verkleinerung der Merkmalskarten zu vermeiden.

Die anschließende Batch-Normalisierung $BN(x)$ wird nach jeder Convolutional-Schicht und vor der ReLU-Aktivierungsfunktion[3] angewendet. Diese Normalisierungsoperation findet im Anschluss an den *Conv1-Block* und innerhalb jedes Convolutional und Identity Blocks (sogenannte Residualblöcke) statt. Sei $F(x)$ eine Funktion, die für jede Merkmalskarte separat die mittlere Aktivierung auf 0 und die Standardabweichung auf 1 bringt, so wird $BN(F(x)) = \gamma \left(\frac{F(x)-\mu_B}{\sqrt{\sigma_B^2+\epsilon}} \right) + \beta$, wobei μ_B und σ_B^2 der Mittelwert und die Varianz des Batches sind, γ und β lernbare Parameter für Skalierung und Verschiebung und ϵ eine kleine Konstante zur Vermeidung von Division durch null. Mithilfe dieser Operationen wird das Problem der internen kovarianten Verschiebung reduziert und die Konvergenz während des Trainings beschleunigt. Die Vorteile der Funktion $BN(x)$ sind folgende: 1. Die Reduzierung der internen kovarianten Verschiebungen. Ohne $BN(x)$ ist es möglich, dass sich die Parameterverteilungen in den tieferen Schichten des Netzes während des Trainings verschieben. Ohne $BN(x)$ verlangsamt sich das Training und das Netz wird anfälliger für die initiale Wahl der Parameter sowie die gewählte Lernrate. Das Ergebnis von $BN(x)$ ist somit eine Standardisierung der Ausgaben jeder Schicht um den Mittelwert 0 und die Varianz von 1. 2. Das Training wird beschleunigt, da nicht mehr zwingend auf sehr kleine Lernraten zurückgegriffen werden muss, um die Robustheit des Modells zu gewährleisten. Wie in Punkt 1 beschrieben, wird das Netz während des Trainings stabiler. 3. Die Regularisierung wird verbessert, da das Problem des Overfitting gemindert wird.

Stufe 1 (Conv1) in Abb.: 2.2 zeigt die initiale Verarbeitungsstufe. In dieser wird eine Convolutional-Schicht genutzt, um die Eingabe auf eine 64-Merkmalskarte zu reduzieren. Es wird ein 7×7 Faltungskern verwendet, wobei die Faltung mit einem Schritt (Stride) von 2 ausgeführt wird, was zu einer räumlichen Reduzierung der Merkmalskarte führt. Mathematisch kann diese Operation als $F_{conv1}(x) = W_{conv1} * x + b_{conv1}$ beschrieben werden, wobei x das Eingabebild, W_{conv1} die Faltungskerne und b_{conv1} der Bias-Vektor sind.

Stufe 2 (Conv2_x) in Abb.: 2.2 lässt die Netzwerkkapazität wachsen und die Anzahl der Merkmalskarten erhöhen. Diese Stufe enthält 3 Residualblöcke, wobei jeder Block 256 Merkmalskarten erzeugt. Die Operation eines Blocks kann als $F_{conv2_x}(x) = \sigma(F_3(F_2(\sigma(F_1(x))))) + x$ beschrieben werden, wobei σ die ReLU-Aktivierungsfunktion

ist und F_1, F_2, F_3 die Faltungsschichten innerhalb eines Blocks darstellen. Der erste Residualblock der Schicht ist ein Conv-Blcok, welcher den Input x durch eine 1x1-Convolution anpasst, um die Dimension der Ausgabe so zu verändern, dass es in dem aktuellen Block mit Identity-Blöcken verarbeitet werden kann.

Stufe 3 (Conv3_x) in Abb.: 2.2 besteht aus 4 Residualblöcken, wobei jeder Block 512 Merkmalskarten erzeugt. Der Aufbau und die Formel sind analog zu Stufe 2, jedoch mit einer höheren Anzahl an Merkmalskarten.

Stufe 4 (Conv4_x) in Abb.: 2.2 besteht aus 6 Residualblöcken, jeder Block erzeugt 1024 Merkmalskarten. Auch hier bleibt der Aufbau identisch zu den vorherigen Stufen.

Stufe 5 (Conv5_x) in Abb.: 2.2 ist die letzte Stufe des ResNet50 und besteht aus 3 Residualblöcken, die jeweils 2048 Merkmalskarten erzeugen. Anschließend erfolgt ein Global Average Pooling, gefolgt von einer Fully-Connected-Schicht für die Klassifikation.

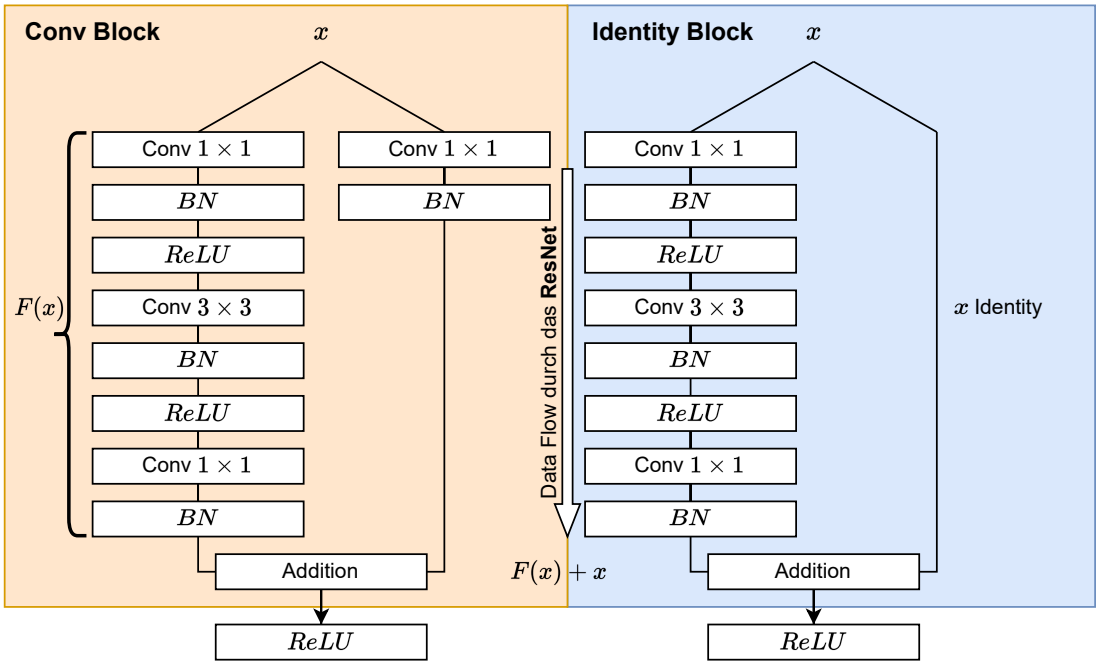

Abbildung 2.3: Residualblöcke: detailierte Operation innerhalb der *Conv-* und *Identity-Blöcke*

Residualblock Wie in Abb. 2.2 zu sehen, bestehen die Stufen 2 bis 5 aus 3, 4, 6 und 3 Residualblöcken. Ein Residualblock ist entweder ein `Conv-Block` (Convolutional

Block) oder ein `Identity-Block`. Diese werden als eine Sequenz von Schichten dazu eingesetzt, die Dimensionen der eingegebenen Merkmalskarten zu ändern. Innerhalb eines `Conv-Blocks` umfassen diese Änderungen sowohl die Erhöhung der Tiefe der Karten als auch die Reduzierung der räumlichen Dimension. Eine Stufe beginnt immer mit einem `Conv-Block`, welche eine initiale Downsampling Operation ausführen. Eine Ausnahme bildet hier die Stufe 1, da sie nur eine einfache Convolutional Schicht enthält. Innerhalb der Stufen 2 bis 5 in einem ResNet50 führt der erste `Conv-Block` folgende Operationen durch.

Innerhalb eines `Conv-Blocks`, wie in Abb.: 2.3 zu sehen ist, werden drei Schichten durchgeführt.

1. 1×1 Convolutional Schicht: diese Schicht reduziert die Tiefe der Merkmalskarten durch Anwendung einer 1×1 Faltung. Die Definition dieser Operation kann als $F_1(x) = W_1 * x + b_1$ beschrieben werden, wobei W_1 die Faltungskerne und b_1 der Bias-Vektor sind.

2. 3×3 Convolutional Schicht: diese Schicht führt ein Down-Sampling durch, indem es eine 3×3 Faltung mit einer Schrittweite (stride) von 2 ausführt. Das Ergebnis der Operation $F_2(F_1(x)) = W_2 * F_1(x) + b_2$ ist damit eine um die Hälfte reduzierte Merkmalskarte, im Vergleich zu $F_1(x)$.

3. 1×1 Convolutional Schicht: diese Schicht hat zur Aufgabe, durch die Anwendung einer 1×1 Faltung die Tiefe der Merkmalskarten zu erhöhen. Sie ähnelt der ersten Schicht. Beide müssen entsprechend den Anforderungen der nächsten Schicht die Anzahl erhöhen. Am Ende der 3 Schichten ergibt sich eine Operation $F_3(F_2(F_1(x))) = W_3 * F_2(F_1(x)) + b_3$.

Nachfolgend einer jeden Schicht wird zusätzlich eine ReLU-Aktivierungsfunktion sowie eine Batch-Normalisierung ($BN(x)$) durchgeführt. Diese Operationen unterstützten die Nichtlinearität innerhalb des Netzwerks und tragen zur Trainingsstabilität bei.

Die andere Art von Residualblock ist der `Identity-Blocks`, wie in Abb. 2.3 zu sehen. Diese Blöcke dienen innerhalb des ResNet50 dazu, die Merkmalskarten durch die verschiedenen Netzwerkschichten weiterzugeben, ohne die Dimensionalität der Merkmalskarten zu verändern. Sie enthalten einen Schlüsselaspekt der ResNet Architektur, den *Shortcut* (oder auch Skip-Connection). Die Verarbeitung innerhalb der `Identity-Blocks`

entspricht den drei Schritten des `Conv-Blocks`, mit dem Unterschied, dass sie die Dimentionalität der Merkmalskarten nicht verändert. Es wird also wieder eine 1×1 Faltung gefolgt von einer 3×3 Faltung gefolgt von einer 1×1 Faltung durchgeführt. Auch hier werden am Ende jeder Schicht sowohl σ als auch $BN(x)$ durchgeführt. Die Ausgabe dieser Sequenz von Transformatioenen wird abschließend dem unveränderten Eingang x (siehe Abb.: 2.3) hinzugefügt.

Durch die Addition des Eingangs x zu der Transformation innerhalb der Residualblöcke wird das Problem des verschwindenden und des explodierenden Gradienten minimiert, da sichergestellt wird, dass x vorwärts und rückwärts durch das Netz fließen kann. Zusätzlich muss bei einem regulären Block (in anderen tiefen Netzwerkarchitekturen) die optimale Funktion zur Vorhersage $H(x)$ direkt gelernt werden. Diese ist in aller Regel komplex. In einem Residualblock muss nur die Residualfunktion $F(x) = H(x) - x$ gelernt werden, wodurch die Identitätszuordnung x leichter zu lernen ist. Dies bedeutet, dass das Netzwerk nicht die gesamte Transformation lernen muss, sondern nur die Differenz (das Residuum) zwischen der Eingabe und der Ausgabe. Die endgültige Ausgabe des Layers ist dann $H(x) = F(x) + x$, wobei x die Eingabe ist, die über die Shortcut Connection hinzugefügt wird. Jeder Block sucht also eine Transformation, sodass $F(x) + x$ so nah wie möglich an $H(x)$ herankommt. Innerhalb des Blocks trägt jede Schicht zum Lernen dieser Transformation bei, jedoch lernt keine Schicht die Residualfunktion $F(x)$ unabhängig. Stattdessen ist es die kumulative Wirkung aller Schichten im Block, die $F(x)$ erlernt[19].

Sollte sich die Dimension der Merkmalskarten zwischen den Eingängen und den Ausgängen der unterschiedlichen Stufen verändert haben (siehe Abb. 2.2.2 zwischen $C_1, ..., C_5$), sei es durch Veränderung der Tiefe der Karten oder der Veränderung der räumlichen Auflösung (Höhe und/oder Breite), dann ist eine direkte Addition der Eingabe x nicht mehr möglich. In einem solchen Fall werden innerhalb des ResNet50 die Dimensionen vorab angepasst. Die Anpassung erfolgt durch *Projektions Shortcuts* (oder Projektions-Verbindungen).

Ein *Projektions Shortcut* verwendet eine 1×1 Faltung mit der entsprechenden Schrittweite von 2, um die Dimensionalität der eingehenden Merkmalskarten so zu verändern, dass diese zu den Ausgangsmerkmalskarten des jeweiligen `Identity-Blocks` passen. Nach Abschluss der Anwendung des *Projektions Shortcuts* ist es dem Netz möglich, die Eingabemerkmalskarten und die Ausgabemerkmalskarten des *Identity Blocks* zu addieren, wodurch eine Integration der Merkmale trotz veränderter Dimension ermöglicht wird.

Diese Shortcuts werden innerhalb der Residualnetze an Übergängen der unterschiedlichen Stufen durchgeführt, da dort durch Down-Sampling die Auflösung der Merkmalskarten verringert und die Tiefe der Karten erhöht wird. Durch diese Operationen werden von Stufe zu Stufe immer komplexere Merkmale von dem Netz erfasst. Diese Anpassung durch *Projektions Shortcuts* ermöglicht es dem Netzwerk, die Vorteile der residualen Architektur zu nutzen, auch wenn die Dimensionalität der Daten durch das Netzwerk verändert wird.

Die Verwendung der Residualblöcke innerhalb des ResNet führt während der Forward- und Backwardpropagation zu einer gleichmäßigeren Anpassung der Gewichte in allen Schichten (im Speziellen den tiefen Schichten) der Architektur. Dieser Ansatz unterscheidet sich von dem Herangehen traditioneller tiefer Netze, die die Zielfunktion $H(x)$ direkt zu lernen versuchen.

Das beschriebene Durchlaufen des ResNet50 entspricht damit dem unten dargestellten Aufbau, wobei Stage 2 bis Stage 5 an das Feature Payramid Network in lateralen Verbindungen weitergegeben werden.

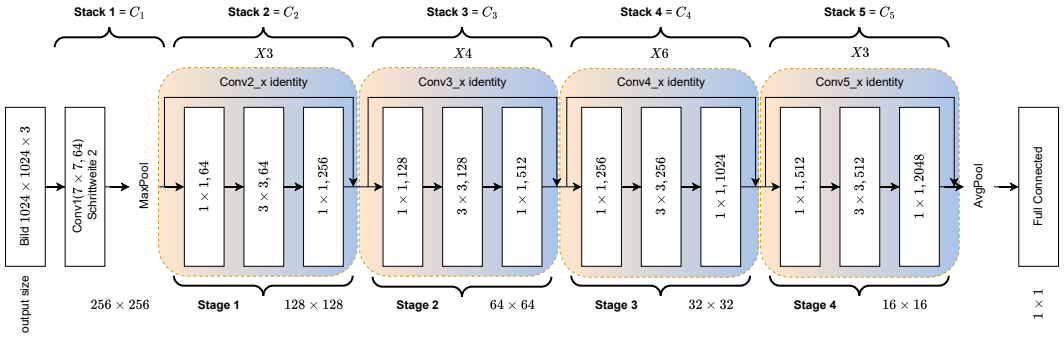

Abbildung 2.4: Detailliertere Darstellung eines ResNet50

2.2.3 Feature-Pyramid-Netz

Das Feature-Pyramid-Netz[31] (FPN) ist eine Architektur zur Extraktion von Merkmalen innerhalb von Bildern. Diese können beliebiger Größe oder Auflösung sein. Der Nutzen der Architektur innerhalb des RetinaNet ist es, die Erkennung der Merkmale unterschiedlicher Größen auf einem Eingabebild zu verbessern. Jede Stufe des FPN erhält sowohl Informationen aus der finalen Ausgabe des ResNet50 Backbone als auch Informationen aus Zwischenebenen des Netzes (dies geschieht über die lateralen Verbindungen). Jede

Stufe des Netzes ist damit für die Erkennung von Objekten unterschiedlicher Maßstäbe verantwortlich.

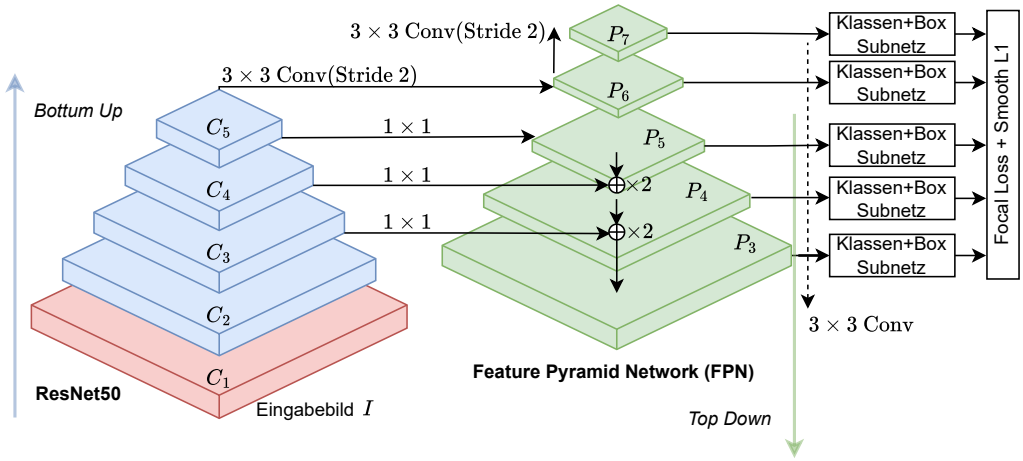

Abbildung 2.5: Die schematische Darstellung eines Feature Pyramid Network (FPN) in grün[31]

Der Vorteil der Nutzung des FPN ist es, einem tiefen neuronalen Netz (wie z.B. dem ResNet50) die Fähigkeit zu geben, Merkmale unterschiedlicher Größen besser zu erlernen und nicht nur die finale Ausgabe des Netzes zu nutzen. Wie in Abb. 2.5 zu sehen, übergibt das Backbone-Netz auf unterschiedlichen Stages Informationen an das FPN. Auf der linken Seite des Diagramms ist der *Bottom-up pathway* zu sehen, der dem in Abschn. 2.2.2 beschriebenen Netz entspricht. Das Eingabebild wird durch sequenziell aufeinanderfolgende Stufen $C_1, C_2, ..., C_5$ geleitet und lernt so auf den Schichten immer gröbere Merkmale zu extrahieren, wobei durch die steigende Anzahl der Merkmalskarten die semantische Reichhaltigkeit erhöht wird. Das FPN nutzt nun den *Top-down pathway*, um mithilfe der Merkmalskarten der vorangegangenen Stufe die räumliche Auflösung wieder zu erhöhen. Das FPN beginnt also mit dem Ergebnis des Bottum-up Pfades C_5. Durch Upsampling wird die Auflösung der Merkmale erhöht und durch Anwendung einer 1×1 Faltung mit dem Ergebnis von C_4 addiert. Diese Operationen werden für alle Schichten bis einschließlich zu C_3 wiederholt. C_1 und C_2 werden in der RetinaNet-Architektur nicht an das FPN übergeben, da die Ergebnisse dieser Stufen zu wenig semantische Informationen enthalten. Das Resultat dieser Operationen sind wieder die Merkmalskarten $(P_3, ..., P_7)$, welche zusätzlich zu den ursprünglichen Merkmalskarten auch die Informationen der vorangegangenen Schicht enthalten. Auf diese Weise sind sie semantisch reichhaltiger.

Innerhalb des RetinaNet wird das Backbone verwendet, um die finalen Merkmalskarten zu generieren. Für jede Stage i im ResNet wird eine Ausgabe C_i erzeugt, die dann als Input für das FPN dient. Das FPN nutzt diese, um die Merkmalskarten P_3, P_4, P_5 zu erzeugen. Für die Schichten P_6 und P_7 werden spezielle Faltungen mit Schrittweite 2 angewendet, um die Merkmalskarten für die Erkennung großer Objekte zu generieren.

1. Bottom-up Pathway (ResNet-Backbone):
   - Jede Stufe i im ResNet-Backbone erzeugt eine Ausgabe C_i, die Merkmalskarten auf einer bestimmten Auflösungsebene darstellt.
   - Diese Merkmalskarten C_i sind das Ergebnis der kumulativen Convolutional, Batch-Normalization und Aktivierungsfunktionen, die auf die Eingabedaten angewendet werden.
2. Top-down Pathway und Laterale Verbindungen:
   - Im Top-down Pathway des FPN wird mit den tiefsten Merkmalskarten C_5 begonnen. Diese haben die niedrigste Auflösung 7×7, bieten jedoch die komplexesten semantischen Informationen.
   - Für jede Stufe i im Top-down Pathway werden die Merkmalskarten C_i aus dem Bottom-up Pathway über eine laterale Verbindung integriert:
     - Für jede Stufe i im ResNet-Backbone:
       C_i = Output der Stufe i
     - Für jede Stufe i im FPN:

$$P_i = \begin{cases} Up(P_{i+1}) + L_i(C_i) & : 3 \leq i \leq 5 \\ 3 \times 3 \text{ Conv Stride } 2(C_5) & : i = 6 \\ \text{ReLU und } 3 \times 3 \text{ Conv Stride } 2(P_6)) & : i = 7 \end{cases}$$

       wobei $Li(C_i)$ die laterale Projektion (1×1-Convolution) der Merkmalskarte C_i ist.
   - Die upgesamplete Merkmalskarte der nächsthöheren Ebene im Top-down Pathway wird mit lateral angepassten Merkmalskarten kombiniert:
     - Sei P_{i+1} die upgesamplete Merkmalskarten der nächsthöheren Ebene, dann ist die Eingabe für die aktuelle Ebene $P_i = \text{Up}(P_{i+1}) + L_i(C_i)$.

3. Output:

    - Jede Ebene i im Top-down Pathway des FPN erzeugt eine Output-Merkmalskarte P_i, die eine Kombination aus der upgesampleten Merkmalskarte und der lateral angepassten Merkmalskarte des ResNet50-Backbones ist.
    - Diese Output-Merkmalskarten P_i enthalten somit semantische Informationen aus den tieferen Schichten des ResNet50 als auch detaillierte Informationen aus den höheren Schichten.

Jede der so erstellten Merkmalskarten $(P_3, P_4, ..., P_7)$ wird weiter durch eine 3×3 Faltung weiterverarbeitet, um die finalen Merkmalskarten des FPN zu erzeugen. Diese werden anschließend in die Subnetze gegeben. Auch wenn alle Merkmalskarten unterschiedliche räumliche Ausdehnungen besitzen, haben sie alle 256 Kanäle.

Ankerboxen

Das RetinaNet nutzt Ankerboxen, um die potenziellen Objekte innerhalb eines Eingabebildes zu lokalisieren. Diese werden im RetinaNet für die Merkmalskarten des FPN P_i $(P_3, P_4, ..., P_7)$ in unterschiedlichen, jedoch festen Skalierungsfaktoren und Aspektverhältnissen definiert. Bei Ankerboxen handelt es sich um Bounding-Boxen, welche den Suchraum der unterschiedlichen Objekte innerhalb des betrachteten Bildes aufteilen[30, pp. 230-236]:

- Die Basisgröße der Ankerboxen skaliert von 32^2 Pixeln auf P_3 bis zu 512^2 Pixeln auf P_7. Diese wurde gewählt, da P_3 die höchste Auflösung (kleinste Objekte) potenziell beinhaltet und P_7 die niedrigste Auflösung (große Objekte) potenziell beinhaltet.
- Für jede Basisgröße gibt es Ankerboxen mit drei Skalierungsfaktoren $\{1, 1.3, 1.6\}$ und drei Aspektverhältnissen $\{1/2, 1, 2\}$.
- Insgesamt verwendet RetinaNet neun verschiedene Ankerarten pro Merkmalskarte.

Die beschriebene Basisgröße der Ankerboxen wird mithilfe der Ground-Truth Daten (Bounding-Boxen) und mit Intersection of Union (IOU) zugewiesen. Hat eine Ankerbox keinen oder einen sehr geringen IOU-Wert, so wird das Netz die Ankerbox als Hintergrund interpretieren. Die Anzahl der Ankerboxen für ein Eingabebild kann anhand der Größe der Merkmalskarte im FPN bestimmt werden. Hierzu folgendes Beispiel einer Berechnung der Ankerboxen.

Hat ein Eingabebild eine Auflösung von 1024×1024, dann ergeben sich folgende unterschiedliche Merkmalskarten pro Ebene innerhalb des FPN:

- Merkmalskarten (P3): 64×64 Positionen
- Merkmalskarten (P4): 32×32 Positionen
- Merkmalskarten (P5): 16×16 Positionen
- Merkmalskarten (P6): 8×8 Positionen
- Merkmalskarten (P7): 4×4 Positionen

Die Gesamtzahl der Positionen der Ankerboxen kann wie folgt berechnet werden: $64 \times 64 + 32 \times 32 + 16 \times 16 + 8 \times 8 + 4 \times 4 = 5,456$ Positionen. Da für jede Position 9 Ankerboxen berechnet werden, ergibt sich eine Gesamtzahl von Ankerboxen für das gesamte Bild $5,456 \times 9$, was ca. 50.000 Ankerboxen entspricht. Ankerboxen werden durch die Zentrumskoordinaten (x_a, y_a), Breite w_a und Höhe h_a definiert. Außerdem gibt es für jede Ankerbox einen Vektor von Klassenwahrscheinlichkeiten sowie einen Vektor für die Bounding-Box-Offset Vorhersage.

- Sei k die Anzahl der Klassen, dann liefert das Klassifikations-Subnetz (siehe Abschn. 2.2.4) für jede Ankerbox einen Wahrscheinlichkeitsvektor $p = (p_1, p_2, ..., p_k)$, wobei jede Komponente p_i die Wahrscheinlichkeit angibt, dass die Ankerbox zur Klasse i gehört.
- Das Box-Regressionsnetz (siehe Abschn. 2.2.5) liefert einen Offset-Vektor $o = (\Delta x, \Delta y, \Delta w, \Delta h)$ für die Anpassung der Ankerboxen an die tatsächlichen Bounding-Boxen. o wird genutzt, um die finale Position und die Größe der Bounding-Box aus den Ankerbox-Parametern x_a, y_a, w_a, h_a zu berechnen.
- Die finale Vorhersage für eine Ankerbox auf Ebene P_i wird als Kombination der Klassenwahrscheinlichkeiten und Bounding-Box-Offsets berechnet: (p, o).

Die Anpassung der Ankerboxen an die tatsächlichen Bounding-Boxen wird wie folgt dargestellt:

$$x_{final} = x_a + w_a \cdot \Delta x \tag{2.1}$$

$$y_{final} = y_a + h_a \cdot \Delta y \tag{2.2}$$

$$w_{final} = w_a \cdot \exp(\Delta w) \tag{2.3}$$

$$h_{final} = h_a \cdot \exp(\Delta h) \tag{2.4}$$

Dabei sind $x_{final}, y_{final}, w_{final}, h_{final}$ die Koordinaten und Größe der finalen Bounding-Box nach der Anpassung durch das Box-Regressionsnetz. Durch die Nutzung von Ankerboxen in dem RetinaNet kann diese unterschiedlichen Objekte unterschiedlicher Größe auf einem Eingabebild erkennen. Hierbei erkennen kleinere Ankerboxen auf den höheren Schichten (z.B. P_3) kleinere Objekte auf dem Bild und größere Ankerboxen auf den niedrigeren Ebenen (z.B. P_7) die größeren Objekte. Durch diese Architektur ist eine Lokalisierung unterschiedlicher Objekte von unterschiedlicher Größe auf einem Eingabebild möglich. Die verwendeten Klassifikations- sowie Box-Regression-Subnetze liefern Klassenvorhersagen und Bounding-Box-Deltas.

- Der Klassifikationskopf prognostiziert $A \times K$ Wahrscheinlichkeiten, wobei A die Anzahl der Ankerboxtypen (9 in RetinaNet) und K die Anzahl der Klassen sind.
- Der Boxdetektionskopf prognostiziert $A \times 4$ Box-Deltas ($\Delta x, \Delta y, \Delta w, \Delta h$).

Die Ankerboxen sind regelmäßig über das Eingabebild verteilt und werden wie beschrieben an jede Ebene des FPN angepasst. Die Verlustberechnung in RetinaNet berücksichtigt zudem die Abweichungen zwischen den Ankerboxen und den tatsächlichen Bounding-Boxen.

2.2.4 Klassifikations-Subnetz

Das Klassifikations-Subnetz erstellt für jede Ankerbox aller Schichten des FPN eine Vorhersage der Wahrscheinlichkeit des Vorhandenseins aller zu erlernenden Klassen. Es besteht aus vier 3×3 Convolutional Layern mit einer ReLU Aktivierung sowie zusätzlicher Batchnormalisierung (aus vergleichbaren Gründen wie auch im ResNet50, siehe Abschn. 2.2.2). Anschließend folgt eine fünfte Schicht, die ebenfalls ein 3×3 Convolutional Layer ist und mit einer Sigmoidaktivierung arbeitet. Diese wird verwendet, um

für jede Ankerbox und jede Klasse eine binäre Wahrscheinlichkeit vorherzusagen. Durch diese wird angegeben, ob ein Objekt in der jeweiligen Klasse innerhalb der jeweiligen Ankerbox vorhanden ist.

2.2.5 Box-Regressions-Subnetz

Das Box-Regressionsnetz in RetinaNet funktioniert analog zum Klassifikations-Subnetz, zielt jedoch darauf ab, die Offsets $\Delta x, \Delta y, \Delta w, \Delta h$ für jede Ankerbox zu berechnen. Diese Offsets dienen dazu, die Ankerboxen an die Ground-Truth Bounding Boxen anzupassen. Der Aufbau folgt dem des Klassifikations-Subnetzes. Der letzte Layer liefert jedoch lineare Ausgaben für die Box-Koordinaten. Hierbei handelt es sich um 4 Δ Werte pro Ankerbox, die für die Anpassung von Zentrum x und y, sowie für Breite w und der Höhe w.

Post-Processing

Nachdem das Klassifikations- und das Box-Regressionsnetz ihre Vorhersagen gemacht haben, folgt ein Post-Processing-Schritt. Dieser verarbeitet die Klassifikationsscores S_{cls} und Regressionsparameter R_{reg} und wendet die Non Maximum Suppression[21] (NMS) Operation an. Das Ergebnis des Postprocessing ist die Detektionen D mit Klassenlabels und Bounding-Box-Koordinaten.

Verlustfunktion

Die verwendete Verlustfunktion des RetinaNet ist eine Kombination aus dem `Focal Loss` und dem `Smooth L1 Loss`. Diese Kombination setzt sich aus der Notwendigkeit zusammen, sowohl ein Klassifikationsproblem als auch ein Regressionsproblem zu lösen. Hierbei nutzt das Klassifikations-Subnetze den Focal Loss und die Box-Regressions-Subnetze den Smooth L1 Loss.

Focal Loss für die Klassifikation

Der *Focal Loss* wird auf die Ausgaben des Klassifikations-Subnetzes angewendet. Die Verlustfunktion wird verwendet, da es innerhalb eines Bildes ein hohes Ungleichgewicht von

Vordergrund- und Hintergrundinformationen gibt. Die Mehrheit der auf einem Bild befindlichen Objekte sind Hintergrundinformationen, der Fokal-Loss adressiert dieses Problem wie folgt[43]:

$$\text{Focal Loss} = -\alpha(1 - p_t)^{\gamma} \log(p_t)$$

wobei:

- p_t die Wahrscheinlichkeit der richtigen Klassifikation für jede Klasse und Ankerbox ist. Diese wird von der Sigmoid-Aktivierungsfunktion im Klassifikations-Subnetz zurückgegeben.
- α ein Gewichtungsfaktor ist, um den Einfluss von schwer zu klassifizierenden Beispielen anzupassen. So können Vordergrundinformationen stärker gewichtet werden und das Training unterstützt.
- γ ein Faktor, der den Einfluss einfacher Beispiele reduziert.

Smooth L1 Loss für die Box-Regression

Für die Box-Regressions-Subnetze wird innerhalb des RetinaNet der *Smooth L1 Loss* verwendet, um die Abweichung zwischen den vorhergesagten Bounding-Boxen und den tatsächlichen Ground-Truth-Bounding-Boxen zu berechnen. Der Smooth L1 Loss ist definiert als[12]:

$$\text{Smooth L1 Loss} = \begin{cases} 0.5(\Delta)^2 & \text{wenn } |\Delta| < 1 \\ |\Delta| - 0.5 & \text{wenn } |\Delta| \geq 1 \end{cases}$$

wobei Δ die Differenz zwischen der vorhergesagten Bounding-Box (bestehend aus den Offset-Werten $\Delta x, \Delta y, \Delta w, \Delta h$) und der Ground-Truth-Bounding-Box ist. Diese Differenz wird für alle 4 Werte berechnet, die Zentrumskoordinaten x, y, die Breite w, und die Höhe h

Kombinierte Loss-Funktion

Da das Klassifikations-Subnetz (mit Focal Loss) und das Box-Regression-Subnetz (mit Smooth L1 Loss) gleichzeitig trainiert werden, ergibt sich die finale Loss-Funktion:

$$\text{Loss} = \text{Focal Loss} + \lambda \times \text{Smooth L1 Loss}$$

wobei:

- **Focal Loss**: Angiebt, wie gut oder schlecht das Modell Objekte in den Ankerboxen klassifiziert.
- **Smooth L1 Loss**: Angiebt, wie gut oder schlecht das Modell die Position und Größe der Bounding-Boxen an die Ground-Truth-Daten anpasst.
- λ ist ein zusätzlicher Faktor zur Gewichtung. Dieser wird üblicherweise auf $\lambda = 1$ gesetzt, um eine Gewichtung zwischen Klassifikations- und Regressions-Loss zu gewährleisten, die ausgewogener ist.

Beide Subnetze werden gleichzeitig trainiert, indem die Gradienten der kombinierten Loss-Funktion durch das gesamte Netzwerk zurück propagiert werden. So werden sowohl das Klassifikations- als auch das Box-Regression-Subnetz gleichzeitig optimiert.

2.3 SAM (Foundation-Modell)

Das Segment Anything Model (SAM)[27] ist ein Modell, welches für die semantische Segmentierung von Bildern entwickelt worden ist. Es basiert grundlegend auf einer Transformer-Architektur. Bei der Veröffentlichung wurde das Modell als ein Foundationmodel (Basismodell) für Bildsegmentierungsaufgaben vorgestellt. Es ist in der Lage, unterschiedlichste Segmente auf Bildern zu erkennen. Es besitzt weder Klassen (Label) noch versteht es Einzelinstanzen. Die Aufgabe des Modells besteht darin, einzelne Pixel zu Gruppen (Segmenten) zuzuordnen. Durch ein iteratives Verfahren zur Verbesserung der Genauigkeit der Segmente erlaubt das Modell, über weitere Prompts des Nutzers die getätigten Segmentierungen zu spezifizieren (dies führt zu einer guten Zero-Shot-Performance über eine Vielzahl von Bildverteilungen). Die Hauptkomponenten sind der Image Encoder,

der Prompt Encoder und der Mask Decoder sowie die außergewöhnlich umfangreiche Datengrundlage. Alle Komponenten werden nachfolgend erläutert.

Datengrundlage und Training Der Trainingsdatensatz für das Segment Anything Model (SAM) besteht aus dem SA-1B-Datensatz. Dieser Datensatz wurde speziell für die Erstellung des SAM hergestellt und besteht aus über 1 Milliarde Masken, die aus 11 Millionen hochauflösenden, lizenzierten und datenschutzfreundlichen Bildern gewonnen wurden [27]. Die Daten beinhalten verschiedenste Bilder, um die Fähigkeit der Generalisierung zu erhöhen. Die Trainingsbilder umfassen dabei keine spezielle Domäne, Themengebiete o. Ä.. Um den Datensatz zu erstellen, haben die Autoren einen dreistufigen Daten-Engine-Prozess implementiert:

1. „assisted-manual stage“: Annotatoren haben mithilfe eines Webinterfaces Masken erstellt, die SAM anschließend versuchte zu verbessern. Hierbei haben die Annotatoren durch die Eingabe von neuen Prompts das Segmentierungsergebnis verbessert. Das Ergebnis der ersten Phase waren ca. 20-44 Masken pro Bild und eine Gesamtmenge von 4,3 Millionen manuell segmentierten Masken.

2. „semi-automatic stage“: SAM generierte in dieser Phase automatisch Masken, die dann durch den Abgleichen mit den manuell validierten Masken (von Annotatoren) der ersten Phase verglichen wurden. Zusätzlich wurden noch fehlende Masken auf den Bildern von den Annotatoren ergänzt, um die Vollständigkeit der Segmentierungen zu verbessern. Das Ergebnis dieser Phase betrug 5,9 Millionen zusätzliche Masken und eine Gesamtmenge an Masken von 10,2 Millionen.

3. „fully automatic stage“: Die abschließende Phase des Trainingsprozesses wurde ohne Hilfe von manueller Korrektur oder Validierung gemacht. Hierfür hat SAM die Trainingsbilder automatisch segmentiert und so weitere Trainingsdaten geschaffen. Ergebnis der Phase waren ca. 100 Masken pro Trainingsbild und eine Gesamtmenge von ca. 1,1 Milliarden Masken innerhalb des SA-1B-Datensatzes.

Der 3-stufige Prozess, der von einer manuellen über eine halb-automatische bis hin zu voll automatischen Segmenterkennung führt, hat zu einer robusten Datenlage unterschiedlichster Objekte und einer damit verbundenen guten Maskenvorhersagefähigkeit beigetragen. Für die Optimierung der Segmentierungen innerhalb des Trainingsprozesses wurde eine Kombination aus Focal Loss und Dice Loss gewählt.

2.3.1 Architektur

Die Architektur des SAM besteht aus drei Hauptkomponenten: dem **Image Encoder**, dem **Prompt Encoder** und dem **Mask Decoder**. Der Workflow der Architektur basiert darauf, dass das zu segmentierende Bild initial durch den Image Encoder verarbeitet wird, um ein latentes Bild-Embedding zu erzeugen. Dieses Image Embedding wird dann zusammen mit den Prompts (Punkten, Boxen, Masken oder Texten) durch den Prompt Encoder in den Mask Decoder gegeben, um die Informationen für die Maskenerstellung zu verarbeiten. Der Mask Decoder nutzt die Informationen aus beiden Encodern, um die entsprechenden Segmentierungsmasken zu erstellen.

Image Encoder: Der Image Encoder (siehe Abb. 2.6 (grün)) basiert auf einem *Vision Transformer (ViT)*-Modell (im Speziellen dem Mask Auto Encoder[16] (MAE)). Durch das Verarbeiten des Bildes in Form von hochdimensionalen Embeddings erstellt SAM eine latente Darstellung des Eingabebildes, welches anschließend weiterverarbeitet wird.

Prompt Encoder: Der Prompt Encoder (siehe Abb. 2.6 (lila)) verarbeitet die unterschiedlichen Eingaben des Nutzers, die sowohl als **sparse promts** als auch als *dense prompts* dem Modell gegeben werden können. Eine Erweiterung für diese Arbeit, um die Instanzsegmentierung zu realisieren, ist das *Contrastive Language-Image Pre-Training[41] (CLIP)* Modell. Es wird genutzt, um Texteingaben des Nutzers in ein prompt embedding zu überführen, welches Text- mit Bildinformationen assoziiert.

Mask Decoder: Der Mask Decoder (siehe Abb. 2.6 (orange)) erhält die unterschiedlichen Embeddings des Images (Image Encoder), der Eingabeprompts (Prompt Encoder) und des Positional Embeddings um aus diesen Informationen 4 Segmentierungsmasken zu erstellen. Die unterschiedlichen Embeddings werden kombiniert und miteinander assoziiert. Nach Abschluss der Operationen innerhalb der Komponente werden 4 Masken berechnet und dem Nutzer wieder gespiegelt.

Die Abbildung 2.6 zeigt die Hauptkomponenten der Architektur. Auf der linken Seite wird ein Satellitenbild durch den Image Encoder (grün) verarbeitet. Anschließend werden Eingabeprompts durch den Prompt Encoder (lila) kodiert und die Masken durch den Mask Decoder (orange) erstellt, welche die zu segmentierenden Objekte im Bild hervorhebt.

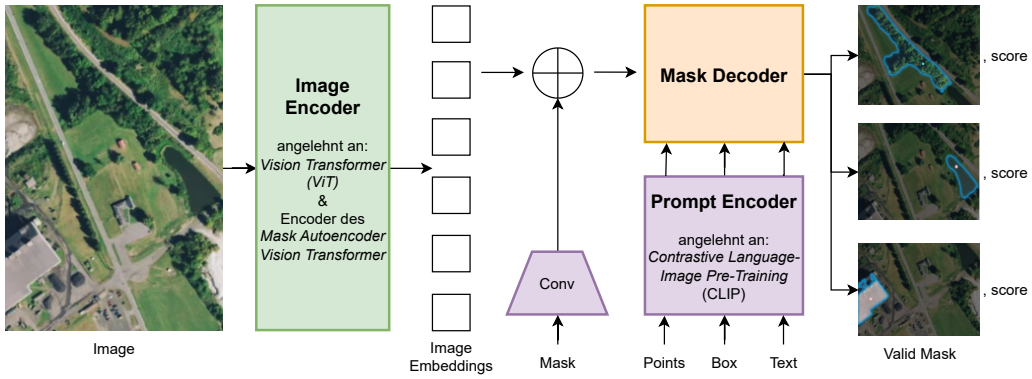

Abbildung 2.6: Segment Anything Model (SAM) Architekturübersicht (angelehnt an[27])

2.3.2 Image Encoder

Der Image Encoder (wie in Abb. 2.6 zu sehen) führt eine Reihe von Operationen aus, die zum Ziel haben, das Eingabebild I in die vom Mask-Decoder benötigte Vektorrepräsentation zu überführen und I um weitere Informationen anzureichern.

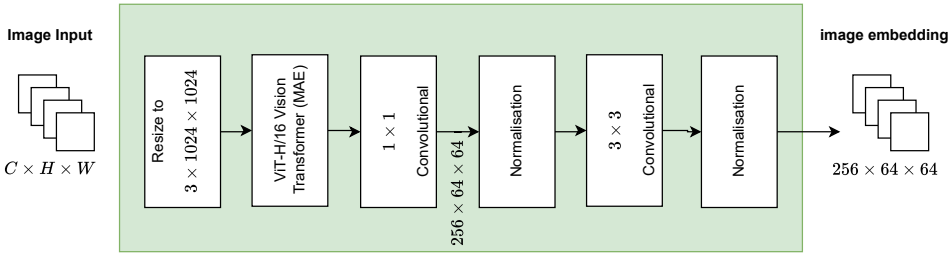

Abbildung 2.7: Der Image Encoder des Segment Anything Model (SAM)

Wie in Abbildung 2.7 zusehen ist, wird $I^{C \times H \times W}$ in den Image Encoder hineingegeben. Dieser verändert die Abmessungen auf $I^{3 \times 1024 \times 1024}$ und gibt I in den Encoder einer speziellen Version eines Vision-Transformer (ViT)[10]. Die Architektur ist darauf ausgelegt, hochauflösende Bilder ($3 \times 1024 \times 1024$) zu verarbeiten und eine latente Repräsentation der Bildinhalte zu erzeugen. Im SAM wird der Encoder Teil des Masked Autoencoder (MAE)[17](Abb. 2.8) verwendet. Es handelt sich dabei um einen vortrainierten Encoder ohne den zugehörigen Decoder des MAE.

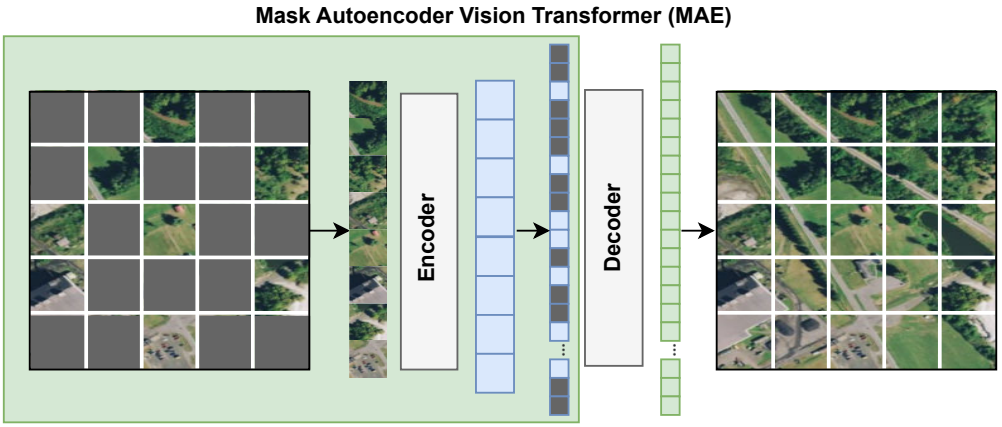

Abbildung 2.8: MAE-Strategie zur Repräsentationsgewinnung im SAM (nur Encoder-Teil)

Das Vorgehen des MAE lässt sich wie folgt beschreiben[17]:

1. **Eingabe des Bildes** I: Das Eingabebild I wird in N nicht überlappende Patches der Größe $P \times P$ unterteilt. Jeder Patch wird als Vektor $\mathbf{x}_i \in \mathbb{R}^{P \times P \times C}$ dargestellt, wobei C die Anzahl der Kanäle ist (3 für RGB-Bilder).

2. **Patch-Embedding**: Jeder Patch $\mathbf{x}_i$ wird durch eine lineare Projektion in eine latente Darstellung $\mathbf{z}_i$ überführt:

$$\mathbf{z}_i = E(\mathbf{x}_i) = \mathbf{W}_e \cdot \text{flatten}(\mathbf{x}_i) + \mathbf{b}_e,$$

   wobei E der Patch-Embedding-Operator ist, $\mathbf{W}_e \in \mathbb{R}^{D \times (P^2 C)}$ die Gewichtungsmatrix und $\mathbf{b}_e \in \mathbb{R}^D$ der Bias-Vektor.

3. **Maskierung**: Ein zufälliger Anteil γ der Patches wird maskiert. Die Maske M ist definiert als:

$$M(i) = \begin{cases} 0, & \text{wenn Patch } i \text{ maskiert ist,} \\ 1, & \text{sonst.} \end{cases}$$

   Die Menge der unmaskierten Patches ist $U = \{i \mid M(i) = 1\}$.

4. **Encoder-Verarbeitung**: Die unmaskierten Patch-Embeddings werden durch den Encoder f_ψ verarbeitet:

$$\mathbf{h} = f_\psi\left(\{\mathbf{z}_i + \mathbf{p}_i \mid i \in U\}\right),$$

   wobei $\mathbf{p}_i$ das Positions-Embedding des i-ten Patches ist.

5. **Decoder-Verarbeitung**: Der Decoder g_ϕ erhält die latente Repräsentation $\mathbf{h}$ und versucht, die ursprünglichen Patches zu rekonstruieren:

$$\hat{\mathbf{x}}_i = g_\phi(\mathbf{h}, \mathbf{p}_i), \quad \forall i \in M,$$

   wobei $M = \{i \mid M(i) = 0\}$ die Menge der maskierten Patches ist.

6. **Berechnung des Verlustes**: Der Rekonstruktionsverlust wird mittels Mean Squared Error berechnet:

$$L = \frac{1}{|M|} \sum_{i \in M} \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|^2 .$$

7. **Backpropagation und Parameteraktualisierung**: Die Parameter ψ (Encoder) und ϕ (Decoder) werden durch Minimierung des Verlustes L aktualisiert:

$$\psi \leftarrow \psi - \eta \frac{\partial L}{\partial \psi}, \quad \phi \leftarrow \phi - \eta \frac{\partial L}{\partial \phi},$$

   wobei η die Lernrate ist.

8. **Wiederholung des Trainingsprozesses**: Die Schritte 1 bis 7 werden über mehrere Epochen wiederholt, bis der Rekonstruktionsverlust konvergiert.

9. **Verwendung des vortrainierten Encoders**: Nach dem Training wird der Encoder f_ψ als vortrainiertes Modell für nachfolgende Aufgaben verwendet.

Die Ausgabe des MAE wird durch eine 1×1-Convolutional Schicht geleitet, um die latente Repräsentation weiter zu verdichten. Dabei wird die drei kanalige Eingabe (RGB) auf eine größere Anzahl von Merkmalskarten (256) erweitert. Außerdem wird die räumliche Dimension verringert, um die Berechenbarkeit zu erleichtern und das erforderliche Format für den Mask-Decoder des MAE zu erhalten.

Die Merkmalskarten (256×64×64) werden normalisiert und durch eine 3×3-Convolutional-Schicht mit Normalisierung geleitet. Als Ergebnis der Verarbeitungsschritte entsteht das

im Mask-Decoder verwendete ***image embedding***. Wichtig hierbei ist, dass im Rahmen des SAM nur der Encoder-Teil des MAE im Image-Encoder des SAM genutzt wird und das beschriebene Training schon vorab passierte.

2.3.3 Prompt-Encoder

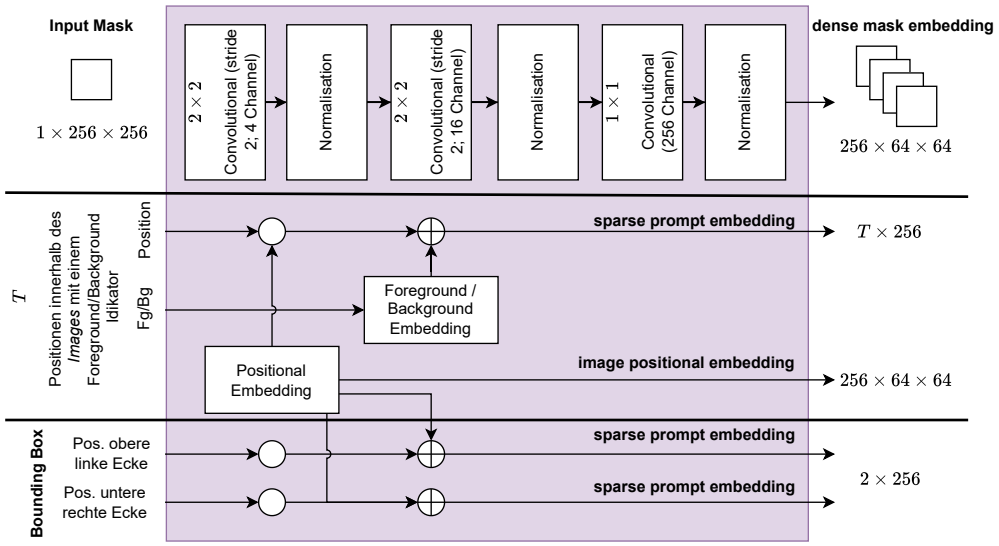

Abbildung 2.9: Detailierte Darstellung der Funktionsweise des Prompt Encoder in SAM

Der Prompt-Encoder im Segment Anything Modell (SAM) verarbeitet verschiedene Arten von Eingabe-Prompts, wie in Abbildung 2.9 zu sehen, um valide Masken vorherzusagen. Es gibt folgende Prompt-Typen:

- **Punkte**: Markierungen, die als Vordergrund oder Hintergrund gekennzeichnet sind. In Abbildung 2.9 in der mittleren Bahn zu sehen.
- **Boxen**: Rechteckige Boxen, innerhalb derer die Masken vorhergesagt werden sollen (einer Bounding-Box). In Abbildung 2.9 in der unteren Bahn zu sehen.
- **Masken**: Binäre Masken in der gleichen Größe des Bildes, die angeben, welche Pixel als Maske vorhergesagt werden sollen. In Abbildung 2.9 in der oberen Bahn zu sehen.

Zusätzlich gibt es einen weiteren Prompt-Typen, **Text**, dieser wird durch CLIP codiert (siehe Abschnitt+2.3.3). Die Text-Prompts sind für die Umsetzung der hier beschriebenen Arbeit essenziell, da sie die Möglichkeit eröffnen, eine Instanzsegmentierung zu implementieren.

Punkt-Prompts

Für Prompts des Typs *Liste von Punkten* wird für jeden der T angeklickten Punkte die Positionscodierung $\mathbf{p}_i$ mit einem vortrainierten Embedding $\mathbf{e}_{\text{FG/BG}}$ kombiniert, das *Vordergrund* oder *Hintergrund* repräsentiert:

$$\mathbf{z}_i = \mathbf{p}_i + \mathbf{e}_{\text{FG/BG}}, \quad \forall i = 1, \ldots, T.$$

Das **Sparse Embedding**, welches hieraus resultiert, hat die Größe $(T \times 256)$. Damit ergibt sich für jeden der T Punkte einen Embedding-Vektor der Länge 256.

Box-Prompts

Für Prompts des Typs *Bounding-Box* werden die Positionscodierungen der oberen linken Ecke $(x_{\text{tl}}, y_{\text{tl}})$ und der unteren rechten Ecke $(x_{\text{br}}, y_{\text{br}})$ jeweils mit einem gelernten Embedding kombiniert:

$$\mathbf{z}_{\text{tl}} = \mathbf{p}_{\text{tl}} + \mathbf{e}_{\text{tl}},$$
$$\mathbf{z}_{\text{br}} = \mathbf{p}_{\text{br}} + \mathbf{e}_{\text{br}},$$

wobei $\mathbf{e}_{\text{tl}}$ und $\mathbf{e}_{\text{br}}$ die Embeddings für die obere linke bzw. untere rechte Ecke sind. Das **Sparse Embedding**, welches hieraus resultiert, hat die Größe (2×256).

Masken-Prompts

Der Prompt vom Typ *Maske* ist eine binäre Matrix $\mathbf{M} \in \{0,1\}^{H \times W}$ in der räumlichen Ausdehnung des Bildes ($H = W = 1024$). Hierbei geben 1-Bit-Pixel die Bereiche des Bildes an, welche segmentiert werden sollen, sowie 0-Bit-Pixel die, welche nicht relevant sind.

Die Maske wird durch ein Convolutional Neural Network $\mathrm{CNN}_{\mathrm{mask}}$ in ein **Dense Prompt Embedding** transformiert:

$$\mathbf{E}_{\mathrm{mask}} = \mathrm{CNN}_{\mathrm{mask}}(\mathbf{M}),$$

wobei $\mathbf{E}_{\mathrm{mask}} \in \mathbb{R}^{C \times H' \times W'}$ die gleiche Größe wie das Bild-Embedding hat ($C = 256$, $H' = W' = 64$).

Text-Prompts

Für die Prompts des Typs *Text* wird das CLIP-Modell[41] (*Contrastive Language–Image Pre-training*) verwendet. Dieses überführt den vom Nutzer eingegebenen Text in ein Embedding, welches **Dense Prompt Embedding** im Mask-Decoder verwendet werden kann. Das CLIP-Modell wurde dazu entwickelt, Beziehungen zwischen Text und Bildern herzustellen und eignet sich somit gut für diese Aufgabe. Im Paper des SAM[27] wurde dies aber nur angedacht und prototypisch implementiert. Die Komponenten des CLIP sind ein Text-Encoder f_{text} und ein Bild-Encoder f_{img}. Diese Encoder bringen den eingegebenen Text $\mathcal{T}$ und das eingegebene Bild $\mathcal{I}$ in denselben hochdimensionalen Vektorraum. Durch dieses Vorgehen können die resultierenden Vektoren $\mathbf{e}_{\mathrm{text}}$ und $\mathbf{e}_{\mathrm{img}}$ verglichen werden:

$$\mathbf{e}_{\mathrm{text}} = f_{\mathrm{text}}(\mathcal{T}) \quad \text{und} \quad \mathbf{e}_{\mathrm{img}} = f_{\mathrm{img}}(\mathcal{I})$$

Hierfür wird ein kontrastives Trainingsziel verfolgt, welches besagt, dass Text-Bild-Paare die zusammengehören, wie auch Text-Bild-Paare die nicht zusammengehören, trainiert werden. Dabei wird bei zusammengehörenden Paaren die Kosinus-Ähnlichkeit maximiert (nahe 1) und bei ungleichen wird sie minimiert (nahe -1). Diese Funktion ist definiert wie folgt[42]:

$$\text{sim}(\mathbf{e}_{\text{text}}, \mathbf{e}_{\text{img}}) = \frac{\mathbf{e}_{\text{text}} \cdot \mathbf{e}_{\text{img}}}{\|\mathbf{e}_{\text{text}}\| \cdot \|\mathbf{e}_{\text{img}}\|}.$$

Für den Segmentierungsprozess in SAM wird dieses Text-Embedding ($\mathbf{e}_{\text{text}}$) genutzt, um ein **Dense Prompt Embedding** zu erstellen. Hierfür muss Text-Embedding um den Embeddingraum des Bildes erweitert werden. Das Resultat ist somit, dass ein **Dense Prompt Embedding** $\mathbf{E}_{\text{text}} \in \mathbb{R}^{C \times H' \times W'}$ entsteht:

$$\mathbf{E}_{\text{text}} = \text{Expand}(\mathbf{e}_{\text{text}}),$$

wobei Expand$(\cdot)$ die Funktion darstellt, welche den Vektor $\mathbf{e}_{\text{text}}$ auf die räumlichen Dimensionen H' und W' erweitert. Das so erstellte erweiterte Text-Embedding $\mathbf{E}_{\text{text}}$ wird dann über eine elementweise Addition mit dem Bild-Embedding kombiniert:

$$\mathbf{E}_{\text{mod}} = \mathbf{E}_{\text{img}} \oplus \mathbf{E}_{\text{text}},$$

wobei $\oplus$ die Embeddingkombination darstellt (elementweise Addition). Die Darstellung des Prompts als Dense-Promt ist dem Umstand geschulet, dass das Text-Embedding ($\mathbf{e}_{\text{text}}$) immer der Dimension des gesamten Bildes entspricht. Durch die Integration des CLIP-Modells ist es SAM somit möglich, semantische Informationen aus textuellen Nutzereingaben zu verwenden und Regionen des Bildes über Text zu identifizieren.

Bildpositionscodierung

Der Prompt Encoder gibt auch die **Positionscodierung** $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{C \times H' \times W'}$ zurück, die verwendet wird, um die unterschiedlichen Regionen im Eingabebild zu unterscheiden. Diese Positionscodierung wird mittels einer dichten Schicht mit lernbaren Gewichten berechnet. In Abbildung 2.9 ist in der mittleren Bahn zu sehen, dass die Ausgaben sowohl an die Punkt-Prompts als auch an die Box-Prompts addiert werden.

Zusammenfassung der Ausgabe

Die Ausgabe des Prompt-Encoders besteht aus:

- **Sparse Embeddings** für Punkt- und Box-Prompts, mit Größe abhängig von der Anzahl der Punkte oder Boxen.
- **Dense Prompt Embedding** für Masken-Prompts, mit den Maßen $C \times H' \times W'$.
- **Bildpositionscodierung** mit den Maßen $C \times H' \times W'$.

Die Embeddings werden anschließend vom Maskendecoder verwendet, um die finalen Masken vorherzusagen.

2.3.4 Mask Decoder

Der Mask Decoder von SAM ist verantwortlich für die Generierung von Segmentierungsmasken, indem er die Eingabe-Prompts des Nutzers und die vom Image Encoder erstellten Bild-Embeddings verarbeitet. Der Decoder verwendet verschiedene Transformer-Mechanismen, um die Beziehungen zwischen Tokens (*Prompts*) und Bild-Embeddings zu verarbeiten.

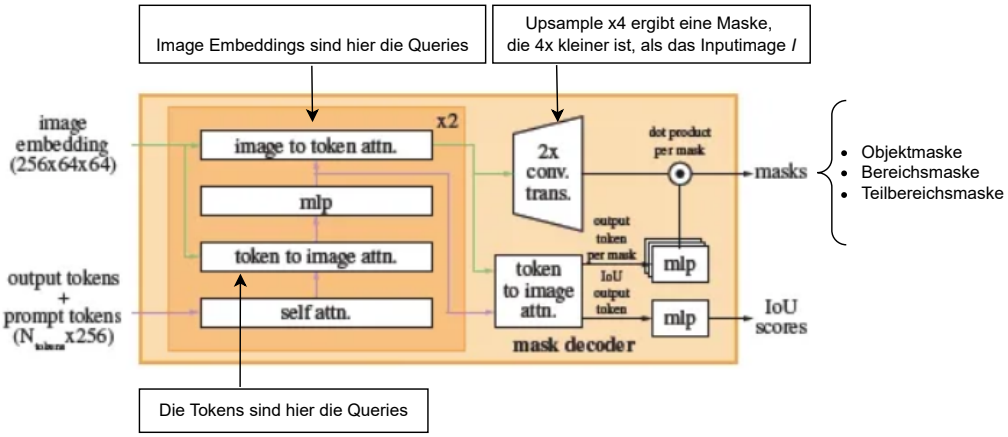

Abbildung 2.10: Schematische Darstellung des Mask Decoders in SAM (angelehnt an[27])

Wie in Abbildung 2.10 zu sehen, besteht die Architektur des Mask Decoders aus folgenden Bestandteilen:

- **Self Attention**: Hier wird der Kontext für die *output token* + *prompt tokens* angereichert.

- **Cross Attention (image to token)**: Anreicherung des Kontextes der image embeddings durch die prompt embeddings.
- **Cross Attention (token to image)**: Anreicherung des Kontextes der prompt embeddings durch die image embeddings.
- **MLP Schichten**: Dienen der nichtlinearen Transformation der unterschiedlichen Token.
- **Upscaling**: Erhöht die Auflösung der image embeddings.

Eingaben

Die in den vorangegangenen Abschnitten beschriebenen Ausgaben werden, wie in Abb. 2.11 zu sehen, angenommen und verarbeitet.

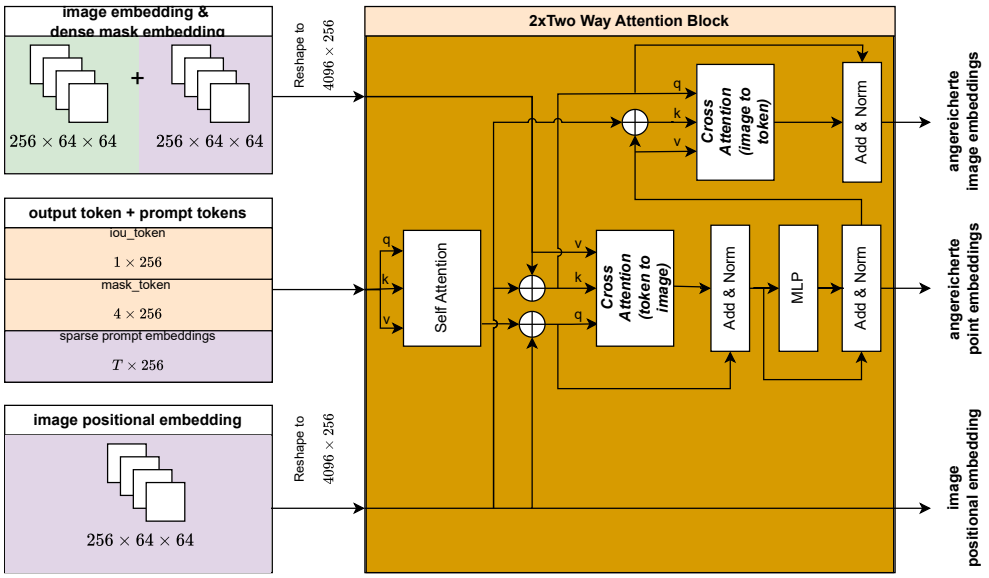

Abbildung 2.11: Eingabeparameter und erste Verarbeitungsschicht des Mask Decoders in SAM

Der Mask Decoder im Segment Anything Modell (SAM) erhält die folgenden Eingaben (siehe Abb. 2.11): $\mathbf{E}_{\text{bild}} \in \mathbb{R}^{C \times H' \times W'}$

- **image embedding** $\mathbf{E}_{\text{img}}^{256\times64\times64}$, berechnet im Image Encoder, plus das **dense mask embedding** $\mathbf{E}_{\text{mask}}^{256\times64\times64}$, berechnet im Prompt Encoder.
- **output token & prompt token**, diese bestehen aus den folgenden drei Elementen:
  1. **iou_token** $\mathbf{t}_{\text{IoU}} \in \mathbb{R}^{1\times256}$. Die Werte dieses Tokens sind nicht mit Benutzereingaben verbunden, sondern werden innerhalb des Mask Decoders angepasst. In jedem Anpassungsschritt erhält dieses Token mehr Informationen aus den Bild- und Prompt-Embeddings. Aus dem iou_token werden die Konfidenzwerte für die vier gültigen Masken abgeleitet (siehe Abb.: 2.12).
  2. **mask_token** $\mathbf{T}_{\text{mask}} \in \mathbb{R}^{4\times256}$. Die Werte des Vektors sind nicht mit Benutzereingaben verbunden, sondern werden in den Schichten des Mask Decoders angepasst. In jedem Anpassungsschritt erhält dieses Array mehr Informationen aus den Bild- und Prompt-Embeddings. Aus den attendierten Masken-Tokens werden die vier Maskenprädiktionsköpfe berechnet, welche die vorhergesagten Segmente definieren.
  3. **sparse prompt embedding** $\mathbf{E}_{\text{prompt}} \in \mathbb{R}^{T\times256}$, berechnet im Prompt-Encoder (Punkte mit Vordergrund- und Hintergrundinformationen).
- **image positional embedding** $\mathbf{E}_{\text{pos}}^{256\times64\times64}$, welche im Prompt-Encoder berechnet wurde und die Positionen innerhalb von I angibt.

Innerhalb des **Two Way Attention Block** werden nun folgende Operationen durchgeführt, wobei die Ausgabe des ersten Durchlaufs als Eingabe des zweiten Durchlaufs genutzt wird. In Abb.2.11 sind diese Ausgaben als **angereicherte point embeddings** und **angereicherte image embeddings** beschrieben. Nachfolgend der Ablauf des ***Two Way Attention Block***:

1. **Self Attention**: Die point embeddings werden durch Self Attention verbessert:

$$\mathbf{E}'_{\text{points}} = \text{Attention}(\mathbf{E}_{\text{points}} + \mathbf{E}_{\text{pos}})$$

   , wobei $\mathbf{E}_{\text{points}}$ die Kombination aus $\mathbf{E}_{\text{prompt}}$, $\mathbf{t}_{\text{IoU}}$ und $\mathbf{T}_{\text{mask}}$ ist.

2. **Token-to-Image Attention**: $\mathbf{E}'_{\text{points}}$ wird nun weiter um Kontextinformationen erweitert, sowie $\mathbf{E}_{\text{pos}}$. Damit kann der Verarbeitungsschritt wie folgt beschrieben

werden:

$$\mathbf{E}''_{\text{points}} = \text{Attention}(\mathbf{E}'_{\text{points}} + \mathbf{E}_{\text{pos}}, \mathbf{E}_{\text{img}}).$$

3. **Add & Norm**: $\mathbf{E}''_{\text{points}}$ wird nun durch eine Normalisierungs-Schicht und eine MLP-Schicht angereichert:

$$\mathbf{E}'''_{\text{points}} = Norm(MLP(Norm(\mathbf{E}''_{\text{points}})))$$

4. **Image-to-Token Attention**:

   Das image embedding wird nun weiter angereichert, um weitere Kontextinformationen zu erhalten. Hierbei ist das gefaltete image embedding (4096×256) die query der Attention:

$$\mathbf{E}'_{\text{img}} = \text{Attention}(\mathbf{E}'''_{\text{points}} + \mathbf{E}_{\text{pos}}, \mathbf{E}_{\text{img}}).$$

Die verwendete $Attention(X) = Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ ist wie folgt definiert[53]:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V$$

,

wobei $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ trainierbare Matizen sind.

Die verwendete $Attention(X, Y) = Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ ist wie folgt definiert[53]:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{Y}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{Y}\mathbf{W}_V$$

,

wobei auch hier $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ trainierbare Matizen sind.

Die Attention wird dann durch Softmax berechnet:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

Die Ausgabe der zweifachen Durchführung des ersten Abschnitts des Mask-Decoders ist somit definiert als:

$$\mathbf{E}^{final}_{\text{points}} = \mathbf{E}'''_{\text{points}} \text{ und } \mathbf{E}^{final}_{\text{img}} = \mathbf{E}'_{\text{img}}$$

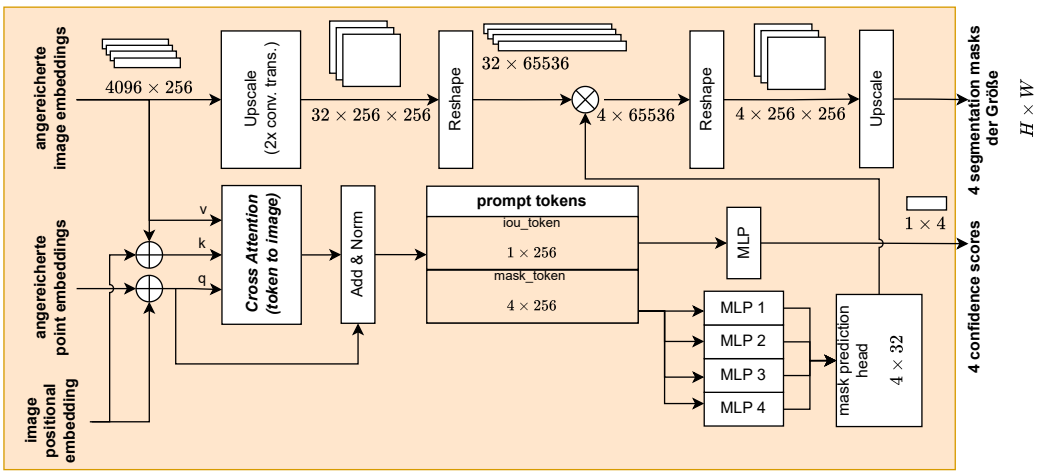

Abbildung 2.12: Eingabeparameter und zweite Verarbeitungsschicht des Mask-Decoders in SAM

Im Anschluss an die zweifache Ausführung des ***Two Way Attention Block*** wird der zweite Abschnitt des Mask-Decoder durchgeführt. Der Ablauf ist hier wie folgt definiert (siehe Abb.2.12):

1. **Upscale image embedding**: Hier wird das $\mathbf{E}_{\text{img}}^{final=4096\times256}$ in das Format $32 \times 256 \times 256$ geändert.

2. **Reshape**: Nun wird das image embedding $\mathbf{E}_{\text{img}}^{final}$ in die Form 32×65536 transformiert.

3. **Token-to-Image Attention**: $\mathbf{E}_{\text{points}}^{final}$ wird weiter um Kontextinformationen erweitert, sowie $\mathbf{E}_{\text{pos}}$. Damit kann der Verarbeitungsschritt wie folgt beschrieben werden:

$$\mathbf{E'}_{\text{points}}^{final} = \text{Attention}(\mathbf{E}_{\text{points}}^{final} + \mathbf{E}_{\text{pos}}, \mathbf{E}_{\text{img}}^{final})$$

4. **Add & Norm**: $\mathbf{E'}_{\text{points}}^{final}$ wird nun durch eine Normalisierungs-Schicht angereichert:

$$\mathbf{E''}_{\text{points}}^{final} = Norm(\mathbf{E''}_{\text{points}}))$$

5. **prompt token Verarbeitung**: der ***iou_token*** wird mithilfe eines MLP in den finalen **confidence score** transformiert in dem Format 1×4. Die ***mask_token***

werden durch 4 einzeln trainierbare MLP-Schichten zu dem:

$$\mathbf{E}_{mask}^{final} = \text{mask prediction head}^{4\times32},$$

wobei jede der so vorhergesagten Masken eine unterschiedliche Granularität bieten (Maske 2-4).

6. **Multiply & Reshape**: $\mathbf{E'}_{\text{img}}^{final}$ wird mithilfe der 4 validen Vorhersagemasken zu den auf dem Image liegenden Ausgabemasken. Dies kann wie folgt beschrieben werden:
$$\mathbf{E''}_{\text{img}}^{final=4\times256\times256} = \mathbf{E'}_{\text{img}}^{final} \times \mathbf{E}_{mask}^{final}$$

7. **Upscale**: Die erstellten finalen 4 Masken werden den Abmessungen des Orignalbildes transformiert. Außerdem werden die Masken $\mathbf{E''}_{\text{img}}^{final}$ als ***mask_ token*** für weitere Segmentierungen genutzt.

2.3.5 Ausgaben des Mask Decoders

Der Mask Decoder liefert somit:

- **Vier niedrigauflösende Segmentierungsmasken**: Diese Masken werden auf die ursprüngliche Größe des Eingabebildes hochskaliert.
- **Konfidenzwerte**: Für jede der vier Segmentierungsmasken wird ein Konfidenzwert aus dem IoU-Token abgeleitet.

3 Implementierung

Der in diesem Abschnitt beschriebene Software-Engineering-Prozess beschreibt das strukturierte Vorgehen der Implementierung, von der Anforderungsanalyse, Planung und Evaluierung bis hin zur Fertigstellung des Frameworks zur Beantwortung der Hypothesen. Es wird erläutert, wie das SAM angepasst und erweitert wurde, um eine Vergleichbarkeit der Ergebnisse des Modells mit dem Basismodell DeepForest zu gewährleisten.

Der komplette Code sowie die Dokumentation des Projektes sind nach Freigabe unter GitHub zu finden.

3.1 Anforderungsanalyse

Die Anforderungen an das zu implementierende System umfassen nicht nur die Beantwortung der aufgestellten Hypothesen (siehe Abschn. 1.3). Es sollen Google Colab Notebooks genutzt werden, wobei die Wiederverwendbarkeit der einzelnen Teilimplementierungen sichergestellt werden muss, damit diese auch für andere Zwecke nutzbar sind. Zudem dürfen keine kommerziellen Bestandteile im System enthalten sein. Aus diesen Anforderungen ergeben sich folgende Qualitätsanforderungen an das zu implementierende System:

- **Modellergebnisvergleichbarkeit**: Die Ergebnisse des SAM müssen mit dem spezialisierten DeepForest-Modell vergleichbar sein.
- **Ausreichende Genauigkeit der Segmentierung**: Das System muss in der Lage sein, Baumkronen in hochauflösenden Satellitenbildern richtig und zuverlässig zu segmentieren.
- **Generalisierbarkeit der Ergebnisse des Modells**: Das System sollte in unterschiedlichen urbanen Regionen, die nicht Teil der Trainingsdaten sind, zuverlässig arbeiten.

- **Validierbarkeit der Ergebnisse**: Zur Validierung der Modellergebnisse müssen Ground-Truth-Daten vorliegen, um eine valide Aussage über die Ergebnisse machen zu können.
- **Skalierbarkeit in Bezug auf geographische Ausdehnung**: Das System muss in der Lage sein, große geographische Bereiche zu segmentieren und auf unterschiedliche Städte anwendbar zu sein.

3.2 Software Design

Um die beschriebenen Anforderungen sinnvoll abzubilden, wurde eine modulare Architektur gewählt. Dieser Architekturstil gewährt die Möglichkeit, die einzelnen Aufgabengebiete zur Erfüllung der Anforderungen modular zu betrachten und zu testen und ist in Notebooks umsetzbar. Der Schnitt wird anhand der funktionalen Aufgaben, die zu erfüllen sind, gemacht. Die Komponenten des Systems sind folgende:

- **Datenbeschaffung und Datenverarbeitung**: Dieses Modul übernimmt die Aufgabe der Datenbeschaffung von hochauflösenden Satellitenbildern eines beliebigen geographischen Gebiets, der Zerlegung und Transformation in die richtigen Auflösungen für den SAM-Mask-Decoder (siehe Abschn. 2.3.4) sowie die anschließende Datenverarbeitung. Das Ergebnis dieses Moduls ist ein Datensatz D zur Nutzung der Feinabstimmung des SAM.
- **SAM-Training**: Dieses Modul verwendet die Ausgabe des vorangegangenen Moduls (den Datensatz D), um das SAM mithilfe der Ground-Truth des Katasteramts und der manuellen Segmentierung fein abzustimmen. Das Ergebnis dieses Moduls ist ein SAM, welches für die Erkennung von Segmenten auf hochauflösenden Satellitenbildern ohne Nutzereingaben optimiert wurde.
- **CLIP-Training**: Dieses Modul nutzt den in Punkt eins erstellten Datensatz D, um ihn für eine Feinabstimmung des CLIP-Modells zu verwenden. Das Ergebnis dieses Moduls ist ein CLIP-Modell, welches auf die Klassifizierung von *Bäumen* und *keinen Bäumen* fein abgestimmt wurde.
- **Ergebnisproduktion**: Dieses Modul kombiniert das SAM aus Punkt zwei mit CLIP aus Punkt drei und verbindet diese. Das so erstellte SAM ist in der Lage, Baumkronen auf hochauflösenden Satellitenbildern zu erkennen. Dieses so er-

stellte Modell zur Instanzsegmentierung wird auf unterschiedlichen geographischen Ausdehnungen verwendet, um Baumsegmente zu bilden. Außerdem wird für den gleichen geographischen Raum das DeepForest-Modell angewendet. Das Ergebnis dieses Moduls sind Segmentierungsergebnisse mit einem Score des fein abgestimmten Basismodells (SAM) und Segmentsprungergebnisse mit einem Score für das spezialisierte Modell (DeepForest).

- **Validierung und Evaluation**: In diesem Modul werden die in Punkt vier beschriebenen Segmentierungsergebnisse durch eine Vielzahl von Metriken ausgewertet und die Güte der Modelle umfassend vergleichbar gemacht. Das Ergebnis dieses Moduls sind verschiedene Metriken zum Leistungsvergleich des fein abgestimmten Basismodells und des Spezialmodells.

Innerhalb jeder Komponente wird ein funktionaler Ansatz gewählt, da eine Implementierung über einen objektorientierten Ansatz bei der Nutzung von Notebooks wenig sinnvoll erscheint. Der Vorteil der Wiederverwendbarkeit von Objekten kann nicht ausgespielt werden, da diese nicht notebookübergreifend existieren, was zu doppeltem schwer wartbarem Code führen würde.

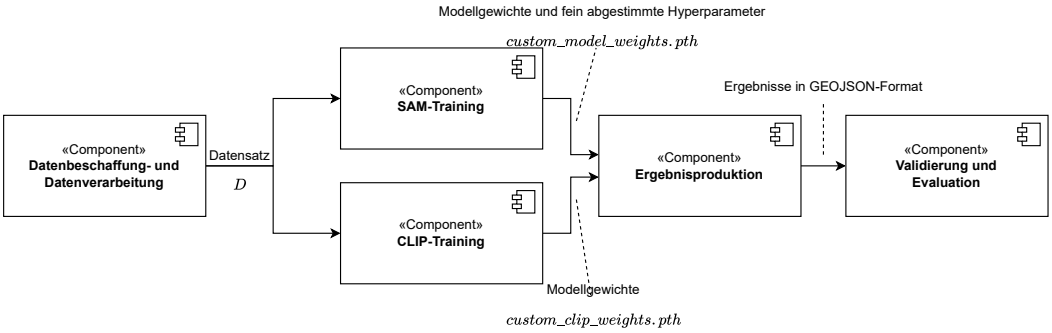

Abbildung 3.1: Komponentendiagramm der implementierten Architektur zur Beantwortung der aufgestellten Hypothesen.

Implementierung Die Implementierung des Systems folgt dem zuvor beschriebenen Design. Dies führt zu einer Architektur, die in Abb. 3.1 dargestellt ist und die folgenden Schritte beinhaltet:

3.3 Datensätze und Datenverarbeitung

Die in diesem Abschnitt beschriebene funktionale Implementierung liegt innerhalb der Komponente Datenbeschaffung und Datenverarbeitung (siehe Abb. 3.1).

3.3.1 Datenbeschaffung

Die Bildbeschaffung wird unter Verwendung der Google Earth API implementiert. Die API stellt hochauflösende Satellitenbilder zur Verfügung, sofern sie für den betrachteten geographischen Bereich vorliegen. Ziel ist es, eine große geographische Ausdehnung in kleinere Kacheln zu zerlegen (Bounding Boxen), die als Eingabebilder und Grundlage für die anschließende Segmentierung dienen. Die Auswahl des Gebiets erfolgt durch die Festlegung eines geografischen Zentrums, um welches die Satellitenbilder geladen werden. Außerdem wird ein Zoom-Level mitgegeben, der angibt, in welcher Auflösung die Bilder von der API angefragt werden. Er hält sich dabei an den Google-Earth-Standard, bei dem höhere Zoom-Level eine höhere Detailauflösung bedeuten. Zur Akquisition der RGB-Bilder der Masterarbeit wird ein **Zoom-Level von 21** verwendet, was der maximalen Detailstufe entspricht und eine sehr hohe Auflösung ermöglicht (ca. 30 cm/px).

Festlegung des geographischen Bereichs und Erstellung der Kacheln Es wird ein Zentrum über $(lat_{center}, lon_{center})$ angegeben. Dieses Zentrum befindet sich bei den Koordinaten $[53.56126, 9.96207]$ und ist somit der Bereich um die Hamburger Sternschanze. Der Gesamtbereich um dieses Zentrum wird anschließend in ein Raster von 16×16-Kacheln aufgeteilt, wobei jede Kachel dieselbe Pixelgröße und Ausdehnung besitzt. Dies ist wichtig für die Verarbeitung im Modell, da es gleich große Kacheln mit derselben Auflösung benötigt. Die Wahl des zu beobachteten Gebiets resultiert aus dem Vorhanden sein von Ground-Truth Daten dieses Bereichs, welche im späteren Verlauf für die Erstellung des Trainingsdatensatzes erforderlich sind.

Es sei:

- N_x: die Anzahl der Kacheln in x-Richtung,
- N_y: die Anzahl der Kacheln in y-Richtung,
- Px: die Anzahl der Pixel in Breite und Höhe der Kachel (1024 Pixel),

- z: der Zoom-Level,
- $S = 256 \cdot 2^z$: die Größe der Karte in Pixeln bei Zoom-Level z.

Berechnung der Bounding Boxes erfolgt durch die initiale Bestimmung der Pixelkoordinaten des geografischen Zentrums $(lat_{center}, lon_{center})$ und der anschließenden Projektion auf die entsprechende Zoom-Ebene. Dies ist durch die Umwandlung der geografischen Koordinaten in Pixelkoordinaten implementiert worden und basiert auf der Web-Mercator-Projektion[56]. Diese Projektion wird auch von Mapdiensten wie Google Maps genutzt:

$$x_{center} = \frac{S}{360} \cdot (lon_{center} + 180)$$
$$y_{center} = \frac{S}{2\pi} \cdot \left(\pi - \ln\left(\tan\left(\frac{\pi}{4} + \frac{lat_{center} \cdot \pi}{360}\right)\right)\right)$$

Der Längengrad lon_{center} wird auf die horizontale Ausdehnung von 360° abgebildet und der Breitengrad lat_{center} wird über eine Projektion auf die vertikale Ausdehnung transformiert. Es ist notwendig, bei den Berechnungen in Gradzahlen auf diese Weise vorzugehen, um die Erdkrümmung zu berücksichtigen und eine angemessene genaue Umrechnung in Pixelkoordinaten zu erhalten. Nun muss für jede Kachel die minimale und die maximale Pixelkoordinate berechnet werden. Es Sei $w_x = Px$ die Kachelbreite und es Sei $w_y = Px$ die Kachelhöhe in Pixeln. Der geographische Bereich jeder Kachel wird dann durch die Umrechnung der Pixelkoordinaten in geographische Koordinaten (Breiten- und Längengrade) bestimmt:

$\text{min\_x}, \text{max\_x}$ die minimalen und die maximalen x-Koordinaten der Kachel in Pixeln,
$\text{min\_y}, \text{max\_y}$ die minimalen und die maximalen y-Koordinaten in Pixeln.

Die geografischen Koordinaten der Bounding-Box einer Kachel werden wie folgt berechnet:

$$\text{lon}_{min} = \frac{360 \cdot \text{min\_x}}{S} - 180, \quad \text{lon}_{max} = \frac{360 \cdot \text{max\_x}}{S} - 180$$

$$\text{lat}_{min} = \frac{180}{\pi} \cdot \left(\text{atan}\left(\text{sinh}\left(\frac{\pi - 2\pi \cdot \text{min\_y}}{S}\right)\right)\right),$$
$$\text{lat}_{max} = \frac{180}{\pi} \cdot \left(\text{atan}\left(\text{sinh}\left(\frac{\pi - 2\pi \cdot \text{max\_y}}{S}\right)\right)\right)$$

Diese Projektion ist die Umkehrung der zuvor beschriebenen Projektion und dient dazu, die Pixelkoordinaten wieder in Breiten- und Längengrade zurück zu überführen.

Erstellung der Kachelgrenzen (Bounding Boxes) Jede Kachel T_{ij} im Raster wird durch ihre minimalen und maximalen Breiten- und Längengrade definiert, die aus den oben berechneten Pixelkoordinaten abgeleitet werden. Die Kacheln überdecken damit das gesamte betrachtete Gebiet, wobei die räumliche Auflösung durch die Anzahl der Kacheln und den Zoom-Level bestimmt wird.

$$T_{ij} = \{(lat, lon) | \text{lon}_{min} \leq lon \leq \text{lon}_{max}, \text{lat}_{min} \leq lat \leq \text{lat}_{max}\}$$

Zusammenhang mit dem Zoom-Level Der Zoom-Level z spielt eine wichtige Rolle bei der Auflösung der Bilder. Ein Zoom-Level von 21 entspricht dabei einer sehr detaillierten Darstellung, bei der einzelne Objekte (im Rahmen der Bearbeitung dieser Arbeit handelt es sich um Bäume) auf Pixelebene erkennbar sind. Er beeinflusst die Größe der Kachel in geografischen Koordinaten und damit auch die Granularität der Bonding-Boxen. Jeder Zoom-Level erhöht die Anzahl der Kacheln, die zur Darstellung der Erdoberfläche benötigt werden, exponentiell. Für jede Erhöhung des Zoom-Levels verdoppelt sich die Auflösung, d.h. es werden viermal so viele Kacheln benötigt wie im vorherigen Level. Beispielsweise wird die gesamte Erde bei Zoom-Level 1 in 4 Kacheln dargestellt (2×2), bei Zoom-Level 2 sind es 16 Kacheln (4×4) und bei Zoom-Level 20 sind es bereits über eine Billion Kacheln ($2^{20} \times 2^{20}$)[33].

Bei Zoom-Level 21 entspricht jede Kachel ungefähr einer räumlichen Abdeckung von wenigen Metern in der realen Welt, was eine präzise Segmentierung, auch von kleineren Objekten, ermöglicht.

Als Resultat dieser Implementierung entsteht ein Datensatz mit sehr hochauflösenden (ca. 30 cm/px) RGB-Satellitenbildern. Der Datensatz besteht aus 256 Einzelbildern, die nahtlos, um ein Zentrum herum angeordnet sind (siehe Abb. 3.2).

Abbildung 3.2: Teilausschnitt des erstellten vorläufigen Datensatzes

Der Trainingsdatensatz erstreckt sich geografisch von 53.55576797, 9.95282327 im Südwesten bis 53.566752407, 9.971317045 im Nordosten. Diese Grenzen geben die räumliche Ausdehnung des Datensatzes in geografischen Koordinaten gemäß dem Koordinatensystem EPSG: 4326[45] an, welches das World Geodetic System 1984 (WGS 84) verwendet. In diesem Koordinatensystem werden Längen- und Breitengrad in Grad angegeben. Dieser Bereich umfasst ein Gebiet in Hamburg, das in mehrere Kacheln unterteilt wurde. Diese Kacheln dienen als Grundlage für die Bildgenerierung und die anschließende Segmentierung.

3.3.2 Datenverarbeitung

Der im Abschnitt 3.3.1 beschriebene Datensatz wird anschließend wie folgt verarbeitet: Jede Kachel durchläuft weitere Schritte, die darauf abzielen, die Satellitenbilder so zuzuschneiden, dass nur Straßen, Fußgängerwege und andere relevante Teile des Straßennetzes sichtbar sind. Dies geschieht, um eine überprüfbare Ground-Truth für die Segmentierung von Straßenbäumen zu haben, basierend auf den Daten des Katasteramts, die ausschließlich Bäume im öffentlichen Straßenraum umfassen.

Pufferung der Straßenränder Initial wird ein Straßennetz über die OpenStreetMap[51] (OSM) API heruntergeladen. Diese erwartet einen `city_name` Parameter, um die gewünschte geografische Ausdehnung zurückzugeben. Es wurden folgende Typen von Kanten (Straße, Fußgängerwege, etc.) als georeferenzierte Vektordatei über OSM-Keys heruntergeladen[2]: highway_edge_types = { motorway, motorway_link, trunk, trunk_link, primary, primary_link, secondary, secondary_link, tertiary, tertiary_link, residential, living_street}

Für jede Straßenkante wird ein Pufferbereich berechnet, um Straßen und Wege als Polygone zu modellieren. Der Pufferabstand d wird hierbei festgelegt als:

$$d = 0.00018\,\text{Grad}$$

Der hier gewählte Puffer wird genutzt, um den Bereich um die jeweilige Kante zu ziehen und somit ein Polygon zu erstellen, welches das komplette Straßennetz mit allen definierten Kanten umfasst. Die Zahl d ist dabei durch Tests entstanden und stellt eine Abstraktion der tatsächlichen Breite der jeweiligen Kante dar.

Berechnung der Polygone Die so erstellten Kanten des Straßennetzes E werden gefiltert, auf die vom Benutzer gemachte Auswahl von Straßentypen highway_edge_types:

$$E' = \{e \in E \mid e.\text{highway} \in \text{highway\_edge\_types}\}$$

Für jede Kante $e \in E'$ wird ein Pufferpolygon Pol_e durch Anwendung eines Puffers mit dem Abstand d erzeugt. Dies ergibt eine Menge von Polygonen:

$$Pol = \{Pol_e \mid e \in E'\}$$

Anschließend werden die Polygone zu einem finalen Polygon Pol_{merged} vereinigt:

$$Pol_{\text{merged}} = \bigcup_{e \in E'} Pol_e$$

Dieses Polygon Pol_{merged} stellt die Fläche dar, die für Straßen, Fußgängerwege und andere vom Nutzer ausgewählte Kantentypen auf den Satellitenbildern relevant ist.

Bereinigung der Satellitenbilder Anschließend wird Pol_{merged} verwendet, um die Satellitenbilder zu bereinigen. Dies wird implementiert über die Überlappung der Bilddaten mit Pol_{merged}, sodass nur die Bereiche nicht schwarz sind, die sich innerhalb des Polygons befinden. Für jedes GeoTIFF-Bild I wird das resultierende Bild I_{cropped} durch die folgende Operation definiert:

$$I_{\text{cropped}} = I \cap Pol_{\text{merged}}$$

Dabei wird nur der Bereich des Bildes I erhalten, der innerhalb des Polygons Pol_{merged} liegt. Dies führt zu einer Darstellung von Straßen und anderen relevanten Kanten, während alle anderen Bereiche ausgeblendet werden (mit schwarzen Pixeln beschrieben).

Als Ergebnis der Datenverarbeitung ergibt sich eine Menge von Kacheln im Datensatz, welche dem Erscheinungsbild in Abb. 3.3 entsprechen.

Abbildung 3.3: Teilausschnitt des erstellten Datensatzes nach der Weiterverarbeitung

Es existieren Bäume innerhalb des schwarzen Bereichs, die unter anderem Hintergärten und andere schwer oder gar nicht zugängliche Bereiche umfassen. Diese Bereiche können nicht visuell validiert werden. Auch gibt es hierfür keine Katasteramt-Ground-Truth-Daten, weshalb sie für die spätere Feinabstimmung nicht geeignet sind.

3.3.3 Erstellung des Datensatzes mit Ground-Truth für das Segment Anything Model (SAM)

Um eine Feinabstimmung des SAM zu ermöglichen, muss der hier erstellte Datensatz für den Mask-Decoder von SAM interpretierbar sein. Für diesen Zweck muss ein Datum des Trainingsdatensatzes das Bild I, die Maske M und die Bounding-Box B erhalten. Um die Maske und die Bounding-Box zu erstellen, wird unter Zuhilfenahme der Daten des Hamburger Straßenbaumkatasters[26] zu den Bauminstanzen zusätzlich eine manuelle Verfeinerung der Segmentierung der Baumkronen unternommen.

Polygonerstellung basierend auf Baumkronen Die Baumdaten des Katasteramts enthalten Informationen über die Position der Bäume sowie die Durchmesser ihrer Baumkronen (feature `kronendurchmesser`). Diese Informationen werden genutzt, um Polygone zu erstellen, welche die Baumkronen approximieren. Die Eingabedaten P aus den GeoJSON-Daten des Katasteramts bestehen aus den Baumpositionen (x_i, y_i) und den Durchmessern der Baumkronen d_i für jeden Baum $i \in \{1, \ldots, N\}$, wobei N die Gesamtzahl der Bäume ist.

Für jeden Baum wird ein kreisförmiges Polygon mit dem Durchmesser d_i erzeugt. Der Radius r_i des Polygons ist dabei gegeben durch:

$$r_i = \frac{d_i}{2} \cdot s$$

wobei s ein Skalierungsfaktor ist (hier $s = 1.5$), um die Baumkrone größer zu erfassen, als sie in der GeoJSON angegeben ist. Grund hierfür ist das Alter des Datensatzes des Straßenbaumkatasters von Hamburg, das auf das Jahr 2020 zurückgeht. Seitdem haben sich die Baumkronen vergrößert. Auch durch visuelle Sichtung der Satellitenbilder konnte verifiziert werden, dass die Durchmesser d der Baumkronen zu klein bemessen sind. Das resultierende Polygon B_i für jeden Baum wird durch ein Rechteck approximiert, das die Grenzen des Kastens um den Punkt (x_i, y_i) festlegt:

$$B_i = \{(x_{\min}, y_{\min}), (x_{\min}, y_{\max}), (x_{\max}, y_{\max}), (x_{\max}, y_{\min})\}$$

mit den Grenzen:

$$x_{\min} = x_i - r_i, \quad x_{\max} = x_i + r_i$$

$$y_{\min} = y_i - r_i, \quad y_{\max} = y_i + r_i$$

Diese Polygone werden dann als GeoJSON-Datei gespeichert.

Manuelle Anpassung und Erstellung der Ground Truth Für die Verbesserung der Qualität der Daten des Straßenbaumkatasteramts wurden die generierten Polygone visuell überprüft und bei Bedarf unter Zuhilfenahme der Informationen innerhalb der geoTIFF-Datei korrigiert. Im Rahmen dieser manuellen Überprüfung der Katasteramtdaten wurden 1600 Bäume segmentiert, um diese als überprüfte Ground-Truth für die Erstellung des Trainingsdatensatzes zu nutzen. Die finalen Polygone, welche die Positionen der Baumkronen darstellen, sind in einer GeoJSON-Datei gespeichert worden (`ground-truth-hand-made.geojson`).

Für jeden Baum $j \in \{1, \ldots, 1600\}$ wurde ein Polygon P_j gezeichnet, das die genaue Ausdehnung der Baumkrone in der Satellitenaufnahme repräsentiert. Diese Polygone sind durch die Koordinaten ihrer Eckpunkte definiert:

$$P_j = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

wobei n die Anzahl der Eckpunkte des Polygons ist.

Erstellung von Schwarz-Weiß-Masken und Bounding Boxes Die manuell erstellten Polygone P_j werden in diesem Schritt genutzt, um für alle Satellitenbilder I binäre Masken M_j zu erstellen. Die erstellten Masken M_j stellen ein binäres Bild dar, welches sich dadurch auszeichnet, dass Pixel, die innerhalb des Polygons liegen $= 1$ und Pixelwerte, die außerhalb des Polygons liegen $= 0$ sind. Die Maske M_j ist definiert durch:

$$M_j(p, q) = \begin{cases} 1 & \text{wenn } (p, q) \in P_j \\ 0 & \text{sonst} \end{cases}$$

Zu den erstellten binären Masken wird für jedes Baumsegment eine Bounding Box B_j berechnet. Diese Bounding-Box ist definiert als das kleinstmögliche Rechteck, welches das gesamte Polygon P_j umschließt. Sie wird durch die minimalen und maximalen x- und y-Koordinaten des Polygons bestimmt:

$$B_j = [x_{\min}, y_{\min}, x_{\max}, y_{\max}]$$

mit:

$$x_{\min} = \min(x_1, x_2, \ldots, x_n), \quad x_{\max} = \max(x_1, x_2, \ldots, x_n)$$
$$y_{\min} = \min(y_1, y_2, \ldots, y_n), \quad y_{\max} = \max(y_1, y_2, \ldots, y_n)$$

Speicherung und Struktur des Datensatzes Der finale Datensatz besteht aus drei Komponenten: 1. Dem Originalbild I_j 2. Die zugehörige Schwarz-Weiß-Maske M_j 3. Die Bounding Box B_j, die das Baumsegment umschließt.

Die drei Komponenten werden in separaten Dictionaries gespeichert, um den Zugriff während des trainings zu vereinfachen und eine konstante Laufzeit zu ermöglichen:

$$\text{image\_dict} = \{\text{base\_name}_j : I_j \mid j = 1, \ldots, 1600\}$$
$$\text{mask\_dict} = \{\text{base\_name}_j : M_j \mid j = 1, \ldots, 1600\}$$
$$\text{bbox\_dict} = \{\text{base\_name}_j : B_j \mid j = 1, \ldots, 1600\}$$

Die so erstellte Struktur ermöglicht es, für jedes Baumsegment j effizient auf das zugehörige Bild I_j, die Maske M_j und die Bounding Box B_j zuzugreifen. Ein Datum innerhalb des Trainingsdatensatzes für SAM entspricht somit der Abbildung 3.4.

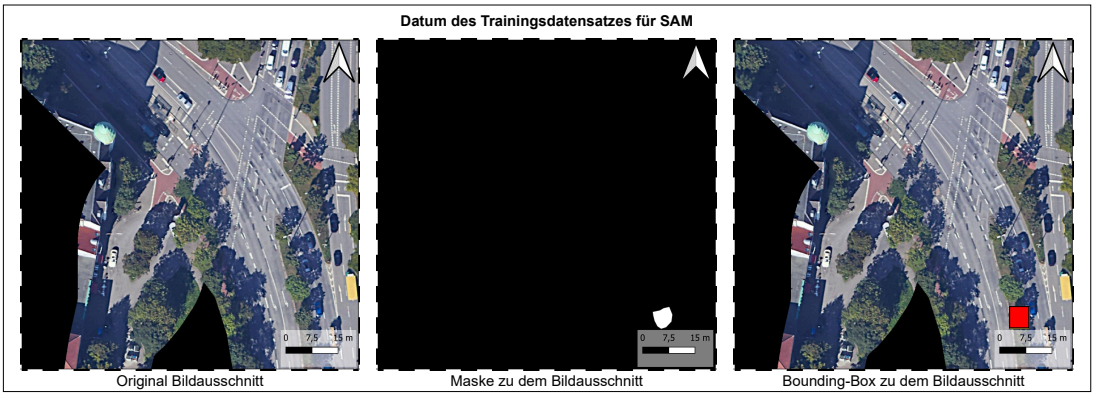

Abbildung 3.4: Ein Datum innerhalb des Trainingsdatensatzes für SAM

3.4 Modellanpassungen und Optimierungen

Die Modellanpassungen müssen unternommen werden, um die in der Anforderungsanalyse angesprochene Vergleichbarkeit der Modellausgaben zu erreichen.

Die Feinabstimmung des Basismodells umfasst drei Stufen. Hierbei ist zu beachten, dass Punkt eins und Punkt zwei dem SAM zuzuordnen sind, während Punkt drei primär das CLIP-Modell betrifft, welches anschließend in das SAM integriert wird.

1. **Feinabstimmung des Mask Decoder**: Ziel ist es, eine Verbesserung der Segmentierung von Baumkronen zu erhalten. Dies ist gerade für ineinander verwachsene Baumkronen notwendig, da diese sonst ggf. optisch nicht voneinander getrennt werden können.

2. **Hyperparametertraining**: In diesem Schritt werden die Hyperparameter fein abgestimmt, die notwendig sind, um dem Modell die Fähigkeit zu verschaffen, auch ohne Eingabeprompts des Benutzers Segmente zuverlässig zu erstellen.

3. **Feinabstimmung des CLIP**: Das CLIP-Modell muss für die Erkennung von *kein Baum* und *Baum* trainiert werden. Die Anforderung ergibt sich aus der Tatsache, dass SAM ohne die CLIP-Erweiterung keine Möglichkeit bietet, Bauminstanzen zu segmentieren, da es an Klassen-Labels mangelt. Für eine Vergleichbarkeit der beiden Modelle ist dies jedoch zwingend erforderlich.

In den folgenden Abschnitten werden die genauen Vorgehensweisen erläutert und die Implementierungsdetails beschrieben.

3.4.1 Finetuning des SAM

Dieser Abschnitt beschreibt Punkt eins und zwei der oben beschriebenen Aufzählung. Punkt drei wird im darauffolgenden beschrieben, da sie unabhängig von dem Mask Decoder und dem SAM implementiert wird. Innerhalb des modularen Aufbaus des Systems liegt diese Implementierung innerhalb der Komponente SAM-Training (siehe Abb. 3.1).

1. Feinabstimmung des Mask-Decoder

Datensatz Der verwendete Datensatz $\mathcal{D} = \{(I_n, M_n, B_n)\}_{n=1}^{N}$ (wie in Abschn. 3.3.2 beschrieben) besteht aus N Trainingsbeispielen, wobei jedes Beispiel aus folgenden Komponenten besteht:

- I_n: Satellitenbild
- M_n: Ground-Truth-Maske der Baumkrone
- B_n: Bounding Box, die um die Maske M_n gezogen wurde

Vorverarbeitung der Daten Jedes Eingabebild I_n wird entsprechend den Anforderungen des SAM vorverarbeitet[27]:

$$I'_n = \text{Resize}(\text{Normalize}(I_n)) \tag{3.1}$$

wobei Normalize$(\cdot)$ die Pixelwerte normalisiert und Resize$(\cdot)$ das Bild auf die vom Modell benötigte Eingabegröße skaliert.

Image- und Prompt-Embeddings Der Image Encoder ϕ_{enc} des SAM erzeugt ein Bild-Embedding E_n:

$$E_n = \phi_{\text{enc}}(I'_n) \tag{3.2}$$

Die Bounding Box B_n wird durch den Prompt Encoder ϕ_{prompt} in ein Prompt-Embedding P_n überführt:

$$P_n = \phi_{\text{prompt}}(B_n) \tag{3.3}$$

Maskenprädiktion Der Mask Decoder ϕ_{dec}, der fein abgestimmt werden soll, erzeugt die vorhergesagte Maske $\hat{M}_n$ unter Verwendung des Bild- und Prompt-Embeddings:

$$\hat{M}_n = \phi_{\text{dec}}(E_n, P_n) \tag{3.4}$$

Die Nutzung des Image Encoders, die Nutzung des Prompt Encoder und die Nutzung des Mask Decoders funktionieren im Detail, wie in Abschnitt 2.3 beschrieben.

Loss-Funktion Zur Bewertung der Übereinstimmung zwischen der vorhergesagten Maske $\hat{M}_n$ und der Ground-Truth-Maske M_n wird der Binäre Kreuzentropie-Loss:

$$\mathcal{L}_n = -\frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} \left[M_n^{(i,j)} \log \hat{M}_n^{(i,j)} + \left(1 - M_n^{(i,j)}\right) \log \left(1 - \hat{M}_n^{(i,j)}\right) \right] \tag{3.5}$$

verwendet, wobei H und W die Höhe und Breite der Maske sind.

Optimierung Die Parameter θ_{dec} des Mask-Decoders werden durch Minimierung des durchschnittlichen Loss über den Trainingsdatensatz aktualisiert:

$$\theta_{\text{dec}}^* = \arg\min_{\theta_{\text{dec}}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n \tag{3.6}$$

Trainingsprozedur Die Prozedur wird mithilfe einer Lernrate von $\eta = 1 \times 10^{-4}$ vorgenommen und das Training erfolgt über $E = 100$ Epochen.

Der Datensatz wird in Trainings- und Testdaten aufgeteilt:

$$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}} = \text{TrainTestSplit}(\mathcal{D}, \text{Testgröße} = 0{,}2) \tag{3.7}$$

In jeder Epoche wird folgendermaßen vorgegangen:

1. **Training:** Für jedes $(I_n, M_n, B_n) \in \mathcal{D}_{\text{train}}$:
   - Berechne E_n, P_n und $\hat{M}_n$.
   - Berechne den Loss $\mathcal{L}_n$.
   - Aktualisiere θ_{dec}:
   $$\theta_{\text{dec}} \leftarrow \theta_{\text{dec}} - \eta \nabla_{\theta_{\text{dec}}} \mathcal{L}_n \tag{3.8}$$
2. **Validierung:** Evaluierung des Modells auf $\mathcal{D}_{\text{test}}$ ohne Parameteraktualisierung.
3. **Checkpointing:** Speichere θ_{dec}, wenn der Validierungs-Loss abnimmt:
   $$\text{Wenn } \mathcal{L}_{\text{val}}^{(e)} < \mathcal{L}_{\text{best}} \text{ dann } \theta_{\text{best}} = \theta_{\text{dec}} \tag{3.9}$$

Implementierungsdetails Die Implementierung wird mit GPU-Unterstützung vorgenommen, da das Training ansonsten Wochen in Anspruch nehmen würde. Weitere wichtige Implementierungsdetails sind:

- Verwendung des vortrainierten SAM-Modells; Nutzung des Checkpoints `vit_h`.
- Vorbereitung der Eingabedaten entsprechend den Modellanforderungen.
- Fixierung des Bildencoders ϕ_{enc} und des Prompt-Encoders ϕ_{prompt}; nur der Mask-Decoder ϕ_{dec} wird trainiert.
- Normalisierung und Skalierung der vorhergesagten Masken und Ground-Truth-Masken vor der Berechnung des Loss.

Ergebnis der Feinabstimmung Durch die Feinabstimmung des Mask-Decoders auf den erstellten Datensatz wird eine Verbesserung der Segmentierungsgenauigkeit für Baumkronen in Satellitenbildern erreicht. Durch das beschriebene Vorgehen werden präzisere Masken vom Mask-Decoder erstellt. Insbesondere bei eng beieinander stehenden Bäumen verbessert sich die Segmentierung, da die Form und Größe der einzelnen Bäume genauer erfasst werden.

2. Hyperparametertraining

In diesem Abschnitt wird die Methodik zur Optimierung der Hyperparameter des SAM mithilfe der Optuna-Bibliothek erläutert. Ziel der Optimierung ist es, das Basismodell in die Lage zu versetzen, auch ohne Benutzereingaben solide Segmentierungsergebnisse innerhalb von Satellitenbildern zu erstellen. Diese Funktionalität ergibt sich erneut aus der Anforderung, dass die Ergebnisse des Basismodells mit denen des Spezialmodells vergleichbar sein müssen und dass das Spezialmodell keine Benutzereingaben benötigt. Durch die Verwendung der Hyperparameter werden faktisch Benutzereingaben vorgetäuscht, wie z.B. die Anzahl der Punkte pro Satellitenbild (gleichmäßig über das Bild verteilt), um Punkteingaben zur Segmenterkennung zu simulieren.

Durch die systematische Optimierung der Parameter wird die Genauigkeit der Segmentierungen erhöht und die Fehlerrate reduziert. Da das SAM-Modell eine Vielzahl von konfigurierbaren Parametern besitzt, ist eine automatisierte Hyperparameteroptimierung sinnvoll, um den optimalen Satz von Parametern für die spezifische Aufgabe zu finden.

Methodik Es wird die Optuna-Bibliothek zur automatischen Hyperparameteroptimierung genutzt. Die Optimierung der Hyperparameter kann als Minimierungsproblem betrachtet werden, in der es darum geht, die Verlustfunktion möglichst niedrig zu halten. Die einzelnen Hyperparameter des `GeoSAM`[60] dienen dabei als freie Variablen. GeoSAM ist ein Wrapper des ursprünglichen SAM und erweitert es um die zu trainierenden Hyperparameter sowie um einige Pflegeoperationen, wie die Überführung in GeoJSON-Datein oder in Shape-Dateien. Es wird im Rahmen dieser Implementierung genutzt, um die Benutzereingaben zu simulieren. Hierbei geht es um die Optimierung der Hyperparameter in Bezug auf die Erkennung und Zählung von Baumkronen. Andere Segmente werden hierbei ignoriert, da das Modell nach wie vor eine semantische Segmentierung durchführt.

Definition der Verlustfunktionen Es werden zwei Verlustfunktionen zur Optimierung genutzt: die euklidische Distanz und den Dice-Loss, welche in einer kombinierten Verlustfunktion zusammengefasst werden. Ziel ist es, sowohl eine optimale Erfassung der Baumkronen zu unterstützen, als auch die richtige Anzahl an vorhandenen Baumkronen vorherzusagen.

Euklidische Distanz Die euklidische Distanz[6] $\mathcal{L}_{\text{ED}}$ misst die Distanz zwischen den vorhergesagten Punktkoordinaten $\mathbf{p}_{\text{pred}}$ und den Zielkoordinaten $\mathbf{p}_{\text{true}}$:

$$\mathcal{L}_{\text{ED}}(\mathbf{p}_{\text{pred}}, \mathbf{p}_{\text{true}}) = \|\mathbf{p}_{\text{pred}} - \mathbf{p}_{\text{true}}\|_2 \tag{3.10}$$

Dice-Loss Der Dice-Loss[47] $\mathcal{L}_{\text{Dice}}$ quantifiziert die Ähnlichkeit zwischen der vorhergesagten Segmentierung $\mathbf{Seg}_{\text{pred}}$ und der Ground-Truth-Segmentierung $\mathbf{Seg}_{\text{true}}$:

$$\mathcal{L}_{\text{Dice}}(\mathbf{Seg}_{\text{pred}}, \mathbf{Seg}_{\text{true}}) = 1 - \frac{2 \cdot |\mathbf{Seg}_{\text{pred}} \cap \mathbf{Seg}_{\text{true}}| + \varepsilon}{|\mathbf{Seg}_{\text{pred}}| + |\mathbf{Seg}_{\text{true}}| + \varepsilon} \tag{3.11}$$

wobei ε ein kleiner Wert zur Vermeidung von Division durch Null ist.

Kombinierte Verlustfunktion Die kombinierte Verlustfunktion $\mathcal{L}_{\text{combined}}$ berücksichtigt sowohl die Segmentierungsqualität als auch die Positionsgenauigkeit und fügt einen Strafterm für eine falsche Anzahl von Segmenten hinzu:

$$\mathcal{L}_{\text{combined}} = \frac{1}{\max(N_{\text{true}}, 1)} \left(\sum_{i=1}^{N_{\text{match}}} \left(\mathcal{L}_{\text{Dice}}^{i} + \mathcal{L}_{\text{ED}}^{i} \right) + |N_{\text{pred}} - N_{\text{true}}| \cdot \lambda \right) \tag{3.12}$$

wobei:

- N_{pred} die Anzahl der vorhergesagten Segmente,
- N_{true} die Anzahl der Ground-Truth-Segmente,
- N_{match} die Anzahl der zugeordneten Segmente,
- λ die Strafkonstante für Koordinatenabweichungen.

Hyperparameterraum Die zu optimierenden Hyperparameter sind:

- `points_per_side` $\in [20, 80]$: Anzahl der Punkte pro Satellitenbild.
- `pred_iou_thresh` $\in [0{,}7, 0{,}95]$: Schwellenwert für den vorhergesagten IoU
- `stability_score_thresh` $\in [0{,}85, 0{,}95]$: Schwellenwert für den Stabilitätsscore
- `box_nms_thresh` $\in [0{,}5, 0{,}9]$: NMS-Schwellenwert für Bounding-Boxen
- `crop_n_layers` $\in [0, 3]$: Anzahl der Zuschnittsebenen
- `crop_nms_thresh` $\in [0{,}5, 0{,}9]$: NMS-Schwellenwert für Zuschnitte
- `crop_overlap_ratio` $\in [0{,}25, 0{,}75]$: Überlappungsverhältnis beim Zuschnitt
- `crop_n_points_downscale_factor` $\in [1, 2]$: Skalierungsfaktor für Punkte
- `min_mask_region_area` $\in [1, 700]$: Minimale Maskenfläche

Optimierungsverfahren Das Optimierungsproblem wird wie folgt formuliert:

$$\min_{\boldsymbol{\theta}} \ \mathcal{L}_{\text{combined}}(\boldsymbol{\theta}) \tag{3.13}$$

wobei $\boldsymbol{\theta}$ den Vektor der Hyperparameter darstellt.

Die Optuna-Bibliothek nutzt folgendes Vorgehen:

1. **Initialisierung**: Setzt die Hyperparameter initial auf einen zufälligen Wert.
2. **Modellgenerierung**: Erstellt das SAM-Modell mit den aktuellen Hyperparametern.
3. **Segmentierung**: Wendet das Modell auf ein Eingabebild an, um Segmente zu generieren. Hier wird immer dasselbe Bild genutzt, da sonst weitere Schwankungen der Parameter zu erwarten sind.
4. **Vorverarbeitung**:
    - Filtert die generierten Segmente, um irrelevante oder fehlerhafte Ergebnisse zu entfernen. Hierbei werden alle Segmente ignoriert und herausgefiltert, die nicht von Interesse für die spätere Segmentierung von Baumkronen sind.

- Bereitet die Ground-Truth-Daten durch Laden der entsprechenden Masken und Koordinaten auf. Die Zuordnung der Ground-Truth Segmente zu den erstellten Segmenten erfolgt über die höchstmögliche IOU, sowie die minimale euklidische Distanz zwischen den Zentrumskoordinaten des Katasteramts und dem Zentrum der vorhergesagten Segmente. Dabei wird angenommen, dass bei einer Überlappung von 1.0 die Baumkrone (wenn auch nicht als Baumkrone erkannt) segmentiert wurde.

5. **Verlustberechnung**: Berechnet die kombinierte Verlustfunktion $\mathcal{L}_{\text{combined}}$, indem die vorhergesagten Segmente den Ground-Truth-Segmenten mittels des Algorithmus (*linear sum assignment*, siehe Gleichung.: 3.12) zugeordnet werden.

6. **Optimierung**: Übergibt die Ergebnisse der Verlustfunktion an Optuna, um die Hyperparameter für den nächsten Durchlauf anzupassen.

Implementierungsdetails

- **Modell**: Es wird eine angepasste Version von SAM (`CustomSamGeo`) genutzt, welche die angegebenen Hyperparameter unterstützt und alle notwendigen Pflegeoperationen beinhaltet. Diese Klasse erleichtert den Umgang mit `sam.masks` innerhalb des GeoSAM.

- **Segmentierung**: Die Segmentierung wird auf dem Satellitenbild `output_tile_1.tif` durchgeführt. Dieses ist zufällig gewählt und dient als Trainingsgrundlage für Optuna, da das Durchwechseln verschiedener GeoTIFF-Dateien einen weiteren starken Einfluss auf das Optimierungsproblem hat. Dieser Einfluss ist bewusst weggelassen worden.

- **Loss-Funktion**: Die kombinierte Verlustfunktion ist in PyTorch implementiert und berücksichtigt sowohl die Segmentierungsqualität als auch die Positionsgenauigkeit.

- **Zuordnung der Segmente**: Die Zuordnung zwischen vorhergesagten und Ground-Truth-Segmenten erfolgt anhand der minimalen euklidischen Distanz der Punktkoordinaten des Katasteramtes und der höchsten IOU.

- **Hyperparameteroptimierung**: Optuna führt 100 Optimierungsschritte (Trials) durch, um die optimalen Hyperparameter zu finden.

- **Fehlerbehandlung**: Bei Auftreten eines Fehlers im Optimierungsprozess wird ein solch hoher Verlust zurückgegeben, dass die genutzte Auswahl den Suchraum wieder deutlich anpasst.
- **Speicherverwaltung**: Nach jedem Trial werden Speicherbereinigungen durchgeführt, um GPU-Speicher freizugeben und Speicherlecks zu vermeiden.

Ergebnis der Optimierung der Hyperparameter von SAM Durch die Optimierung der Hyperparameter des SAM (angepasste Version `GeoSam`[60]) wird das Basismodell in die Lage versetzt, auch ohne Benutzereingaben Segmente innerhalb des Satellitenbildes zu erstellen. Es ist bisher nicht möglich, Bäume von anderen Segmenten zu unterscheiden; es handelt sich weiterhin um eine semantische Segmentierung.

3.4.2 Feinabstimmung des CLIP-Modells zur Unterscheidung von Baum und Kein Baum

Dieser Abschnitt beschreibt die Methodik zur Feinabstimmung des CLIP-Modells, um zwischen Segmenten zu unterscheiden. Unter zu Hilfenahme des CLIP Modells kann das Basismodell Einzelinstanzen segmentieren und *Baum* von *kein Baum* unterscheiden. Hierzu werden dieselben Bilder des Datensatzes verwendet, wie in Abschnitt 3.3.2 beschrieben sind. Ein Unterschied besteht in den verwendeten Ground-Truth-Daten. Diese müssen *Baum* und *kein Baum* trennen. Folgend wird das Vorgehen genau beschrieben. Innerhalb des modularen Aufbaus des Systems liegt diese Implementierung innerhalb der Komponente CLIP-Training (siehe Abb. 3.1). Nach Abschluss der Feinabstimmung dieses Modells wird die Anforderung an die Vergleichbarkeit der untersuchten Modelle gegeben sein, da beide Modelle anschließend eine Instanzsegmentierung von Baumkronen durchführen.

Datensatzaufbereitung

Aus den GeoTIFF-Bildern des Datensatzes D werden nun Bildausschnitte herausgeschnitten, die entweder Bäume oder keine Bäume sind. Hierbei geht es nur um Bildausschnitte, welche nicht schwarz sind und somit außerhalb des betrachteten Straßennetzes liegen. Die vorhandenen Ground-Truth-Daten im GeoJSON-Format werden in diesem

Schritt genutzt, um die Position von Bäumen zu bestimmen. Für jeden Baum wird ein entsprechender Bildausschnitt erstellt:

$$\text{Baum-Bildausschnitt}_i = I_{\text{TIFF}}\left[x_i : x_i + w_i,\ y_i : y_i + h_i\right] \tag{3.14}$$

wobei I_{GeoTIFF} das ursprüngliche GeoTIFF-Bild ist und (x_i, y_i, w_i, h_i) die Koordinaten und Dimensionen der Bounding-Box des i-ten Baumes darstellen.

Für die Bereiche, welche keine Bäume sind, werden die Bounding-Boxes zufällig innerhalb der Bildbereiche verschoben, bis diese *keine Bäume* enthalten. Diese werden anschließend als *kein Baum* Datum des Datensatzes verwendet:

$$\text{Nicht-Baum-Bildausschnitt}_j = I_{\text{TIFF}}\left[x'_j : x'_j + w'_j,\ y'_j : y'_j + h'_j\right] \tag{3.15}$$

wobei (x'_j, y'_j, w'_j, h'_j) die Koordinaten und Dimensionen der zufällig platzierten Bounding-Boxes sind, die keine Überlappung mit den Ground-Truth-Bäumen besitzen. Hierbei ist es möglich, beliebig viele *kein Baum* Ausschnitte zu erstellen. Um einen ausgeglichenen Datensatz zu erhalten, werden hier so viele *Kein Baum* Ausschnitte erstellt, wie es *Baum* Ausschnitte nach Anwendung der Augmentations-Pipeline gibt (`target_tree_count` = 20000).

Datenaugmentation

Um die Größe des Trainingsdatensatzes zu erhöhen und damit die Generalisierungsfähigkeit des Modells zu verbessern, werden verschiedene Datenaugmentationsmethoden auf die vorhandenen Bildausschnitte angewendet. Diese können wie folgt beschrieben werden:

Rotation Die Rotation eines Bildes I um einen Winkel θ erfolgt durch eine affine Transformation[6]:

$$I_{\text{rot}}(x, y) = I\left(x\cos\theta - y\sin\theta,\ x\sin\theta + y\cos\theta\right) \tag{3.16}$$

wobei I_{rot} das rotierte Bild ist.

Spiegelung Die Spiegelung eines Bildes erfolgt entlang einer Achse[6]:

- **Horizontale Spiegelung** (an der vertikalen Achse):

$$I_{\text{horiz}}(x, y) = I(-x, y) \tag{3.17}$$

- **Vertikale Spiegelung** (an der horizontalen Achse):

$$I_{\text{vert}}(x, y) = I(x, -y) \tag{3.18}$$

Helligkeits- und Kontrastanpassung Die Anpassung der Helligkeit und des Kontrasts eines Bildes erfolgt durch Gamma-Korrektur:

$$I_{\text{adj}}(x, y) = \left(\frac{I(x, y)}{255}\right)^{\gamma} \times 255 \tag{3.19}$$

wobei γ der Gamma-Wert ist. Ein $\gamma < 1$ erhöht die Helligkeit, während $\gamma > 1$ die Helligkeit verringert.

Hinzufügen von Rauschen Dem Bild I wird Rauschen hinzugefügt, um die Robustheit des Modells gegenüber Störungen zu erhöhen[40]:

- **Gaussianisches Rauschen**:

$$I_{\text{noise}}(x, y) = I(x, y) + n(x, y) \tag{3.20}$$

  wobei $n(x, y) \sim \mathcal{N}(0, \sigma^2)$ ein Gaußsches Rauschen mit Varianz σ^2 ist.

- **Salt-and-Pepper-Rauschen**:

  Bei einer Wahrscheinlichkeit p wird ein Pixelwert zufällig auf 0 (schwarz) oder 255 (weiß) gesetzt:

$$I_{\text{noise}}(x,y) = \begin{cases} 0, & \text{mit Wahrscheinlichkeit } \frac{p}{2} \\ 255, & \text{mit Wahrscheinlichkeit } \frac{p}{2} \\ I(x,y), & \text{sonst} \end{cases} \tag{3.21}$$

Datenaugmentationspipeline Für jeden Originalbildausschnitt I_{orig} wird eine Menge augmentierter Bilder $\{I_{\text{aug}}^k\}$ durch Anwendung der Augmentationsfunktionen A_k erstellt:

$$I_{\text{aug}}^k = A_k(I_{\text{orig}}) \tag{3.22}$$

wobei A_k eine der oben beschriebenen Augmentationstechniken ist.

Datensatzaufteilung

Der Datensatz wird in Trainings-, Validierungs- und Testmengen aufgeteilt, wobei ein Verhältnis von 80% Training, 10% Validierung und 10% Testen genutzt wird:

$$\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}} \cup \mathcal{D}_{\text{test}} \tag{3.23}$$

Modell Training

Das vortrainierte CLIP-Modell wird mit der Vision Transformer (ViT)-Basiskonfiguration genutzt(`openai/clip-vit-base-patch32`). Das Modell besteht aus einem Bild Encoder ϕ_{img} und einem Text Encoder ϕ_{text}. Dies sind nicht die Encoder des SAM. Das genaue Verhalten des CLIP-Modells ist in Abschnitt 2.3.3 beschrieben.

Feinabstimmungsprozess

Der Feinabstimmungsprozess verläuft wie folgt:

1. **Datenvorbereitung**: Laden der Bild-Label-Paare, wobei die Titel entweder *Baum* oder *Kein Baum* sind.

2. **Tokenisierung**: Verarbeitung der Texteingaben mittels des CLIP-Tokenizers:

$$T_n = \phi_{\text{tokenizer}}(\text{Titel}_n) \tag{3.24}$$

3. **Bildvorverarbeitung**: Normalisierung und Anpassung der Bildgrößen entsprechend den Anforderungen des CLIP-Bild Encoders.

4. **Feature-Extraktion**: Berechnung der Bild- und Text Embeddings:

$$E_n^{\text{img}} = \phi_{\text{img}}(I_n) \tag{3.25}$$
$$E_n^{\text{text}} = \phi_{\text{text}}(T_n) \tag{3.26}$$

5. **Berechnung der Logits**: Ermittlung der Ähnlichkeitswerte zwischen Bild- und Text Embeddings:

$$\text{Logits}_{\text{Bild}} = E_n^{\text{img}} \cdot (E_n^{\text{text}})^\top \tag{3.27}$$
$$\text{Logits}_{\text{Text}} = E_n^{\text{text}} \cdot (E_n^{\text{img}})^\top \tag{3.28}$$

6. **Loss-Funktion**: Verwendung des Kreuzentropie-Losses für die Bild- und Text-Logits:

$$\mathcal{L} = \frac{1}{2} \left(\mathcal{L}_{\text{CE}}(\text{Logits}_{\text{Bild}}, y) + \mathcal{L}_{\text{CE}}(\text{Logits}_{\text{Text}}, y) \right) \tag{3.29}$$

   wobei y die korrekten Klassenzuordnungen sind.

7. **Optimierung**: Aktualisierung der Modellparameter durch Backpropagation und Optimierung mit dem Adam-Optimizer.

Trainingsprozess

Das Training wird über 100 Epochen durchgeführt. Die Lernrate wird auf $\eta = 5 \times 10^{-5}$ gesetzt. Der Adam-Optimizer wird mit den Parametern $\beta_1 = 0{,}9$, $\beta_2 = 0{,}98$, $\epsilon = 1 \times 10^{-6}$ verwendet und einen Gewichtungszerfall von $\lambda = 0{,}2$.

Implementierungsdetails

- **Bibliotheken**: Verwendung von PyTorch für die Implementierung des Trainingsprozesses.
- **Datenverwaltung**: Die Bildpfade und zugehörigen Labels werden in JSON-Dateien gespeichert und zur Erstellung von benutzerdefinierten Datensätzen verwendet (`Dataset`-Klasse von PyTorch).
- **Datenlader**: Implementierung einer eigenen `Dataset`-Klasse (`ImageTitleDataset`), um die Bilder und Labels effizient zu laden.
- **Fehlerbehandlung**: Implementierung von Überprüfungen, um NaN- oder Inf-Werte während des Trainings zu erkennen und zu behandeln.
- **Speicherung des Modells**: Nach Abschluss des Trainings werden die Modellgewichte gespeichert, um sie für die spätere Anwendung zu verwenden. Hierbei ist es wichtig, dass nur bei einer Verbesserung des Modells das Ergebnis gespeichert wird. Die gespeicherten Modellgewichte werden als Checkpoint-Datei `custom-clip-weights.pth` gespeichert.

Ergebnis

Durch das beschriebene Vorgehen zur Feinabstimmung des CLIP-Modells auf einen spezifischen Datensatz ist dieses in der Lage, zwischen `Baum` und `kein Baum` innerhalb von urbanen Wäldern zu unterscheiden.

Auf Abbildung A.1 ist zu sehen, dass die Unterscheidung von *Baum* und *kein Baum* Segmenten gut funktioniert. Es bestehen nach wie vor Unsicherheiten, gerade bei Schatten und andersfarbigen Blätterdächern. Dennoch lassen die Ergebnisse den Schluss zu, dass das Modell in der Lage ist, grundsätzlich die Unterscheidungen in angemessener Art und

Weise zu unternehmen. Somit ermöglicht die Feinabstimmung und Integration von CLIP in SAM eine Instanzsegmentierung durchzuführen.

3.5 Testen und Validierung

Es wurden mehrere Teststufen durchgeführt, um die Korrektheit und Leistung des Systems zu verifizieren sowie zu validieren. Wichtig hierbei ist, dass die Anforderung einer Implementierung in Google Colab Notebooks für ein umfassendes Testframework und eine automatisierte Testumgebung nur begrenzt umsetzbar ist. Angesichts dessen wurde eine einheitliche und semi-automatisierte Testumgebung erdacht, die zum Ziel hat, Tests in verschiedenen Komponenten (Notebooks) auf eine sehr vergleichbare und verständliche Art und Weise zu implementieren.

3.5.1 Unit-Tests

Unit-Tests wurden innerhalb jeder Komponente durchgeführt. Der Aufbau der Komponenten ist hierfür nach einem vergleichbaren Schema aufgebaut worden. 1. Die Implementierung der genutzten Funktionen. 2. Die Anwendung der Funktionen auf testbare Artefakte. 3. Die Sicherstellung der Funktionalität der getesteten Funktionen durch Überprüfung der Testartefakte. 4. Anwendung der implementierten Funktionen auf die tatsächlichen Artefakte.

Durch den beschriebenen Aufbau wurde sichergestellt, dass die grundlegenden Funktionen des Systems überprüft wurden und ihre Korrektheit sichergestellt ist.

3.5.2 Integrationstests

Integrationstests wurden erdacht, um sicherzustellen, dass die Ergebnisse einer Komponente mit den erwarteten Eingaben der darauffolgenden Komponente übereinstimmen. Hierfür wurden Funktionen implementiert, welche die richtige Form und Struktur initial überprüfen und beim Nichtzutreffen den Nutzer hierüber informieren. Durch die Implementierung dieser Funktionen sollte ein reibungsloser und korrekter Ablauf komponentenübergreifend sichergestellt werden.

3.5.3 Systemtests

Das implementierte Framework (System) wurde auf unterschiedlichen geographischen Ausdehnungen getestet. Es wurden verschiedene Anwendungen durchgeführt und die Ergebnisse auf ihre Korrektheit überprüft. Durch die Durchführung dieser Systemtests in unterschiedlichen, jedoch realen Umgebungen wurde die Validität des Systems sichergestellt.

3.6 Fachliches Design der Experimente mit Bezug auf die Hypothesen

Das fachliche Design der Experimente wurde dazu entwickelt, um die Hypothesen dieser Arbeit zu überprüfen. Die Experimente konzentrieren sich auf den Vergleich der Modelle SAM und DeepForest, um deren Fähigkeit zur Segmentierung von Bäumen in urbanen Umgebungen zu validieren. Diese Fähigkeiten sind der Anforderungsanalyse zu entnehmen.

3.6.1 Design der Experimente

Zur Validierung der Hypothesen wurden Experimente unter verschiedenen Bedingungen durchgeführt, die unterschiedliche Schwierigkeitsgrade der Segmentierungsaufgabe widerspiegeln. Im modularen Aufbau des Systems befindet sich diese Implementierung innerhalb der Komponente Validierung und Evaluierung (siehe Abb.: 3.1).

Ziel der Experimente

Das Ziel der durchgeführten Experimente besteht darin, die Genauigkeit und Präzision der beiden untersuchten Modelle in einer spezifisch definierten Domäne aufzuzeigen. Hierbei werden unterschiedliche Floren, Städte und Distanzen zwischen Baumkronen berücksichtigt. Mit der schrittweisen Erhöhung der Segmentierungsschwierigkeit wird untersucht, wie gut die Modelle mit komplexeren Szenarien umgehen können.

Datengrundlage

Vorhersagedaten Die Modelle produzieren GeoJSON-Dateien, welche die Vorhersagen beinhalten. Sie sind im Ordner `geojson-pred` gespeichert. Jede Vorhersagedatei enthält die vorhergesagten Baumsegmente für ein bestimmtes Gebiet und ist entsprechend dem verwendeten Modell benannt:

- **SAM_CLIP**: Dateien mit dem Namensmuster `{Nummer}_SAM_CLIP.geojson`
- **DeepForest**: Dateien mit dem Namensmuster `{Nummer}_DeepForest.geojson`

Die Nummern identifizieren das jeweilige Gebiet.

Ground-Truth-Daten Die Ground-Truth-Daten bestehen aus manuell annotierten Baumsegmenten und Korridorgeometrien (Polygonen) und befinden sich im Ordner `geojson-ground-truth`. Für jedes Testgebiet liegen folgende Dateien vor:

- **Baumsegmente**: `{Nummer}_ground_truth.geojson`
- **Korridore**: `{Nummer}_korridor_{Typ}.geojson`, wobei `{Typ}` der Korridortyp angibt:
  - `nicht_eng`: klar trennbare Baumkronen
  - `eng`: eng stehende Baumkronen
  - `ineinander`: verschmelzende Baumkronen

Dies ist notwendig, um eine gesonderte Untersuchung der unterschiedlichen Schwierigkeitsgrade innerhalb der Segmentierungsaufgaben zu ermöglichen.

Testdesign mit steigender Schwierigkeit

Die Tests sind so gestaltet, dass sie die Schwierigkeit der Segmentierungsaufgabe von einfach bis schwer erhöhen:

1. **Klar trennbare Bäume (`nicht_eng`)**: Bäume sind deutlich voneinander getrennt, und ihre Kronen überlappen nicht. Dies stellt die einfachste Segmentierungsaufgabe dar.

2. **Eng stehende Bäume (`eng`)**: Bäume stehen dicht beieinander, ihre Kronen können sich berühren, ohne jedoch vollständig zu verschmelzen. Dies erhöht die Komplexität der Segmentierung.

3. **Verschmelzende Baumkronen (`ineinander`)**: Baumkronen überlappen erheblich und bilden komplexe Strukturen. Dies stellt die größte Herausforderung für die Modelle dar.

Abbildung 3.5: Links `nicht_eng` stehende Baumkronen und rechts `eng` und `ineinander` stehende Baumkronen

Die Abb. 3.5 zeigt die unterschiedlichen Schwierigkeitsgrade. Das linke Bild beinhaltet vollständig getrennte Baumkronen und fällt somit in Kategorie eins. Der Korridor erstreckt sich über den komplett nicht geschwärzten Bereich des Satellitenbildes. Das rechte Bild derselben Abbildung beinhaltet sowohl komplexe Strukturen von Baumkronen (siehe im linken Teil des rechten Bildes), als auch Baumkronen, welche dicht beieinander liegen, jedoch nicht verschmelzen. Dieses Trainingsbild würde somit zwei Korridordateien besitzen. Einen Korridor, welcher die Kategorie drei, also den linken Teil des rechten Bildes beinhaltet, wie auch einen Korridor, welcher die senkrechte Straße mittig auf dem Bild beinhaltet (Kategorie zwei der Schwierigkeit).

Diese Tests werden für verschiedene Floren durchgeführt, um die Modelle unter unterschiedlichen vegetativen Bedingungen zu evaluieren. Hierbei wurde sich für `Hamburg`,

`Kapstadt`, `San Francisco` und `Tokio` entschieden. Die Wahl ist aus folgenden Gründen getroffen worden:

1. **Hamburg**: Bietet eine gute Ground-Truth-Datenlage mit Daten vom Katasteramt Hamburg.
2. **Kapstadt**: Liegt in einem tropischen Klima und bietet eine gänzlich andere Flora als Hamburg.
3. **San Francisco**: Eine gemäßigte Klimazone mit anderen Baumarten als die vorherigen Städte.
4. **Tokio**: Eine weitere Region, die sich klimatisch und geographisch stark von den anderen Städten unterscheidet.

3.6.2 Evaluationsmetriken

Wahl der Metriken um ein vollständiges und umfassendes Bild der Leistungsfähigkeit der Modelle zu erhalten, wurden Metriken zur Bewertung der Modelle verwendet. Hierbei wurde sich bewusst gegen die oft genutzte Metrik der *Accuracy* entschieden, da diese die Kenntnis über alle im Bild befindlichen Objekte erfordert. Diese Information liegt nicht vor. Auch eine Konfusionsmatrix benötigt Informationen, welche in der hier beschrieben Implementierung nicht vorliegen und konnte nicht berücksichtigt werden. Dennoch ist die Kombination der folgenden Auswahl an Metriken in der Lage, ein umfassendes und quantifizierbares Ergebnis der Leistungsfähigkeiten der beiden Modelle zu erstellen.

Die folgenden Variablen werden in der Berechnung der Metriken verwendet:

- S_{TP}: Anzahl der richtig erkannten Bäume (True Positives),
- S_{FP}: Anzahl der falsch erkannten Bäume (False Positives),
- S_{FN}: Anzahl der tatsächlichen, aber nicht erkannten Bäume (False Negatives),
- $S_{\mathrm{GT\_Korridor}}$: Anzahl der Ground-Truth-Bäume im betrachteten Korridor.

True Positive in Prozent (TP%)

Diese Metrik gibt den Prozentsatz an, der die korrekt identifizierten Baumkronen widerspiegelt. Es geht hierbei um die Aufzeigung der Sicherheit einer Segmentierung der Mododelle. Die Berechnung erfolgt nach der Formel[50]:

$$\mathrm{TP\%} = \left(\frac{S_{\mathrm{TP}}}{S_{\mathrm{GT\_Korridor}}} \right) \times 100 \tag{3.30}$$

False Positives in Prozent (FP%)

Diese Metrik misst den Prozentsatz der inkorrekt identifizierten Baumkronen, die innerhalb des Korridors liegen. Hiermit soll aufgezeigt werden, wie häufig die Modelle fälschlicherweise Baumkronen identifiziert haben und infolgedessen die Sicherheit der Segmentierungen der Modelle aufzeigen. Die Berechnung erfolgt nach der Formel[50]:

$$\mathrm{FP\%} = \left(\frac{S_{\mathrm{FP}}}{S_{\mathrm{GT\_Korridor}}} \right) \times 100 \tag{3.31}$$

Precision

Die Precision wird als das Verhältnis der korrekt vorhergesagten positiven Instanzen zu allen vorhergesagten positiven Instanzen berechnet[50] sie zeigt auf, wie hoch die Genauigkeit der Modelle bei der Vorhersage der Baumsegmente sind:

$$\mathrm{Precision} = \frac{S_{\mathrm{TP}}}{S_{\mathrm{TP}} + S_{\mathrm{FP}}} \tag{3.32}$$

Recall

Durch die gewählte Metrik wird aufgezeigt, wie die Fähigkeit der Modelle ist, alle Baumsegmente zu erkennen und wie viele übersehen werden. Der Recall wird somit als das Verhältnis der korrekt vorhergesagten positiven Instanzen zu allen tatsächlichen positiven Instanzen berechnet[50]:

$$\text{Recall} = \frac{S_{\text{TP}}}{S_{\text{TP}} + S_{\text{FN}}} \tag{3.33}$$

F1-Score

Der F1-Score ist das harmonische Mittel von Precision und Recall[50]. Hierbei geht es um das optimale Gleichgewicht (Kompromiss) zwischen Precision und Recall, da beide Werte für die Messung der Leistungsfähigkeit relevant sind:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.34}$$

Receiver Operating Characteristic (ROC) Kurve und Area Under the Curve (AUC)

Die ROC-Kurve stellt die True Positive Rate (TPR) gegen die False Positive Rate (FPR) bei verschiedenen Schwellenwerten dar. Durch die Berechnung kann dargestellt werden, wie gut die Modelle zwischen positiven und negativen Klassen unterscheiden können. Die Berechnung erfolgt mit folgenden Formeln[50]:

- **True Positive Rate (TPR)**:

$$\text{TPR} = \frac{S_{\text{TP}}}{S_{\text{TP}} + S_{\text{FN}}} \tag{3.35}$$

- **False Positive Rate (FPR)**:

$$\text{FPR} = \frac{S_{\text{FP}}}{S_{\text{FP}} + S_{\text{TN}}} \tag{3.36}$$

3.6.3 Evaluationsverfahren

Das Evaluationsverfahren umfasst mehrere Schritte, um eine systematische Bewertung der Modelle zu erhalten.

Datenvorbereitung

1. **Bestimmung des Modelltyps**: Anhand des Dateinamens wird ermittelt, von welchem Modell die Vorhersage stammt. Entweder vom Basismodell SAM_CLIP oder vom spezialisierten Modell DeepForest.
2. **Zuordnung der Ground-Truth- und Korridordateien**: Jeder Vorhersagedatei wird die zugehörige Ground-Truth-Datei und Korridordatei zugeordnet.
3. **Laden der GeoJSON-Dateien**: Die Vorhersage-, Ground-Truth- und Korridordaten werden mithilfe als GeoDataFrames geladen.
4. **Validierung der Geometrien**: Alle Geometrien werden auf Gültigkeit geprüft und notfalls repariert.

Evaluierung der Vorhersagen

Die Vorhersagen werden anhand ihrer Übereinstimmung mit den Ground-Truth-Daten und der Position innerhalb des Korridors bewertet.

Bestimmung der korrekt identifizierten Bäume (True Positives) Ein Baum wird als richtig identifiziert (True Positive), wenn das vorhergesagte Segment eine Überlappung von mindestens 50% mit einem Ground-Truth-Segment hat und zu mindestens 50% innerhalb des jeweiligen Korridors liegt.

Bestimmung der falsch identifizierten Bäume (False Positives) Ein Baum wird als falsch identifiziert (False Positive), wenn das vorhergesagte Segment zu mindestens 50% im Korridor liegt und weniger als 50% mit einem Ground-Truth-Segment überlappt.

Bestimmung der falsch negativen Bäume (False Negatives) Ein Ground-Truth-Segment wird als falsch negativ identifiziert (False Negative), wenn es zu mindestens 50% im Korridor liegt und von keinem vorhergesagten Segment ausreichend überlappt wird.

Berechnung der Metriken

Anhand der ermittelten Werte S_{TP}, S_{FP} und S_{FN} werden die Metriken TP% und FP%, Precision, Recall, F1-Score sowie die TRP und die FPR berechnet.

3.6.4 Visualisierung der Ergebnisse

Die Ergebnisse werden mithilfe unterschiedlicher Diagrammtypen dargestellt:

- **Boxplots**: Darstellung der Verteilungen von TP%, FP% für die unterschiedlichen Korridortypen und Modelle bei einer Vorhersagewahrscheinlichkeit von 50%.
- **Precision-Recall-Kurven**: Visualisierung des Zusammenspiels zwischen Precision und Recall, einschließlich des F1-Scores.
- **ROC-Kurven**: Vergleich der Modelle hinsichtlich ihrer True Positive und False Positive Rates bei verschiedenen Schwellenwerten und zeigt den AUC-Wert an, um die Fähigkeit der klaren Trennung der Klassen aufzuzeigen.

3.6.5 Implementierungsdetails

Die Implementierung des Evaluationsprozesses wurde in Python mit folgenden Bibliotheken umgesetzt:

- **GeoPandas**: Verarbeitung von Geodaten und geometrischen Operationen.
- **Shapely**: Manipulation und Analyse von geometrischen Objekten.
- **Matplotlib**: Zur Erstellung der Diagramme.

Zusätzlich wurden folgende Techniken verwendet:

- **Automatisierte Dateiauswahl**: Die Funktionen `determine_model_type`, `get_ground_truth_file` und `get_korridor_files` ermöglichen die Zuordnung der entsprechenden Dateien.
- **Geometrievalidierung**: Ungültige Geometrien werden durch Anwendung eines Null-Puffers (`buffer(0)`) korrigiert.

- **Effiziente Datenverarbeitung**: Iterationen über Geometrien werden mithilfe der besprochenen Bibliotheken optimiert.
- **Score-Filterung**: Bei Bedarf können die vorhergesagten Segmente anhand eines minimalen Score-Schwellenwerts gefiltert werden. Dies erleichtert das Testen der Funktionalität der Methoden und Auswertungen aus den GEOJSON-Dateien.

3.6.6 Erwartetes Ergebnis

Durch das beschriebene Testdesign sollen die Unterschiede der Leistungsfähigkeit beider Modelle visuell dargestellt werden. Ziel ist die Validierung der Genauigkeiten und der Modellleistungen im direkten Vergleich des fein abgestimmten Basismodells und des Spezialmodells in unterschiedlich schweren Kategorien in verschiedenen Floren.

3.7 Zusammenfassung der Implementierung

Das Kapitel *Implementierung* beschreibt den Aufbau eines Frameworks, welches auf der Anforderungsanalyse basierende Funktionalitäten aufweist. Es wurden das Design, die genaue Implementierung und die durchgeführten Tests der unterschiedlichen Module beschrieben. Diese wurden so implementiert, dass sie abgestimmt auf die in dieser Arbeit gestellten Hypothesen aussagekräftige Experimente durchführen können, um die Güte zweier Modelle zu vergleichen.

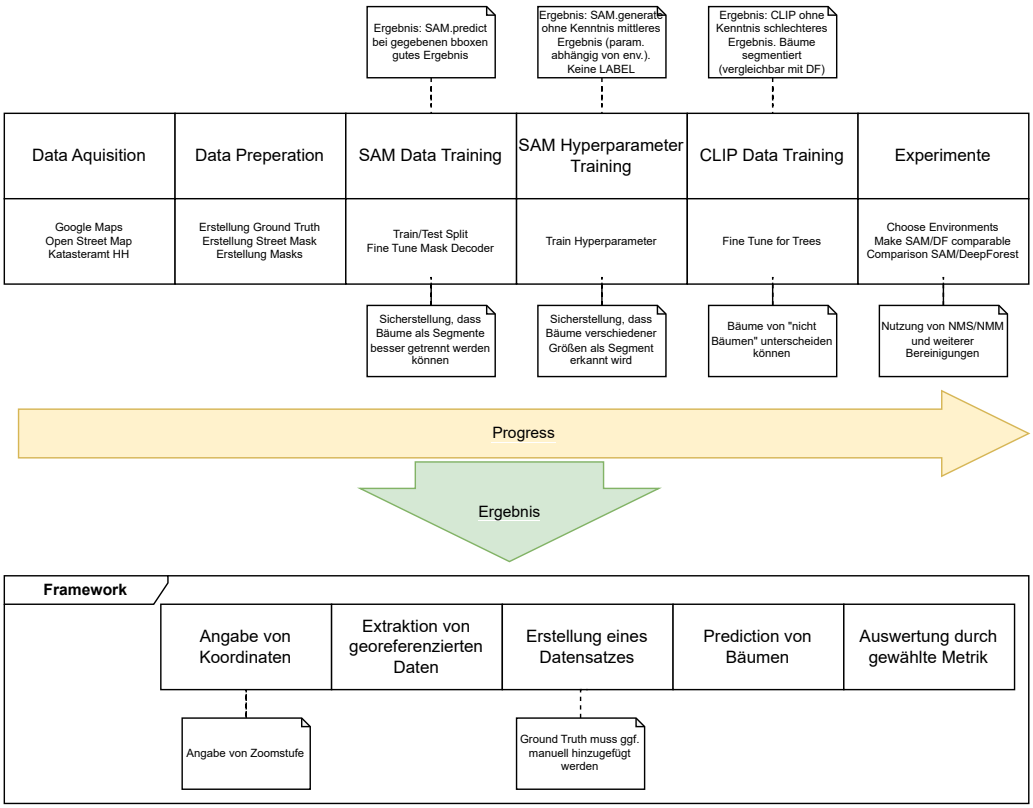

Abbildung 3.6: Überblick des implementierten Frameworks zur Beantwortung der in der Arbeit gestellten Hypothesen.

Das in Abb. 3.6 dargestellte Framework ist das Ergebnis der beschriebenen Implementierung. Es wurde so konzipiert, dass es ohne große Anpassungen für weitere Fragestellungen verwendet werden kann. Der modulare Aufbau ermöglicht den Austausch einzelner Teile des Frameworks, da die Ausgaben der jeweiligen Module durch andere, strukturell gleiche ersetzt werden können. Es erfüllt somit alle in Abschn. 3.1 beschriebenen Anforderungen.

4 Ergebnisse

Im folgenden Abschnitt werden die Ergebnisse präsentiert, welche die Leistungsfähigkeit der Modelle SAM_CLIP und DeepForest im Rahmen der Erkennung von Instanzen von Baumkronen vergleichen. Die Experimente sind in drei Segmentierungsschwierigkeitskategorien unterteilt:

- **Klar trennbare Bäume (`nicht_eng`)**: Bäume, die deutlich voneinander getrennt sind, ohne Überlappung der Baumkronen und mit einem erkennbaren Abstand zwischen diesen.
- **Eng stehende Bäume (`eng`)**: Bäume, deren Baumkronen sich leicht berühren oder die sehr nahe beieinander stehen.
- **Verschmelzende Baumkronen (`ineinander`)**: Baumkronen, die stark überlappen und schwer voneinander zu trennen sind. Bei dieser Kategorie entstehen visuell komplexe Strukturen von Baumkronen.

Die so unterteilten Experimente in Kategorien werden zudem in vier floral unterschiedlichen geographischen Ausdehnungen untersucht, um die Generalisierbarkeit der Modelle zu überprüfen:

- **Hamburg**: Mit Ground-Truth-Daten aus dem Katasteramt.
- **Kapstadt**: Tropisches Klima, deutlich abweichende Flora im Vergleich zu der Flora, welche in den Trainingsdaten enthalten waren.
- **San Francisco**: Gemäßigtes Klima, stark abweichende Baumarten. Auch diese Flora war nicht Bestandteil der Trainingsdaten.
- **Tokio**: Geographisch und vegetativ stark unterschiedlich zu den anderen Regionen. Auch diese Flora war nicht Bestandteil der Trainingsdaten.

Jedes der konzipierten Experimente zur Bestimmung der Leistungsfähigkeit der Modelle und der Beantwortung der Hypothesen wird mithilfe der zuvor beschriebenen Metriken berechnet und visualisiert:

- **Precision**: Anteil der korrekt vorhergesagten positiven Baumkronensegmente.
- **Recall**: Verhältnis der korrekt vorhergesagten Baumkronen zu allen tatsächlichen Baumkronen.
- **F1-Score**: Das harmonische Mittel von Precision und Recall.
- **ROC-AUC**: Fläche unter der ROC-Kurve, welche die Fähigkeit des Modells misst, zwischen positiven und negativen Segmenten zu unterscheiden.

Die Ergebnisse werden in Form von TP% und FP% als Boxplots (mit einem Score von 0.5), Precision-Recall-Kurven sowie ROC-Kurven dargestellt. Durch das beschriebene Vorgehen wird ein umfassendes Bild der Leistungsfähigkeit beider Modelle gewährleistet und dem Betrachter die Möglichkeit des Vergleichs auf quantitative Art und Weise ermöglicht.

4.1 Performance in Hamburg

True Positive Percentage (TP%) und False Positive Percentage (FP%) für Hamburg

Die Abbildungen 4.1 und 4.2 zeigen die Ergebnisse für die True Positive Percentage (TP%) und die False Positive Percentage (FP%) der Modelle SAM_CLIP und DeepForest in Hamburg, jeweils aufgeteilt in die Schwierigkeitsgrade `nicht_eng`, `eng` und `ineinander`. Dabei wird der Prozentsatz der korrekt erkannten Bäume (TP%) sowie der falsch erkannten Bäume (FP%) für beide Modelle in Form von Boxplots visualisiert.

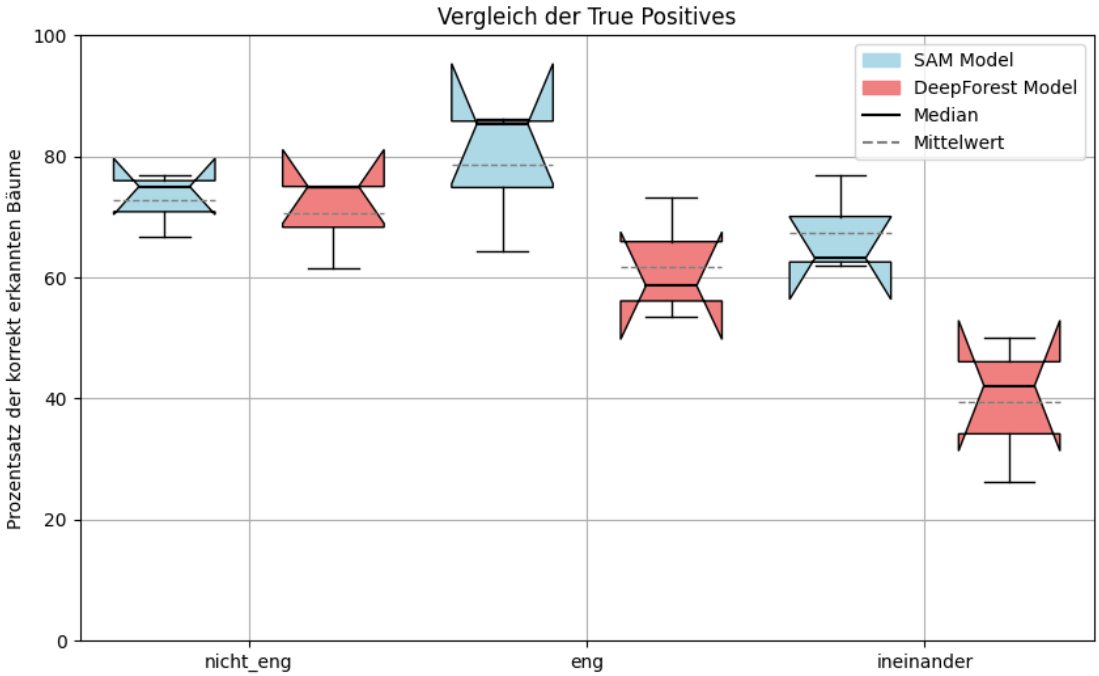

Abbildung 4.1: Boxplot der True Positive Percentage (TP%) für Hamburg in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

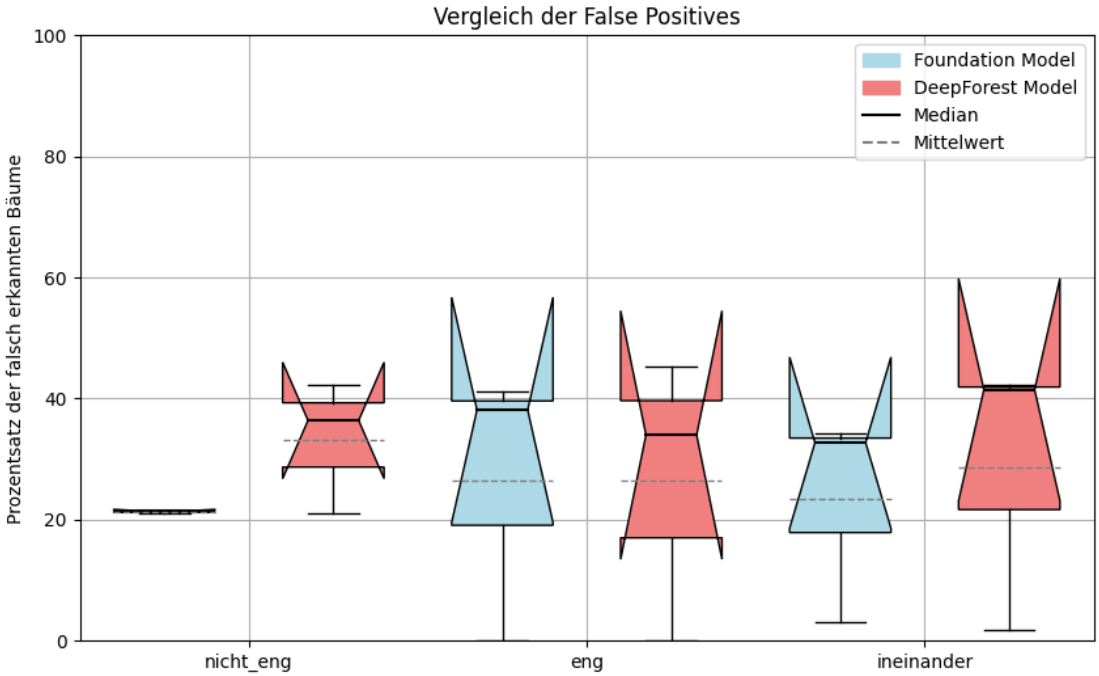

Abbildung 4.2: Boxplot der False Positive Percentage (FP%) für Hamburg in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Die Abbildungen 4.1 und 4.2 zeigen die Ergebnisse der True Positive Percentage (TP%) und der False Positive Percentage (FP%) der Modelle SAM_CLIP und DeepForest in Hamburg, aufgeteilt in die Schwierigkeitsgrade `nicht_eng`, `eng` und `ineinander`. Dabei wird der Prozentsatz der korrekt erkannten Bäume (TP%) sowie der falsch erkannten Bäume (FP%) in Form von Boxplots visualisiert.

Kategorie `nicht_eng`: Für die Kategorie `nicht_eng`, bei der die Bäume weiter auseinander stehen, zeigt das SAM_CLIP Modell bei den True Positives eine starke Performance mit einem Median von etwa 76% und einem Mittelwert von ca. 72%. Die Verteilung der Werte liegt zwischen 71% und 76%, während die Whiskers von 68% bis 77% reichen. Das DeepForest Modell erreicht einen Median von etwa 76% (identisch mit SAM) und einen Mittelwert von rund 70%. Die Quartile erstrecken sich von 69% bis 72% und entspricht damit dem Mittelwert von SAM. Die Whiskers reichen von 61% bis 72%.

Bei den False Positives schneidet das SAM_CLIP Modell ebenfalls deutlich besser ab, mit einem Median von ca. 21% und einem Mittelwert von 21%. Die Verteilung der Werte ist hierbei praktisch nicht messbar, da sie ca. 1% beträgt. Das DeepForest Modell zeigt höhere False Positive Werte mit einem Median von ca. 32% und einem Mittelwert von 35%. Die Verteilung erstreckt sich von 28% bis 39%, und die Whiskers reichen von 21% bis 42%.

Kategorie `eng`: In der Kategorie `eng`, bei der die Bäume näher beieinander stehen, erreicht das SAM_CLIP Modell bei den True Positives einen Median von etwa 70% und einen Mittelwert von ca. 76%. Die Verteilung der Werte ist gestreut, mit Quartilen von 76% bis 85% und Whiskers von 65% bis 86%. Das DeepForest Modell zeigt in dieser Kategorie eine deutlich geringere Leistung, mit einem Median von 61% und einem Mittelwert von 58%. Die Quartile liegen hier zwischen 55% und 65%, während die Whiskers von 45% bis 71% reichen.

Bei den False Positives zeigt sich in dieser Kategorie eine Verschlechterung der Performance beider Modelle, die sich durch eine breite Streuung der Ergebnisse äußert. Das SAM_CLIP Modell hat einen Median von ca. 26% und einen Mittelwert von 39%. Die Quartile reichen von 19% bis 40%, während die Whiskers von 0% bis 41% reichen. Das DeepForest Modell liegt hier mit einem Median von etwa 26% gleichauf mit dem SAM und hat mit einem Mittelwert von 35% leicht unter dem des SAM abgeschnitten. Die Werte verteilen sich von 18% bis 40%, mit Whiskers zwischen 0% und 45%.

Kategorie `ineinander`: In der schwierigsten Kategorie `ineinander`, bei der die Bäume ineinander stehen und optisch kaum zu trennen sind, sinken die Werte der True Positives bei beiden Modellen, wobei das DeepForest modell eine deutliche senkung der TP% aufzeigt. Das SAM_CLIP Modell erreicht einen Median von ca. 66% und einen Mittelwert von 64%. Die Quartile liegen zwischen 62% und 70%, mit Whiskers, die von 62% bis 75% reichen. Das DeepForest Modell zeigt eine deutlich geringere Leistung mit einem Median von ca. 39% und einem Mittelwert von rund 42%. Die Verteilung der Werte erstreckt sich von 32% bis 44%, und die Whiskers reichen von 26% bis 50%.

Bei den False Positives ist in der Kategorie `ineinander` hat das SAM_CLIP Modell einen Median von ca. 23% und einen Mittelwert von 31%. Die Quartile erstrecken sich von 18% bis 32%, und die Whiskers reichen von 3% bis 31%. Das DeepForest Modell zeigt die höchste Anzahl an False Positives mit einem Median von ca. 29% und einem

Mittelwert von 41%. Die Quartile liegen zwischen 22% und 42%, während die Whiskers von 1% bis 42% reichen.

Zusammenfassung: Die Analyse der True Positive und False Positive Raten zeigt, dass das SAM_CLIP Modell in allen Schwierigkeitsgraden insgesamt bessere True Positive Ergebnisse erzielt, während es gleichzeitig niedrigere False Positive Raten aufweist als das DeepForest Modell. Besonders in den Kategorien `nicht_eng` und *ineinander* schneidet das SAM_CLIP Modell deutlich besser ab. In der Kategorie `eng` ist das erzielte Ergebnis des SAM_CLIP Modells zwar ein wenig besser, jedoch fallen die Unterschiede hier geringer aus. Gerade in der Kategorie *ineinander* hat das DeepForest Modell Schwierigkeiten seine guten Ergebnisse zuhalten, sowohl bei den TP% – als auch bei den FP% – Werten.

Precision-Recall-Kurven

Die Precision-Recall-Kurven für Hamburg in den drei Schwierigkeitsgraden werden in Abbildung 4.3 gezeigt. Hierbei ist zu erkennen, dass SAM und DeepForest in allen Szenarien (`eng` und `ineinander`) vergleichbare Precision und Recall-Werte aufweisen. Lediglich im Bereich des Recalls von 0.0 bis 0.2 ist die Precision des DeepForest Modells deutlich höher. Richtung Recall von 1.0 gleichen sich diese Werte jedoch sehr an.

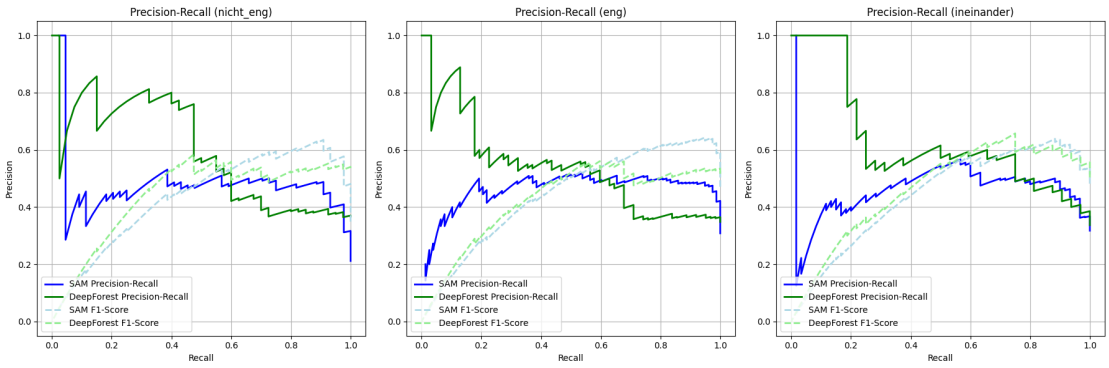

Abbildung 4.3: Precision-Recall-Kurve für Hamburg in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: In der Kategorie `nicht_eng` zeigt das SAM_CLIP Modell (blaue Linie) eine etwas bessere Leistung bei mittleren Recall-Werten. Es erreicht fast

durchgehend eine Precision von etwa 0.5, selbst bei steigendem Recall. Der F1-Score von SAM_CLIP (hellblaue gestrichelte Linie) bleibt über den gesamten Verlauf relativ stabil und erreicht Werte um 0.5 bis 0.6. Das DeepForest Modell (grüne Linie) zeigt in dieser Kategorie leicht bessere Ergebnisse bei niedrigen Recall-Werten. Der Precision-Wert beginnt stark, sinkt jedoch mit steigendem Recall und fällt unter 0.4. Der F1-Score von DeepForest (grün gestrichelte Linie) bleibt konstant bei etwa 0.5.

Kategorie `eng`: In der Kategorie `eng` bleibt die Leistung des SAM_CLIP Modells (blaue Linie) nahezu identisch. Die Precision schwankt nur leicht und fällt bei höheren Recall-Werten wieder auf ca. 0.4 ab. Der F1-Score (hellblaue gestrichelte Linie) steigt leicht an und liegt bei etwa 0.6 bis 0.62 bei höheren Recall-Werten. Auch das DeepForest Modell (grüne Linie) verändert in dieser Kategorie seine Leistung nur marginal und wird ein wenig schlechter. Die Precision bleibt bis zu einem Recall-Wert von 0.7 relativ stabil, dann sinkt er deutlich ab und liegt im Bereich von 0.35, wobei der F1-Score (grün gestrichelte Linie) deutlich unter dem des SAM Modell im Bereich von 0.5 liegt.

Kategorie `ineinander`: In der schwierigsten Kategorie `ineinander` zeigt das SAM_CLIP Modell (blaue Linie) bis zu hohen Recall-Werten von 0.85 fast keine Änderung in der Precision, welche stabil bei etwa 0.5 bleibt. Bei sehr hohen Recall-Werten sinkt die Precision dann jedoch auf 0.4. Der F1-Score (hellblaue gestrichelte Linie) bleibt im Vergleich zu den vorangegangenen Schwierigkeitskategorien sehr ähnlich und erreicht Werte um 0.65. Auch das DeepForest Modell (grüne Linie) zeigt hier vergleichbare Ergebnisse im Vergleich zu den vorherigen Schwierigkeitskategorien. Die Precision fällt ab Recall-Werten über 0.9 deutlich. Der F1-Score von DeepForest (grün gestrichelte Linie) ist ebenfalls vergleichbar und liegt bei hohen recall-Werten bei ca. 0.61.

Zusammenfassung: Die Precision-Recall-Kurven für Hamburg zeigen, dass das SAM_CLIP Modell in der Kategorie `nicht_eng` eine bessere Leistung erzielt, mit stabiler Precision und höheren F1-Scores im Vergleich zu DeepForest, das bei steigendem Recall an Precision verliert. In der Kategorie `eng` bleiben beide Modelle auf einem ähnlichen Niveau, wobei sie bei hohen Recall-Werten an Precision einbüßen. DeepForest zeigt in dieser Kategorie zwar eine etwas stabilere Precision bei niedrigen Recall-Werten, fällt jedoch beim F1-Score hinter SAM_CLIP zurück. In der schwierigsten Kategorie `ineinander` schneiden beide Modellesehr vergleichbar ab.

ROC-Kurven

Die ROC-Kurven für Hamburg in den drei Schwierigkeitsgraden werden in Abbildung 4.4 gezeigt. Die ROC-Kurven zeigen die True Positive Rate (TPR) gegen die False Positive Rate (FPR) für die Modelle SAM_CLIP und DeepForest in den Kategorien `nicht_eng`, `eng` und `ineinander`. Der AUC-Wert (Area Under the Curve) ist ebenfalls angegeben, um die Gesamtleistung zu quantifizieren.

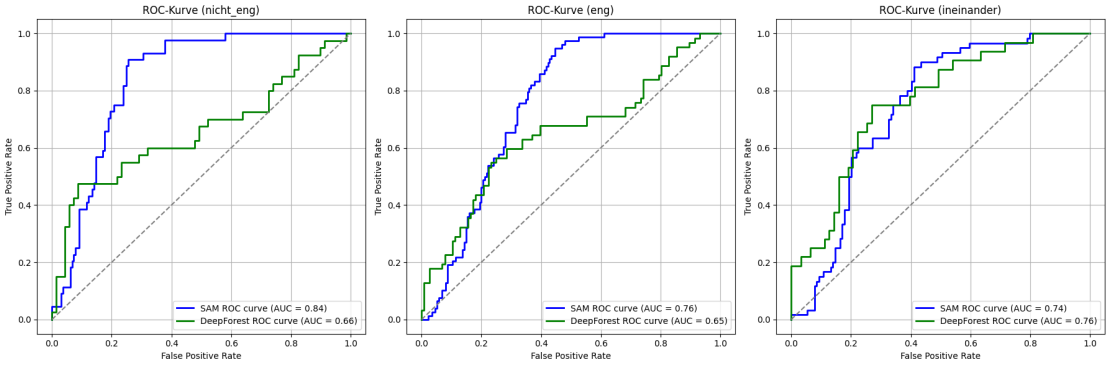

Abbildung 4.4: ROC-Kurve für Hamburg in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: In der Kategorie `nicht_eng` zeigt das SAM_CLIP Modell (blaue Linie) eine AUC von 0.84. Die ROC-Kurve zeigt eine schnelle Steigerung der True Positive Rate bei niedrigen False Positive Raten und erreicht hohe Werte. Das DeepForest Modell (grüne Linie) zeigt eine AUC von 0.66, was deutlich unter der Leistung von SAM_CLIP liegt. Die Kurve zeigt eine schwächere Trennung der Klassen bei höheren False Positive Raten.

Kategorie `eng`: In der Kategorie `eng` zeigt das SAM_CLIP Modell (blaue Linie) eine höhere Performance mit einer AUC von 0.76. Das DeepForest Modell (grüne Linie) zeigt hier eine schlechtere Leistung mit einer AUC von 0.65, was eine schwächere Trennung der Klassen bei höheren True Positive Raten anzeigt.

Kategorie `ineinander`: In der Kategorie `ineinander` zeigt das SAM_CLIP Modell (blaue Linie) eine AUC von 0.74. Die ROC-Kurve zeigt eine moderate Leistung mit einer gewissen Trennung der Klassen. Das DeepForest Modell (grüne Linie) zeigt eine

leicht höhere AUC von 0.76, was auf eine minimal bessere Klassifizierung und eine etwas stärkere Trennung der Klassen hinweist.

Zusammenfassung: Die ROC-Kurven für Hamburg zeigen, dass das SAM_CLIP Modell in der Kategorie `nicht_eng` eine deutlich bessere Performance aufweist als das DeepForest Modell, mit einer AUC von 0.84 im Vergleich zu 0.66. In der Kategorie `eng` bleibt SAM_CLIP ebenfalls überlegen, allerdings mit einem geringeren Abstand (AUC 0.76 gegenüber 0.65). In der schwierigsten Kategorie `ineinander` hingegen zeigt DeepForest eine leicht höhere AUC von 0.76 im Vergleich zu 0.74 bei SAM_CLIP, was auf eine minimal bessere Klassifizierung und Trennung der Klassen bei DeepForest in diesem Fall hinweist. Insgesamt bleibt SAM_CLIP jedoch in den einfacheren Kategorien stärker.

4.2 Performance in Kapstadt

True Positive Percentage (TP%) und False Positive Percentage (FP%)

Die Ergebnisse für Kapstadt werden in den Boxplots 4.5 und 4.6 dargestellt. Die True Positive Percentage (TP%) und die False Positive Percentage (FP%) der Modelle SAM_CLIP und DeepForest werden für die Schwierigkeitsgrade `nicht_eng`, `eng` und `ineinander` visualisiert.

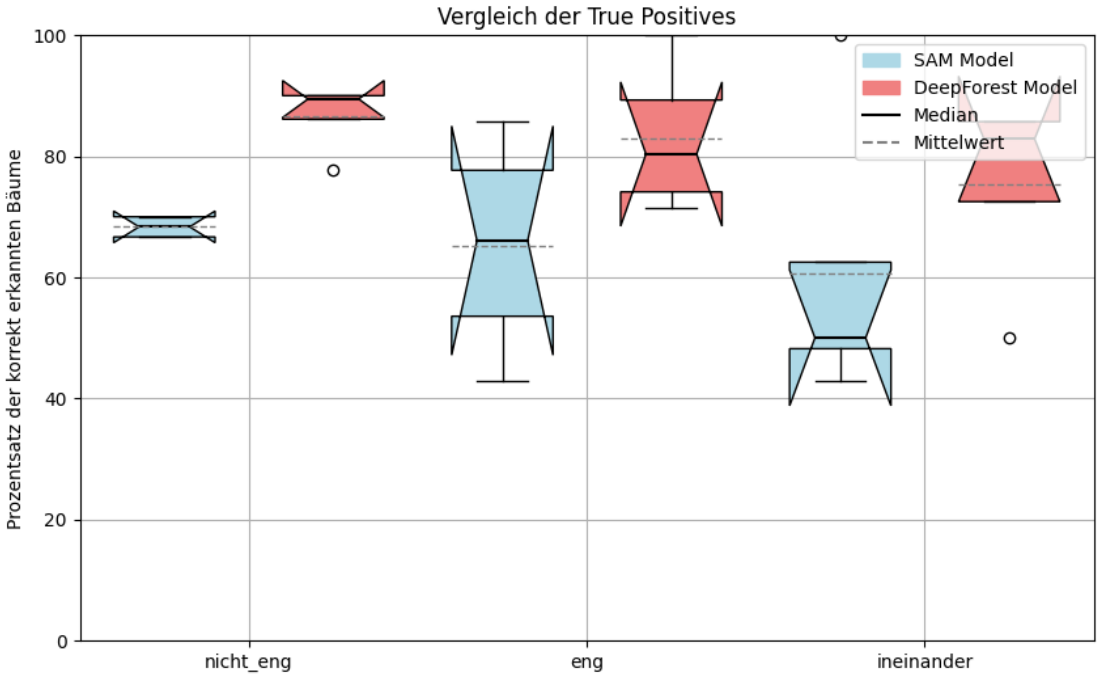

Abbildung 4.5: Boxplot der True Positive Percentage (TP%) für Kapstadt in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

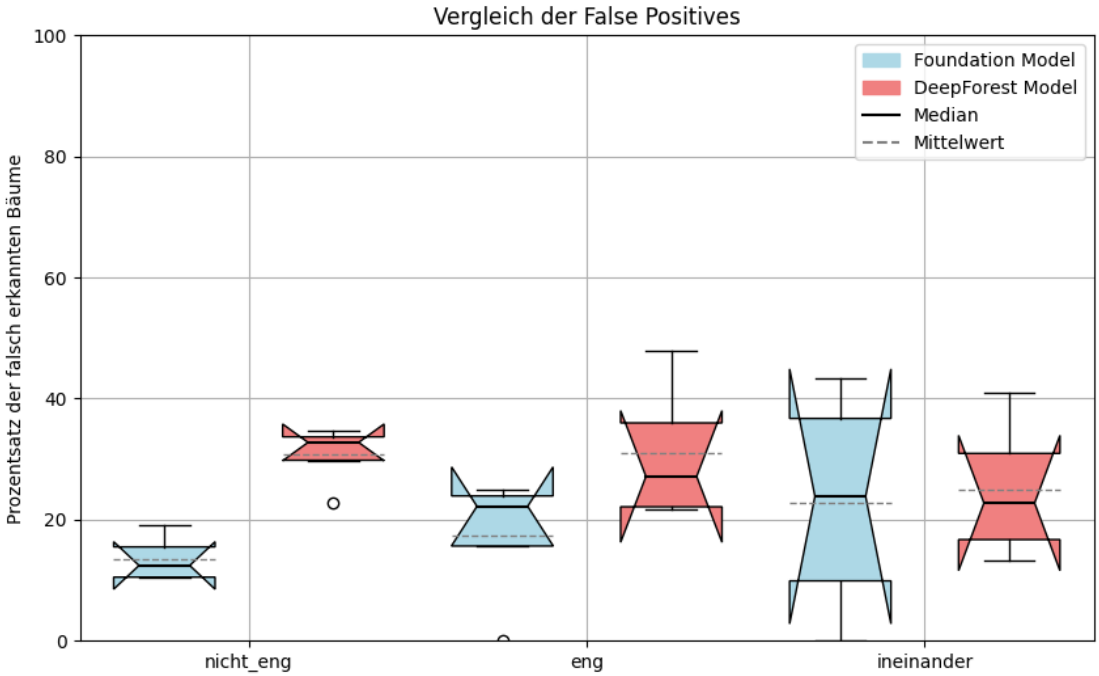

Abbildung 4.6: Boxplot der False Positive Percentage (FP%) für Kapstadt in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: Bei den **True Positives** zeigt das SAM_CLIP Modell in der Kategorie `nicht_eng` einen Median und einen Mittelwert von etwa 69%. Die Whiskers sind praktisch nicht messbar, da sie mit den Quantilen korrelieren. Im Vergleich dazu weist das DeepForest Modell einen höheren Median von etwa 85% auf, während der Mittelwert bei ca. 89% liegt. Die Whiskers sind auch hier nicht erkennbar. Ein Ausreißer ist bei 78% zu erkennen.

Bei den **False Positives** zeigt das SAM_CLIP Modell einen Median von etwa 14%, und der Mittelwert liegt ebenfalls bei ca. 13%. Die Whiskers reichen von etwa 13% bis 19%. Das DeepForest Modell zeigt einen deutlich höheren Median von etwa 30% und einen Mittelwert von ca. 32%. Die Whiskers reichen von etwa 30% bis 34%, mit einem Ausreißer unterhalb von 22%.

Kategorie `eng`: In der Kategorie `eng` sinkt die Leistung des SAM_CLIP Modells bei den **True Positives** minimal, wobei die Streuung deutlich zunimmt. Der Median liegt bei etwa 65%, und der Mittelwert bei ca. 66%. Die Whiskers reichen von etwa 43% bis

85%. Das DeepForest Modell zeigt hier einen Median von etwa 80%, und der Mittelwert liegt bei etwa 83%. Die Whiskers erstrecken sich von 72% bis 100%.

Bei den **False Positives** zeigt das SAM_CLIP Modell einen Median von etwa 18% und einen Mittelwert von ca. 22%. Die Whiskers reichen von etwa 18% bis 25%. Das DeepForest Modell zeigt hier einen schlechteren Median von etwa 30%, und der Mittelwert liegt bei ca. 26%. Die Whiskers reichen von 21% bis 44%.

Kategorie `ineinander`: In der Kategorie `ineinander` zeigt das SAM_CLIP Modell bei den **True Positives** eine schwächere Leistung mit einem Median von ca. 62%, und der Mittelwert liegt bei ca. 50%. Die Whiskers reichen von etwa 43% bis 62%. Das DeepForest Modell erreicht hier einen Median von etwa 75%, und der Mittelwert liegt bei ca. 84%. Die Whiskers reichen von 75% bis 86%, wobei es Ausreißer bei 50% gibt.

Bei den **False Positives** zeigt das SAM_CLIP Modell einen Median von ca. 23%, und der Mittelwert liegt bei ca. 24%. Die Whiskers erstrecken sich von etwa 0% bis 43%. Das DeepForest Modell zeigt einen etwas höheren Median von etwa 26%, und der Mittelwert liegt bei etwa 23%. Die Whiskers reichen von 14% bis 41%.

Zusammenfassung: In Kapstadt zeigt das DeepForest Modell bei den **True Positives** in allen Kategorien eine bessere Performance als das SAM_CLIP Modell. Besonders in den Kategorien `nicht_eng` und `eng` erreicht DeepForest sowohl höhere Median- als auch Mittelwerte, was auf eine stabilere Erkennung von Bäumen hinweist. Bei den **False Positives** sind die beiden Modelle insgesamt vergleichbarer. Während SAM_CLIP hier in allen Kategorien eine etwas geringere Fehlerrate zeigt, liefert DeepForest insgesamt eine robustere Leistung in Bezug auf die True Positives, während SAM bei den False Positives besser abschneidet.

Precision-Recall-Kurven

Die Precision-Recall-Kurven für Kapstadt in den drei Schwierigkeitsgraden werden in Abbildung 4.7 gezeigt. Sowohl die Precision als auch der Recall der Modelle SAM_CLIP und DeepForest werden für die Kategorien `nicht_eng`, `eng` und `ineinander` visualisiert. Zusätzlich sind die F1-Scores der beiden Modelle durch gestrichelte Linien dargestellt.

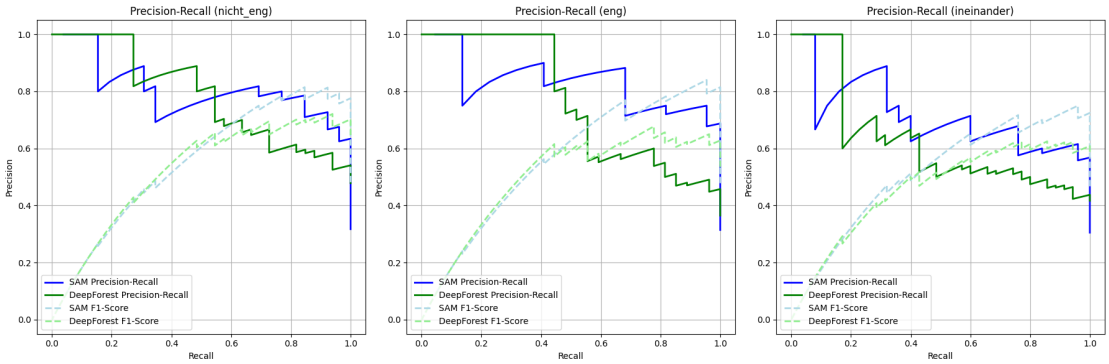

Abbildung 4.7: Precision-Recall-Kurve für Kapstadt in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: In der Kategorie `nicht_eng` zeigt das SAM Modell (blaue Linie) eine sehr gute Precision von ca. 0.8, die bis zu einem Recall von etwa 0.85 stabil bleibt. Der F1-Score von SAM (blau gestrichelte Linie) bewegt sich ebenfalls im hohen Bereich um 0.8. Im Vergleich dazu zeigt das DeepForest Modell (grüne Linie) eine anfangs etwas stärkere Precision, die jedoch bei steigendem Recall abfällt und bei etwa 0.55 bis 0.6 bleibt. Der F1-Score von SAM_CLIP (hellgrün gestrichelte Linie) liegt im Bereich von 0.7.

Kategorie `eng`: In der Kategorie `eng` zeigen beide Modelle eine anfänglich hohe Precision von 0.8 bis 1.0, die jedoch bei steigendem Recall deutlich abfällt und bei höheren Recall-Werten sinkt. Die Precision von DeepForest sinkt dabei auf ca. 0.5, während die Precision von SAM bei hohen recall-Werten auf bis zu 0.7 bleibt. Der F1-Score von DeepForest (grün gestrichelte Linie) liegt im Bereich von 0.6 bis 0.7. Der F1-Score von SAM_CLIP (hellblaue gestrichelte Linie) bleibt stabil bei Werten um 0.7 bis 0.85.

Kategorie `ineinander`: In der Kategorie `ineinander` zeigt das DeepForest Modell (grüne Linie) eine relativ stabile Precision, die um 0.6 liegt, bevor sie bei sehr hohen Recall-Werten über 0.95 sinkt. Der F1-Score (grün gestrichelte Linie) bleibt ab einem Recall von 0.5 im Bereich von 0.5 bis 0.6. Das SAM_CLIP Modell (blaue Linie) zeigt hingegen eine deutlich bessere Leistung mit einer Precision zwischen 0.55 und 0.7. Der F1-Score von SAM (hellblaue gestrichelte Linie) bleibt bei höheren Recall-Werten ab 0.5 stabil im Bereich von 0.6 bis 0.7.

Zusammenfassung: Die Precision-Recall-Kurven für Kapstadt zeigen, dass das SAM_CLIP Modell allen Kategorien eine bessere Leistung in Bezug auf Precision und F1-Score aufweist. SAM_CLIP erreicht eine Precision von ca. 0.8, die bis zu einem Recall von etwa 0.85 stabil bleibt (*eng*), während DeepForest nicht auf solch hohen Precisionen steigt. In der Kategorie `eng` fällt die Precision bei beiden Modellen mit steigendem Recall deutlich ab, jedoch hält SAM_CLIP die Precision auch bei höheren Recall-Werten stabiler im Bereich von 0.7, während DeepForest auf ca. 0.5 absinkt. In der Kategorie `ineinander` zeigt SAM_CLIP ebenfalls eine deutlich bessere Leistung mit einer stabileren Precision zwischen 0.55 und 0.7, während DeepForest eine Precision von ca. 0.6 zeigt, die bei hohen Recall-Werten abfällt. Der F1-Score von SAM_CLIP bleibt in allen Kategorien stabiler und höher im Vergleich zu DeepForest, insbesondere bei höheren Recall-Werten.

ROC-Kurven

Die ROC-Kurven für Kapstadt in den drei Schwierigkeitskategorien sind in Abbildung 4.8 dargestellt. Die ROC-Kurven zeigen die True Positive Rate (TPR) gegen die False Positive Rate (FPR) für die Modelle SAM_CLIP und DeepForest in den Kategorien `nicht_eng`, `eng`, und `ineinander`. Der AUC-Wert (Area Under the Curve) ist ebenfalls angegeben, um die Gesamtleistung zu quantifizieren.

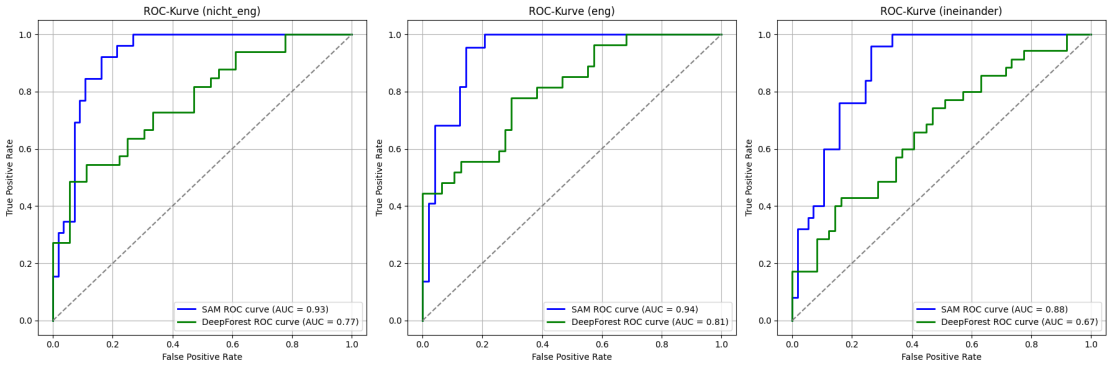

Abbildung 4.8: ROC-Kurve für Kapstadt in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: In der Kategorie `nicht_eng` zeigt das SAM_CLIP Modell (blaue Linie) eine hervorragende Leistung mit einer AUC von 0.93. Die ROC-Kurve

nähert sich schnell der oberen linken Ecke des Diagramms, was auf eine hohe True Positive Rate bei einer geringen False Positive Rate hinweist. Das DeepForest Modell (grüne Linie) schneidet mit einer AUC von 0.77 in dieser Kategorie schlechter ab. Die ROC-Kurve des DeepForest Modells steigt langsamer an und zeigt eine weniger effiziente Trennung der Klassen.

Kategorie `eng`: In der Kategorie `eng` zeigt das SAM_CLIP Modell (blaue Linie) erneut eine starke Performance mit einer AUC von 0.94. Die ROC-Kurve steigt früh an und erreicht schnell hohe True Positive Raten bei einer geringen False Positive Rate. Das DeepForest Modell (grüne Linie) zeigt in dieser Kategorie eine AUC von 0.81. Auch hier ist die Kurve weniger steil und bleibt hinter der Performance von SAM_CLIP zurück, zeigt jedoch eine ordentliche Klassentrennung.

Kategorie `ineinander`: In der Kategorie `ineinander` zeigt das SAM_CLIP Modell (blaue Linie) eine etwas schwächere Performance im Vergleich zu den vorherigen Kategorien, jedoch bleibt es stark mit einer AUC von 0.88. Die ROC-Kurve verläuft etwas weniger steil, erreicht jedoch eine solide True Positive Rate. Das DeepForest Modell (grüne Linie) zeigt in dieser Kategorie eine deutlich schlechtere Leistung mit einer AUC von 0.67. Die Kurve verläuft flacher und zeigt eine höhere False Positive Rate bei steigender True Positive Rate.

Zusammenfassung: Die ROC-Kurven für Kapstadt zeigen, dass das SAM_CLIP Modell in allen Kategorien (`nicht_eng`, `eng` und `ineinander`) eine bessere Performance aufweist als das DeepForest Modell. Besonders in den Kategorien `nicht_eng` und `eng` zeigt SAM_CLIP eine deutlich höhere True Positive Rate bei geringerer False Positive Rate, was sich in den höheren AUC-Werten widerspiegelt. Das DeepForest Modell schneidet insgesamt schlechter ab, besonders in der schwierigeren Kategorie `ineinander`.

4.3 Performance in San Francisco

True Positive Percentage (TP%) und False Positive Percentage (FP%)

Die Ergebnisse für San Francisco werden in den Boxplots 4.9 und 4.10 dargestellt. Die Boxplots visualisieren die True Positive Percentage (TP%) und False Positive Percen-

tage (FP%) der Modelle SAM_CLIP und DeepForest in den drei Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`.

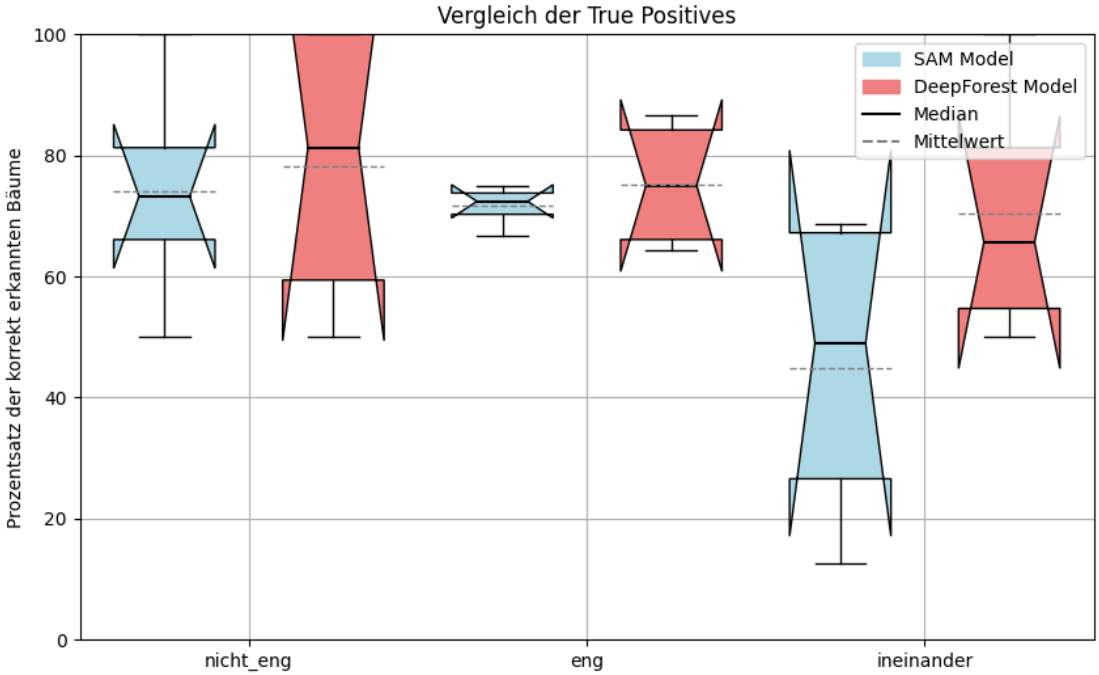

Abbildung 4.9: Boxplot der True Positive Percentage (TP%) für San Francisco in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

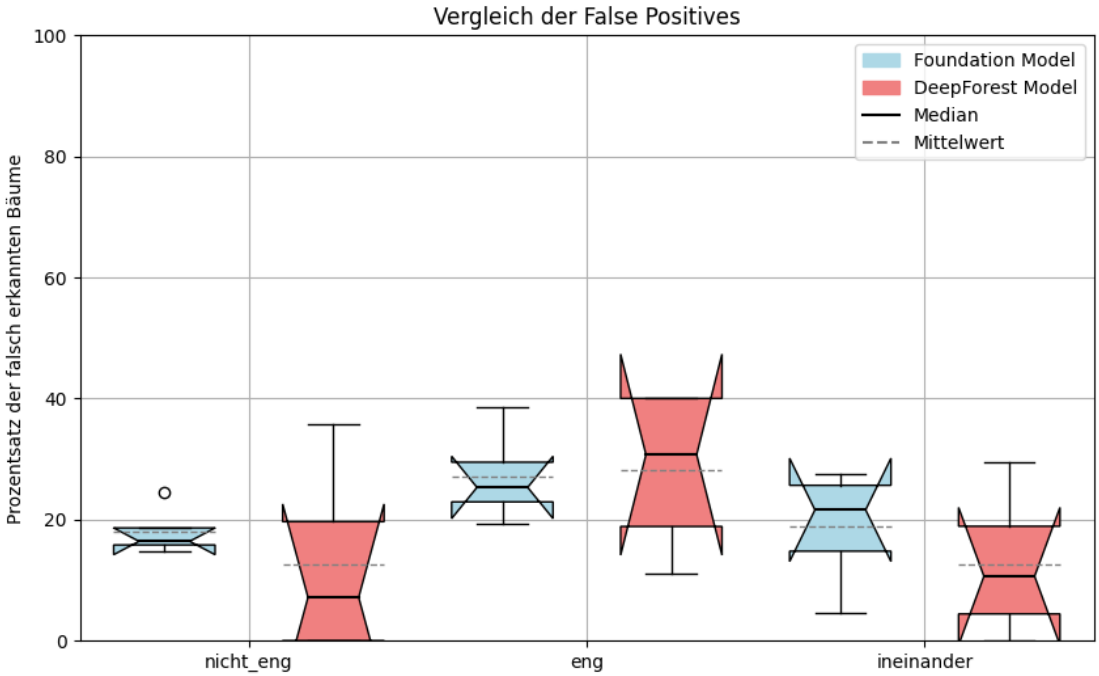

Abbildung 4.10: Boxplot der False Positive Percentage (FP%) für San Francisco in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: Bei den **True Positives** zeigt das SAM_CLIP Modell in der Kategorie `nicht_eng` einen Median von etwa 72%, während der Mittelwert bei ca. 71% liegt. Die Whiskers erstrecken sich von etwa 50% bis 100%. Im Vergleich dazu zeigt das DeepForest Modell einen leicht höheren Median von etwa 78%, während der Mittelwert bei ca. 81% liegt. Die Whiskers reichen ebenfalls von 50% bis 100%.

Bei den **False Positives** zeigt das SAM_CLIP Modell einen Median von etwa 19%, während der Mittelwert ebenfalls bei ca. 19% liegt. Die Whiskers reichen von etwa 15% bis 19%. Das DeepForest Modell weist einen etwas niedrigeren Median von etwa 11% und einen Mittelwert von ca. 9% auf. Die Whiskers reichen von etwa 0% bis 35%.

Kategorie `eng`: In der Kategorie `eng` zeigt das SAM_CLIP Modell bei den **True Positives** eine Leistung mit einem Median von etwa 70%, während der Mittelwert bei ca. 71% liegt. Die Whiskers reichen von etwa 67% bis 72%, hierbei ist das enge Quantil im Vergleich zu den anderen Schwierigkeitsgraden auffällig. Das DeepForest Modell zeigt

hier wieder einen leicht höheren Median und Mittelwert von etwa 71%. Die Whiskers erstrecken sich von 65% bis 86%.

Bei den **False Positives** zeigt das SAM_CLIP Modell einen Median von etwa 28% und einen Mittelwert von ca. 27%. Die Whiskers reichen von etwa 19% bis 38%. Das DeepForest Modell weist hier einen leicht höheren Median von etwa 29% auf, während der Mittelwert bei ca. 31% liegt. Die Whiskers reichen von 11% bis 40%.

Kategorie `ineinander`: In der Kategorie `ineinander` zeigt das SAM_CLIP Modell bei den **True Positives** eine Leistung mit einem Median von ca. 45%, während der Mittelwert bei ca. 50% liegt. Die Whiskers reichen von etwa 15% bis 70%. Das DeepForest Modell erreicht hier einen Median von etwa 67%, während der Mittelwert bei ca. 70% liegt. Die Whiskers reichen von 50% bis 100%.

Bei den **False Positives** zeigt das SAM_CLIP Modell einen Median von ca. 19%, während der Mittelwert bei ca. 22% liegt. Die Whiskers erstrecken sich von etwa 6% bis 29%. Das DeepForest Modell zeigt einen leicht niedrigeren Median von etwa 12%, während der Mittelwert bei etwa 10% liegt. Die Whiskers reichen von 0% bis 30%.

Zusammenfassung: In San Francisco zeigt das DeepForest Modell bei den **True Positives** in allen Kategorien durchweg leicht bessere bis deutlich bessere Ergebnisse als das SAM_CLIP Modell, insbesondere in der Kategorie `ineinander`, wo DeepForest einen deutlich höheren Median und Mittelwert erreicht. In den Kategorien `nicht_eng` und `eng` liegen die Mediane beider Modelle relativ nah beieinander. DeepForest hat in den Kategorien nicht eng und ineinander leicht niedrigere False Positives, während SAM_CLIP in der Kategorie nicht eng etwas niedrigere **False Positives** aufweist. Die Streuung des SAM_CLIP Modells besitzt in allen Kategorien eine deutlich engere Streuung der Ergebnisse.

Precision-Recall-Kurven

Die Precision-Recall-Kurven für San Francisco in den drei Schwierigkeitsgraden werden in Abbildung 4.11 gezeigt. Sowohl die Precision als auch der Recall der Modelle SAM_CLIP und DeepForest werden für die Kategorien `nicht_eng`, `eng` und `ineinander` visualisiert. Zusätzlich sind die F1-Scores der beiden Modelle durch gestrichelte Linien dargestellt.

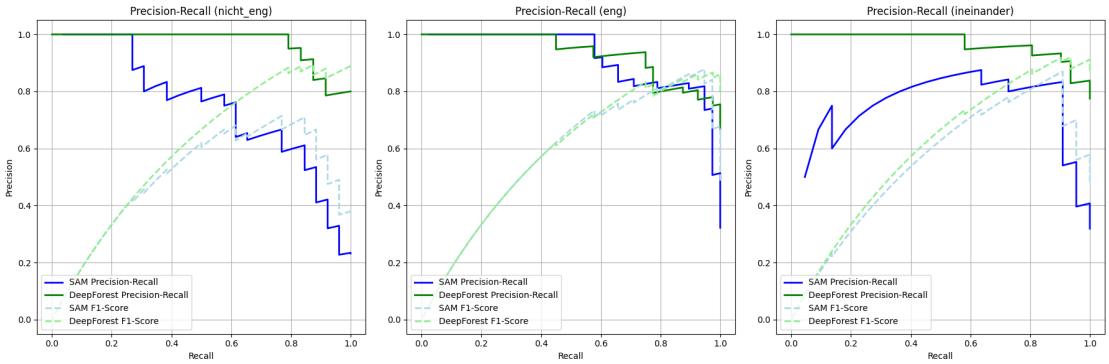

Abbildung 4.11: Precision-Recall-Kurve für San Francisco in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: In der Kategorie `nicht_eng` zeigt das DeepForest Modell (grüne Linie) eine nahezu perfekte Precision von 1.0, die bis zu einem Recall von etwa 0.8 stabil bleibt. Der F1-Score (grün gestrichelte Linie) bewegt sich im hohen Bereich von 0.85 bis 0.9. Im Vergleich dazu zeigt das SAM_CLIP Modell (blaue Linie) eine deutlich geringere Precision, die bei höheren Recall-Werten unter 0.6 fällt und zum besten Zeitpunkt auf 0.7 steigt. Auch scheint das Modell Probleme zu haben, eine stabile Precision herzustellen. Der F1-Score (hellblaue gestrichelte Linie) liegt im Bereich von 0.4 bis 0.7 und wird gerade in den sehr hohen Recall-Werten ab 0.9 deutlich schwächer.

Kategorie `eng`: In der Kategorie `eng` zeigt das DeepForest Modell (grüne Linie) eine starke Performance mit einer hohen Precision von 1.0, die jedoch bei einem Recall von etwa 0.75 abfällt. Anschließend sinkt der Precisions-Wert auf bis zu 0.75. Der F1-Score (grün gestrichelte Linie) liegt bei höheren Recall-Werten ab 0.5 im Bereich von 0.7 bis 0.85. Das SAM_CLIP Modell (blaue Linie) zeigt eine vergleichbare Performance, wobei die Precision ab einem Recall von 0.6 schon beginnt zu fallen und ca. bei 0.8 stabil bleibt. Bei sehr hohen Recall-Werten ab 0.95 sinkt die Precision beider Modelle merklich. Der F1-Score von SAM_CLIP (hellblaue gestrichelte Linie) bleibt bei höheren Recall-Werten ab 0.6 ansonsten bei etwa 0.8 stabil.

Kategorie `ineinander`: In der Kategorie `ineinander` zeigt das DeepForest Modell (grüne Linie) eine gute Precision, die bis zu einem Recall von etwa 0.6 stabil bleibt. Bei höheren Recall-Werten fällt diese dann jedoch auf ca. 0.8 ab. Der F1-Score (grün gestrichelte Linie) bewegt sich bei höheren Recall-Werten ab 0.6 im Bereich von 0.8

bis 0.95. Das SAM_CLIP Modell (blaue Linie) zeigt etwas schlechtere Performance mit einer Precision-Kurve, die brückenartig aussieht. Die Precisions-Kurve startet schwach bei 0.5, steigt dann bis zu einem Recall von 0.6 auf eine gute Precision von 0.85 und fällt anschließend wieder ab. Der Abfall der Precision ist gerade in den sehr hohen Recall-Werten ab 0.9 deutlich zu sehen. Der F1-Score (hellblaue gestrichelte Linie) steigt auf bis zu 0.85 und sinkt dann wieder.

Zusammenfassung: Die Precision-Recall-Kurven für San Francisco zeigen, dass das DeepForest Modell in allen drei Kategorien insgesamt eine leicht stärkere Performance aufweist. In der Kategorie `nicht_eng` zeigt DeepForest eine nahezu perfekte Precision, die bis zu einem Recall von 0.8 stabil bleibt, während SAM_CLIP hier deutlich schwächere Precisionswerte erreicht und Probleme hat, eine konstante Leistung zu erbringen. In der Kategorie `eng` ist SAM_CLIP überlegen, wenn auch nur leicht. SAM_CLIP zeigt hier eine stabilere Performance im mittleren Bereich, fällt jedoch bei sehr hohen Recall-Werten stärker ab als DeepForest. In der schwierigsten Kategorie `ineinander` zeigt DeepForest eine solide Leistung mit stabiler Precision und einem hohen F1-Score. SAM_CLIP hingegen hat hier Schwierigkeiten, eine stabile Leistung zu erbringen, wobei seine Precisionskurve stark variiert und bei hohen Recall-Werten abfällt.

ROC-Kurven

Die ROC-Kurven für San Francisco in den drei Schwierigkeitsgraden werden in Abbildung 4.12 gezeigt. Die ROC-Kurven zeigen die True Positive Rate (TPR) gegen die False Positive Rate (FPR) für die Modelle SAM_CLIP und DeepForest in den Kategorien `nicht_eng`, `eng`, und `ineinander`. Der AUC-Wert (Area Under the Curve) ist ebenfalls angegeben, um die Gesamtleistung zu quantifizieren.

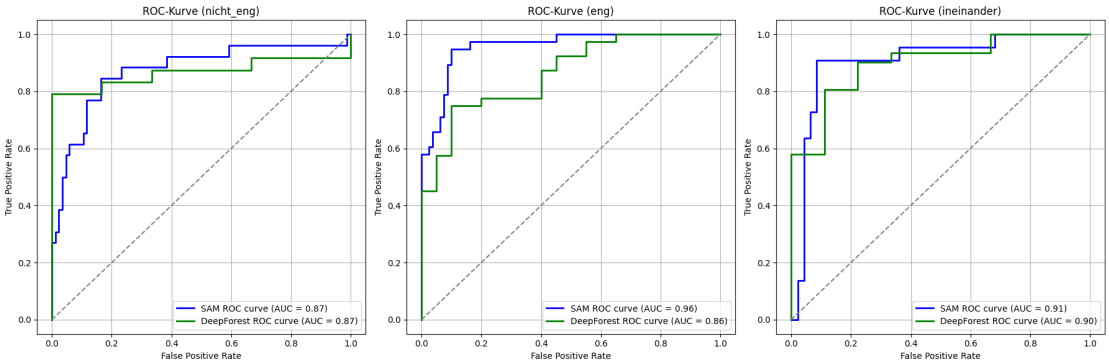

Abbildung 4.12: ROC-Kurve für San Francisco in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: In der Kategorie `nicht_eng` zeigt das SAM_CLIP Modell (blaue Linie) eine AUC von 0.87. Die ROC-Kurve zeigt eine schnelle Steigerung der True Positive Rate bei einer geringen False Positive Rate. Das DeepForest Modell (grüne Linie) erreicht in dieser Kategorie ebenfalls eine AUC von 0.87 und steigt sogar noch ein wenig steiler. Beide Modelle sind hier sehr vergleichbar und liefern ähnliche Ergebnisse, wobei SAM_CLIP auf eine etwas höhere True Positive Rate kommt.

Kategorie `eng`: In der Kategorie `eng` zeigt das SAM_CLIP Modell (blaue Linie) eine ausgezeichnete Performance mit einer AUC von 0.96. Die ROC-Kurve erreicht schnell hohe True Positive Werte und bleibt nah an der oberen linken Ecke. Das DeepForest Modell (grüne Linie) zeigt eine gute AUC von 0.86. Hier zeigt die Kurve von DeepForest eine etwas schwächere Performance im Vergleich zu SAM_CLIP, insbesondere bei höheren False-Positive-Raten.

Kategorie `ineinander`: In der Kategorie `ineinander` zeigt das SAM_CLIP Modell (blaue Linie) eine AUC von 0.91. Die ROC-Kurve steigt steil an und zeigt eine gute Trennung der Klassen bei höheren True Positive Raten. Das DeepForest Modell (grüne Linie) erreicht in dieser Kategorie eine AUC von 0.90, was eine vergleichbare Performance mit SAM_CLIP zeigt, jedoch insgesamt eine leicht geringere Steigung aufweist.

Zusammenfassung: Die ROC-Kurven für San Francisco zeigen, dass beide Modelle in allen drei Kategorien eine starke Leistung aufweisen. In der Kategorie `nicht_eng`

erreichen sowohl SAM_CLIP als auch DeepForest eine AUC von 0.87, mit ähnlichen Ergebnissen in Bezug auf die Trennung der Klassen, wobei SAM_CLIP eine etwas höhere True Positive Rate erzielt. In der Kategorie `eng` übertrifft SAM_CLIP das DeepForest Modell messbar, mit einer AUC von 0.96 im Vergleich zu 0.86 bei DeepForest. Hier zeigt SAM_CLIP eine bessere Trennung der Klassen und eine höhere True Positive Rate, insbesondere bei geringeren False Positive Raten. In der Kategorie `ineinander` liefern beide Modelle erneut vergleichbare Ergebnisse mit einer AUC von 0.91 für SAM_CLIP und 0.90 für DeepForest, wobei SAM_CLIP leicht überlegen ist. Insgesamt zeigt SAM_CLIP eine stärkere Performance, insbesondere in der Kategorie `eng`, während DeepForest in den anderen Kategorien nahe an SAM_CLIP herankommt.

4.4 Performance in Tokio

True Positive Percentage (TP%) und False Positive Percentage (FP%)

Die Ergebnisse für Tokio werden in den Boxplots 4.13 und 4.14 dargestellt. Die Boxplots zeigen die Verteilung der TP% und FP% für die Modelle SAM_CLIP und DeepForest in den drei Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`.

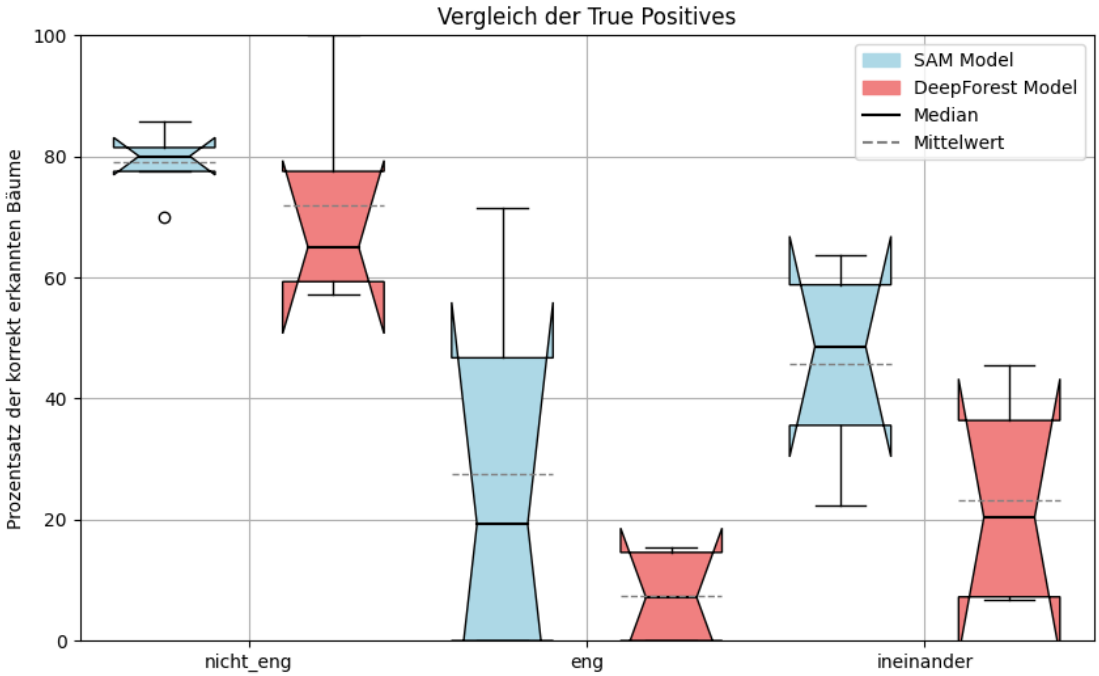

Abbildung 4.13: Boxplot der True Positive Percentage (TP%) für Tokio in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

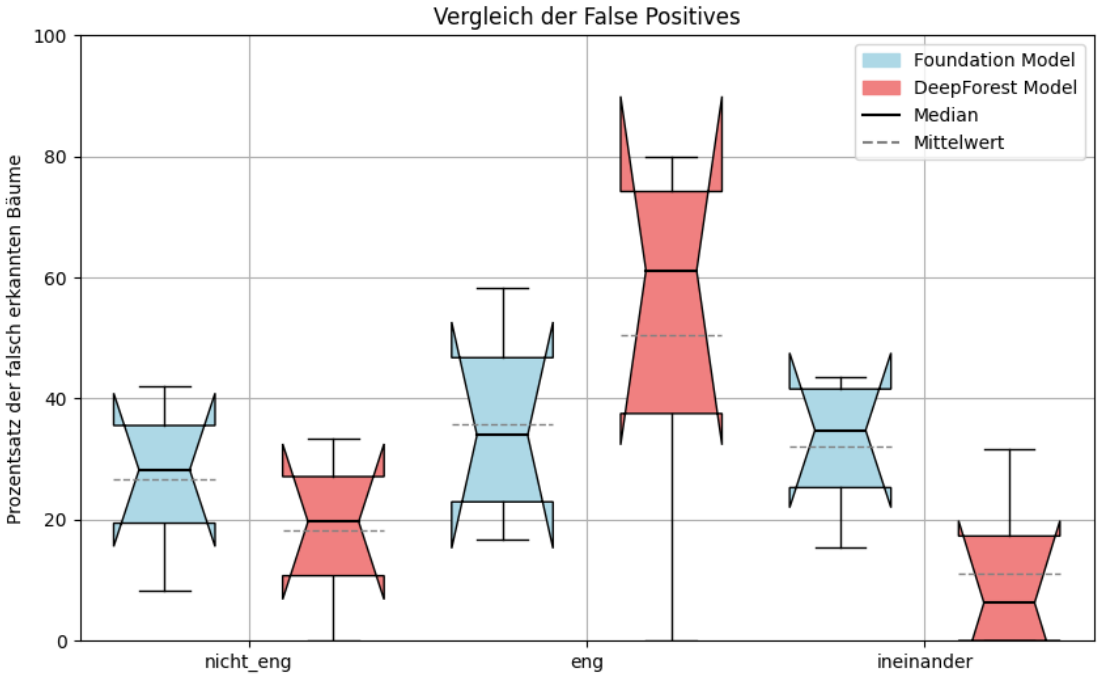

Abbildung 4.14: Boxplot der False Positive Percentage (FP%) für Tokio in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: Bei den **True Positives** zeigt das SAM_CLIP Modell in der Kategorie `nicht_eng` eine starke Performance mit einem Median von etwa 79%, während der Mittelwert bei ca. 80% liegt. Die Whiskers erstrecken sich von etwa 76% bis 86%, mit einem Ausreißer unterhalb von 70%. Das DeepForest Modell zeigt in dieser Kategorie eine etwas geringere Leistung, mit einem Median von etwa 73% und einem Mittelwert von ca. 65%. Die Whiskers reichen von etwa 56% bis 100%.

Bei den **False Positives** zeigt das SAM_CLIP Modell einen Median von etwa 28%, während der Mittelwert bei ca. 29% liegt. Die Whiskers erstrecken sich von etwa 10% bis 42%. Das DeepForest Modell zeigt eine geringere Fehlerrate mit einem Median von etwa 19% und einem Mittelwert von ca. 20%. Die Whiskers reichen von 0% bis 32%.

Kategorie `eng`: In der Kategorie `eng` zeigt das SAM_CLIP Modell bei den **True Positives** eine schwächere Leistung mit einem Median von etwa 36%, während der Mittelwert bei ca. 20% liegt. Die Whiskers erstrecken sich von etwa 0% bis 72%. Im Vergleich

dazu zeigt das DeepForest Modell einen Median und einen Mittelwert von etwa 9%. Die Whiskers reichen von etwa 0% bis 15%.

Bei den **False Positives** zeigt das SAM_CLIP Modell einen Median von etwa 35%, während der Mittelwert bei ca. 34% liegt. Die Whiskers reichen von 17% bis 59%. Das DeepForest Modell zeigt hier eine schlechtere Leistung mit einem Median von etwa 50% und einem Mittelwert von ca. 61%. Die Whiskers erstrecken sich von 0% bis 80%.

Kategorie `ineinander`: In der Kategorie `ineinander` zeigt das SAM_CLIP Modell bei den **True Positives** eine Leistung mit einem Median von etwa 48%, während der Mittelwert bei ca. 50% liegt. Die Whiskers erstrecken sich von etwa 22% bis 65%. Das DeepForest Modell erreicht hier einen Median von etwa 22%, und der Mittelwert liegt bei etwa 20%. Die Whiskers reichen von 9% bis 48%.

Bei den **False Positives** zeigt das SAM_CLIP Modell eine niedrigere Fehlerrate mit einem Median von etwa 31%, und der Mittelwert liegt bei ca. 32%. Die Whiskers erstrecken sich von 16% bis 42%. Das DeepForest Modell zeigt in dieser Kategorie eine geringere Fehlerrate mit einem Median von etwa 11% und einem Mittelwert von etwa 8%. Die Whiskers reichen von 0% bis 31%.

Zusammenfassung: In Tokio zeigt das SAM_CLIP Modell in allen Kategorien eine bessere Leistung bei den **True Positives**, mit einem höheren Median und Mittelwert im Vergleich zu DeepForest. Die Werte sind jedoch bei allen Kategrien ausgenommen von `eng` alle unterdurchschnittlich. Bei den **False Positives** zeigt SAM_CLIP in den Kategorien nicht eng und eng eine niedrigere Fehlerrate als DeepForest auf. In der Kategorie `ineinander` erreicht DeepForest bei den **False Positives** eine bessere Leistung als SAM_CLIP. Insgesamt zeigt SAM_CLIP in allen Kategorien eine höhere Erkennungsrate, während DeepForest besonders bei den **False Positives** in der Kategorie `nicht_eng` und `ineinander` punktet.

Precision-Recall-Kurven

Die Precision-Recall-Kurven für Tokio in den drei Schwierigkeitsgraden werden in Abbildung 4.15 gezeigt. Diese Diagramme zeigen die Precision und den Recall für die Modelle SAM_CLIP und DeepForest, einschließlich der F1-Scores für die verschiedenen Schwierigkeitsgrade `nicht_eng`, `eng` und `ineinander`.

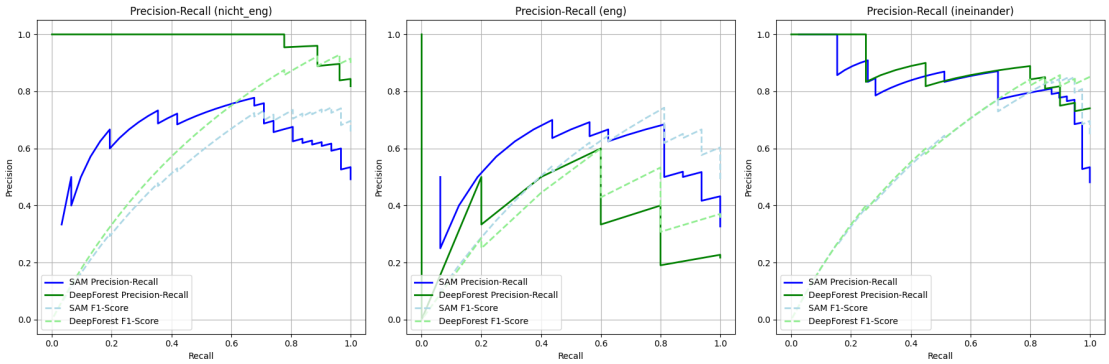

Abbildung 4.15: Precision-Recall-Kurve für Tokio in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: In der Kategorie `nicht_eng` zeigt das DeepForest Modell (grüne Linie) eine sehr hohe Precision von nahezu 1.0 bei fast allen Recall-Werten, was auf eine hervorragende Klassifizierung bei geringem Fehlerrisiko hindeutet. Die Precision sinkt erst ab einem Recall-Wert von ca. 0.8 auf ca. 0.82 ab. Das SAM_CLIP Modell (blaue Linie) zeigt hingegen eine eher mäßige Performance mit Precision-Werten. Die Precision-Recall-Kurve beschreibt einen Bogen, der bei einer Precision von ca. 0.4 beginnt, dann auf ca. 0.75 ansteigt und schließlich wieder auf 0.5 fällt. Der F1-Score von DeepForest (grün gestrichelte Linie) übersteigt deutlich den von SAM_CLIP in dieser Kategorie und ist zu seinem höchsten Stand bei ca. 0.9 im Vergleich zu dem höchsten Wert des F1-Scores von SAM_CLIP von ca. 0.75.

Kategorie `eng`: In der Kategorie `eng` zeigen sowohl DeepForest als auch SAM_CLIP, eine kurvenförmige Darstellung. Die Precision-Werte des DeepForest Modells (grüne Linie) beginnen bei etwa 1.0, fallen dann sofort auf 0.0, steigen bis zu einem Recall von 0.6 auf eine Precision von 0.6 und fallen anschließend wieder auf 0.2. SAM_CLIP beschreibt eine ähnliche Brückenkurve, jedoch mit einer konstant höheren Precision um ca. 0.1 Punkte im Vergleich zu DeepForest. Beide Modelle weisen eine größere Unsicherheit in der Klassifizierung auf. Die F1-Scores von SAM_CLIP zeigen ebenfalls eine durchgehend bessere Gesamtleistung im Vergleich zu DeepForest auf.

Kategorie `ineinander`: In der schwierigsten Kategorie, `ineinander`, liegen DeepForest und

SAM_CLIP nahezu gleichauf. Die Precision des DeepForest und des SAM_CLIP Modells bleibt über den Großteil des Recall-Bereichs stabil bei Werten um 0.8. Erst bei sehr hohen Recall-Werten ab 0.97 sinken beide Modelle etwas ab, wobei SAM_CLIP etwas stärker abfällt als DeepForest. Auch die F1-Score der beiden Modelle steigen konstant an und sinken erst am Ende der Recall-Werte etwas ab.

Zusammenfassung: In Tokio zeigt DeepForest in der Kategorie `nicht_eng` eine deutlich bessere Performance mit einer konstant hohen Precision von nahezu 1.0, während SAM_CLIP schwächere Ergebnisse liefert, wobei die Precision bei mittleren Recall-Werten auf 0.75 ansteigt, jedoch bei höheren Recall-Werten wieder sinkt. Der F1-Score von DeepForest übertrifft SAM_CLIP klar. In der Kategorie `eng` zeigen beide Modelle eine bogenförmige Precision-Kurve, jedoch schneidet SAM_CLIP durchgehend etwas besser ab, mit stabileren F1-Scores. In der schwierigsten Kategorie `ineinander` sind beide Modelle vergleichbar, mit stabilen Precision-Werten um 0.8, die bei sehr hohen Recall-Werten leicht abfallen, wobei SAM_CLIP etwas stärker absinkt. Insgesamt bleibt DeepForest in `nicht_eng` überlegen, während SAM_CLIP in `eng` etwas besser abschneidet.

ROC-Kurven

Die ROC-Kurven für Tokio in den drei Schwierigkeitsgraden `nicht_eng`, `eng`, und `ineinander` werden in Abbildung 4.16 dargestellt. Diese Diagramme zeigen den True Positive Rate (TPR) gegen die False Positive Rate (FPR) für die Modelle SAM_CLIP und DeepForest, zusammen mit dem jeweiligen AUC (Area Under the Curve)-Wert.

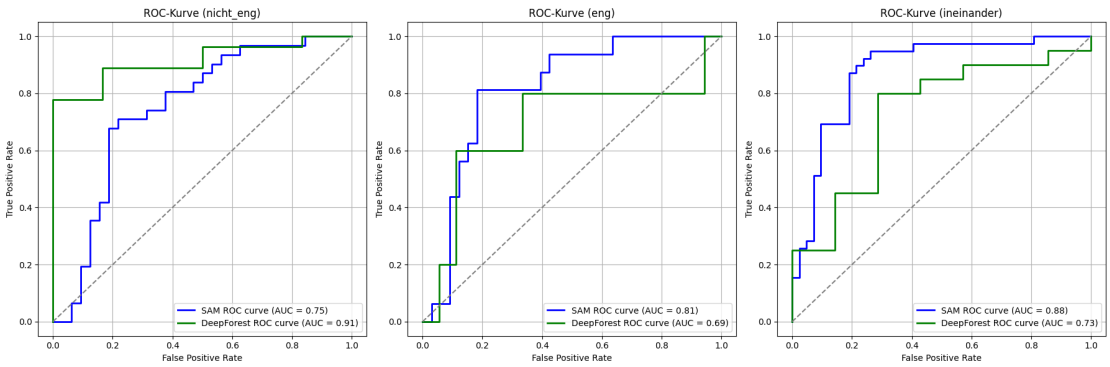

Abbildung 4.16: ROC-Kurve für Tokio in den Schwierigkeitsgraden `nicht_eng`, `eng` und `ineinander`

Kategorie `nicht_eng`: In der Kategorie `nicht_eng` zeigt das DeepForest Modell (grüne Linie) mit einem AUC von 0.91 eine exzellente Klassifizierungsleistung. Die ROC-Kurve von SAM_CLIP (blaue Linie) verläuft im Vergleich dazu weniger optimal, mit einem AUC von 0.75, was auf eine etwas schlechtere Unterscheidung von True Positives und False Positives hinweist.

Kategorie `eng`: In der Kategorie `eng` liegt der AUC-Wert von SAM_CLIP bei 0.81, während der AUC-Wert von DeepForest nur 0.69 beträgt. Dies zeigt, dass SAM_CLIP in dieser Schwierigkeitsstufe eine bessere Klassifizierungsgenauigkeit aufweist. DeepForest hat in diesem Fall eine schwächere Performance, was sich in der geringeren Genauigkeit der Trennung von True Positives und False Positives widerspiegelt.

Kategorie `ineinander`: In der anspruchsvollsten Kategorie `ineinander` zeigt SAM_CLIP erneut eine bessere Leistung, mit einem AUC von 0.88, im Vergleich zu DeepForest, das einen AUC von 0.73 erreicht. Dies deutet darauf hin, dass SAM_CLIP in dieser Kategorie besser in der Lage ist, zwischen True Positives und False Positives zu unterscheiden.

Zusammenfassung: Die Ergebnisse der ROC-Kurven für Tokio zeigen, dass DeepForest in der einfacheren Kategorie `nicht_eng` eine ausgezeichnete Performance bietet, während SAM_CLIP in den schwierigeren Kategorien `eng` und `ineinander` überlegen ist. Insbesondere bei eng beieinanderstehenden Bäumen und Bäumen, die ineinander übergehen, erzielt SAM_CLIP eine höhere Genauigkeit in der Unterscheidung von True und False Positives.

4.5 Zusammenfassung der Ergebnisse

Die durchgeführten Experimente zur Leistungsbewertung der Modelle SAM_CLIP und DeepForest in der Segmentierung von Einzelbäumen in urbanen Wäldern haben in verschiedenen Städten und Segmentierungsschwierigkeitsgraden
(`nicht_eng`, `eng`, `ineinander`) teils unterschiedliche Ergebnisse geliefert.

In der Kategorie `nicht_eng` zeigen die Ergebnisse, dass das SAM_CLIP Modell in den meisten Städten eine solide Performance liefert, besonders in Hamburg und Kapstadt, in

der es sowohl bei den True Positives als auch bei den False Positives stabilere und bessere Werte erzielt. DeepForest hingegen zeigt in Tokio eine sehr starke Performance mit einer nahezu perfekten Precision und einem hohen F1-Score, was auf eine hervorragende Klassifizierung bei niedrigem Fehlerrisiko hinweist. In San Francisco erreichen beide Modelle ähnliche Werte, wobei SAM_CLIP eine etwas geringere Fehlerrate aufweist.

In der Kategorie `eng` zeigt SAM_CLIP insgesamt die bessere Performance, insbesondere in Kapstadt und San Francisco, wo es durchgängig bessere Precision-Werte und F1-Scores erzielt. DeepForest schneidet in dieser Kategorie schwächer ab, insbesondere in Städten wie Tokio und Kapstadt, in der es bei den True Positives hinter SAM_CLIP zurückbleibt. Trotzdem zeigt DeepForest in San Francisco bei mittleren Recall-Werten eine stabile Precision.

In der schwierigsten Kategorie `ineinander` liegen die Ergebnisse zwischen SAM_CLIP und DeepForest näher beieinander. In Kapstadt und Tokio zeigt SAM_CLIP eine bessere Leistung, insbesondere bei hohen Recall-Werten, während DeepForest in San Francisco bessere Werte bei den True Positives erzielt. Bei den False Positives schneidet SAM_CLIP in den meisten Städten insgesamt besser ab, insbesondere in Kapstadt, in der es deutlich niedrigere Fehlerraten zeigt.

Die Precision-Recall- und ROC-Kurven verdeutlichen, dass SAM_CLIP in den komplexeren Segmentierungsszenarien (`eng` und `ineinander`) insgesamt stabiler ist und sowohl bei der Precision als auch bei den F1-Scores besser abschneidet. DeepForest zeigt seine Stärke insbesondere in einfacheren Szenarien wie `nicht_eng`, wo es in Städten wie Tokio eine nahezu perfekte Precision erreicht.

Insgesamt zeigen die Ergebnisse, dass SAM_CLIP in den schwierigen Szenarien (`eng` und `ineinander`) stabilere und bessere Ergebnisse liefert, während DeepForest in den einfacheren Szenarien (`nicht_eng`) eine etwas bessere Leistung erbringt. Die Entscheidung für ein Modell hängt daher stark von der jeweiligen Segmentierungsumgebung und dem Schwierigkeitsgrad der Aufgabe ab.

5 Diskussion

In dieser Arbeit wurde der Versuch unternommen, die Leistung eines fein abgestimmten Basismodells mit der Leistung eines Spezialmodells zu vergleichen. Dabei wurde sich auf eine Domäne fokussiert; die der Erkennung von Baumkronen auf RGB-Satellitenbildern in urbanen Wäldern. Als Basismodell wurde das Segment Anything Modell gewählt, während als Spezialmodell DeepForest eingesetzt wurde. Die zugrunde liegenden Hypothesen waren:

- **H1**: Ein fein abgestimmtes Basismodell ist mindestens genauso leistungsfähig wie ein spezialisiertes Modell im Bereich der Segmentierung von Baumkronen in urbanen Wäldern.
- **H2**: Die Feinabstimmung eines Basismodells zur Identifikation von Baumkronen in urbanen Wäldern mittels RGB-Bildern sollte einen positiven Einfluss auf die Segmentierungsergebnisse in einer anderen Flora haben, da auf RGB-Bildern eine Flora nicht von einer anderen zu unterscheiden ist.

DeepForest, das speziell für die Erkennung von Baumkronen auf Luftbildaufnahmen und Satellitenbildern trainiert wurde, zeigte hierbei eine relativ konsistent gute Leistung, wobei diese auch abhängig von unterschiedlichsten Faktoren messbar gestört wurde. Die richtigen Segmentierungen haben zudem mit ansteigender Schwierigkeit an Qualität verloren.

SAM, welches als ein Foundation-Model vorgestellt und mit einer Vielzahl an unterschiedlichen Bildern aus unterschiedlichen Domänen trainiert wurde, erbrachte gute Segmentierungsergebnisse in einer Domäne, in der das Modell nicht trainiert wurde. Gerade in schwierigen Umgebungen (`ineinander`) konnten gute Segmentierungsergebnisse erzielt werden, was ggf. auf die Zero-Shot-Fähigkeit von SAM zurückzuführen ist, welche es dem Modell ermöglichen soll, auch bei bisher unbekannten Bildern gute Ergebnisse zu erzielen[27]. Diese Erkenntnisse stützen **H1** und **H2**.

5.1 Vergleichbarkeit der Modelle

SAM und DeepForest basieren auf unterschiedlichen Architekturideen. Die Architektur von SAM ist darauf ausgelegt, nach der Durchführung des Bild-Encodings nahezu in Echtzeit Segmentierungen durchzuführen. Diese Stärke hat keinen Einfluss auf den Vergleich der Leistungsfähigkeit der Modelle, da die Performance nicht berücksichtigt wurde. Auch die Rechenzeit, der manuelle Aufwand oder der benötigte Stromverbrauch beim initialen Training der Modelle haben keinen Einfluss auf die Bewertung.

Um die Modelle miteinander vergleichbar zu machen, wurde SAM mithilfe des CLIP Modells die Möglichkeit der Instanzsegmentierung verliehen, die auch von DeepForest gemacht wird.

5.1.1 Anpassung des Foundation Models

Bevor die finalen Ergebnisse besprochen werden, muss erwähnt werden, dass die nicht finalen Ergebnisse des SAM hier nicht berücksichtigt werden. Die Ergebnisse des SAM im Verlauf der Feinabstimmung können in drei Abschnitte unterteilt werden.

1. Die Ergebnisse des SAM nach Abschluss des Datentrainings (wie im Abschn.: 3.4.1).
2. Die Ergebnisse des SAM nach Abschluss des Hyperparametertrainings (wie im Abschn.: 3.4.1).
3. Die Ergebnisse des SAM nach Abschluss des Trainings und dem Einsatz von CLIP (wie im Abschn. 3.4.2), welche die finalen und vergleichbaren Ergebnisse darstellen, werden mit den Instanzsegmentierungen des DeepForest Modells verglichen.

Ergebnisse des Datentrainings von SAM

Das Ergebnis der Segmentierungen dieses Verarbeitungsschritts sind nahezu perfekte Segmentierungen in annähernd jeder Umgebung. Grund hierfür ist, dass SAM nach Abschluss der Feinabstimmung in der Lage ist, selbst ineinander stehende Baumkronen solide zu trennen. Sollte es doch einmal ein Problem bei der eindeutigen Trennung geben, so kann der Benutzer mithilfe des Prompt Encoders positive und negative Punkte auf dem Satellitenbild setzen und so die Segmentierungsergebnisse weiter nach seinen

Vorstellungen verfeinern. Dies war der Grundgedanke des SAM, leider ist dies kein vergleichbares Ergebnis, zu dem DeepForest Modell, welches einzig ein Bild benötigt und anschließend die Bauminstanzen selbstständig erkennt und via Instanzsegmentierung zurückgibt. Das Vorgehen und das Ergebnis dieses Verarbeitungsschrittes kann somit nicht als vergleichbar angesehen werden, in Bezug auf die Ergebnisse von DeepForest.

Ergebnisse des Hyperparametertrainings von SAM

Das Ergebnis der Segmentierungen von SAM nach diesem Trainingsschritt sind herausragende Segmentbildungen auf den verwendeten Satelitenbildern, die in Abb. 5.1 visualisiert sind. Durch die Optimierung der Hyperparameter wird das Satellitenbild ohne zusätzliche Eingabe von Prompts des Benutzers ausgeführt und erkennt so eigenständig eine Vielzahl von Objekten auf dem zu verarbeiteten Bild.

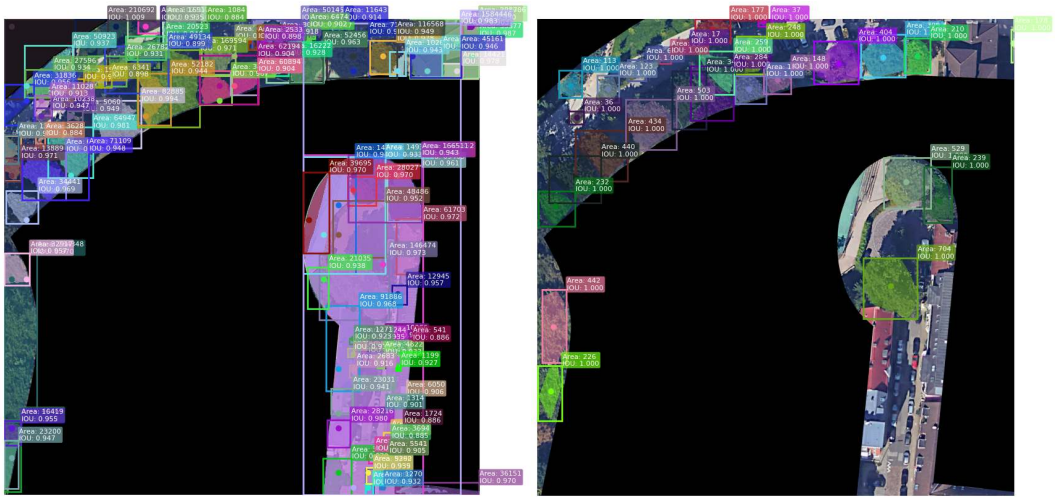

Abbildung 5.1: Links die erkannten Segmente von SAM und rechts die darin enthaltenen Baumkronen.

Es wird deutlich, dass innerhalb eines Satellitenbildes eine Vielzahl von Objekten enthalten ist. Selbst nach der Anwendung eines Korridors um die Kanten eines Straßengraphen herum bleiben hier unüberschaubar viele weitere Objekte bestehen; hierzu gehören Straßen, Autos, Dächer, etc.. Das rechte Bild der Abbildung 5.1 zeigt die Wunschsegmentierung dieser Arbeit und gleichzeitig auch die Ground-Truth des betrachteten Abschnittes. Die hier abgebildeten Segmente sind in den Segmenten der linken Seite der Abbildung

enthalten, allerdings eine Minderheit unter diesen. Um diese gewollten Segmente zu erhalten, muss ein Benutzer ebendiese Segmente manuell heraussuchen und als gewollt markieren. Nach wiederholten Überprüfungen ist aufgefallen, dass die richtigen Segmente immer in der Gesamtmenge aller gemachten Segmente enthalten sind. Das bedeutet, würde man diese manuelle Arbeit durchführen und das daraus erhaltene Ergebnis als vergleichbar zu DeepForest betrachten, so wäre das Basismodell SAM mit Segmenten, die nahe bei 100 Prozent liegen, das mit Abstand stärkere Modell. DeepForest erstellt jedoch vollautomatisch und selbstständig die gewünschten Segmente und klassifiziert diese als Baum. Damit kann das Ergebnis dieser Feinabstimmung weiterhin nicht als vergleichbar angesehen werden, da es manuellen Aufwand benötigt, um die Baumkronen zu identifizieren.

Ergebnisse der Feinabstimmung von CLIP

Der finale Verarbeitungsschritt des Basismodells SAM endet mit der Integration und der Feinabstimmung des zusätzlichen Modells CLIP. Dieses Modell wurde im Paper von SAM prototypisch inkludiert. Es handelt sich um ein Modell, welches Wörter und Bilder miteinander assoziiert und in einen gemeinsamen Vektorraum projiziert. Ziel der Integration des Modells ist es, dem Basismodell SAM mit der Texteingabe *Tree* mitzuteilen, dass nun aus der Masse der Segmentierungen einzig die Bäume segmentiert werden sollen. Um dies zu implementieren, übernimmt das fein abgestimmte CLIP Modell innerhalb der Prompt Encoders die Aufgabe, *Baum* von *kein Baum* zu unterscheiden. Somit ist es dem Basismodell möglich, durch die Eingabe eines Satellitenbildes ohne weiteres Zutun des Nutzers die segmentierten Einzelbauminstanzen zurückzubekommen, um diese mit dem Spezialmodell zu vergleichen. Das Ergebnis ist somit vergleichbar mit dem von DeepForest, da beide Modelle nun eine Instanzsegmentierung von Bäumen durchführen. Die Genauigkeit der Erkennung von Bäumen ist jedoch deutlich unter den vorangegangenen 100 Prozent, die in den beiden ersten Verarbeitungsschritten erreicht wurden.

5.1.2 Postprocessing

Das Postprocessing beim Feinabstimmen des SAM wurde auf die gleiche Art umgesetzt, wie das Postprocessing bei DeepForest implementiert wurde. Dieser Schritt ist wichtig, um zu verhindern, dass dieser Verarbeitungsschritt einen Einfluss auf die Segmentierungsergebnisse der Modelle hat.

5.2 Analyse der Modellleistung: SAM vs. DeepForest

Bei der Untersuchung der Ergebnisse des Basismodells im Vergleich zu dem Spezialmodell sind verschiedene Begebenheiten aufgefallen. Nachfolgenden eine konsolidierte Zusammenfassung der Ergebnisse über alle Floren (Hamburg, Kapstadt, San Francisco und Tokio) hinweg sowie über alle definierten Schwierigkeitskategorien (nicht eng, eng und ineinander). Hierbei wird die Betrachtungsart von SAM zu DeepForest eingenommen.

Tabelle 5.1: Zusammenfassung der Leistung der Modelle SAM und DeepForest in verschiedenen Städten und Schwierigkeitsgraden

Stadt	**Schwierigk.**	**TP**	**FP**	**Prec.-Rec.**	**ROC**
Hamburg	Nicht eng	Besser	Besser	Besser	Besser
	Eng	Besser	Gleich gut	Besser	Besser
	Ineinander	Besser	Besser	Gleich gut	Besser
Kapstadt	Nicht eng	Schlechter	Besser	Besser	Besser
	Eng	Schlechter	Besser	Besser	Besser
	Ineinander	Schlechter	Gleich gut	Besser	Besser
San Francisco	Nicht eng	Schlechter	Schlechter	Schlechter	Gleich gut
	Eng	Gleich gut	Gleich gut	Gleich gut	Besser
	Ineinander	Schlechter	Schlechter	Gleich gut	Gleich gut
Tokio	Nicht eng	Besser	Schlechter	Schlechter	Schlechter
	Eng	Besser	Besser	Besser	Besser
	Ineinander	Besser	Schlechter	Gleich gut	Besser

Die Tabelle beschreibt das Basismodell im Vergleich zu dem Spezialmodell und besitzt die Abstufungen *Besser*, *Schlechter* und bei leichten Unterschieden der Ergebnisse *etwas besser* oder *etwas schlechter*. Diese Werte sind die kumulierten Ergebnisse aus den Tabellen im Anhang (siehe Abschnitt: A.2). Sollten die Ergebnisse der Modelle nahezu identisch sein, so wird dies in der Tabelle vereinfacht als *gleich gut* dargestellt. Diese Darstellung dient der einfacheren Zusammenfassung aller Ergebnisse, welche ansonsten schwer zusammenzubringen sind. Es wird ersichtlich, dass das Basismodell in Hamburg eine deutlich bessere Performance im Vergleich zum Spezialmodell erbringt. Diese Erkenntnis stützt klar **H1**, da das fein abgestimmte Basismodell nicht nur vergleichbare, sondern sogar bessere Ergebnisse als das Spezialmodell in seiner Domäne liefert. In den Städten Kapstadt, San Francisco und Tokio sind die Ergebnisse vergleichbarer, wobei zu

sehen ist, dass die Aufteilung der Schwierigkeitskategorien offensichtlich nicht dediziert genug gewählt wurde. Einige andere Einflussfaktoren beeinflussen die Segmentierungsergebnisse neben den gewählten Kategorien erheblich. Der Konsens der vergleichbaren Ergebnisse bleibt von dieser Tatsache jedoch unberührt und stützt die Hypothese **H2**, da es sich nur um geographische Ausdehnungen und Floren handelt, welche nicht Teil der Feinabstimmung waren.

Es wird außerdem ersichtlich, dass das Spezialmodell häufiger relativ hohe True Positive (in Prozent) Werte erreicht. Dies lässt sich bei der Untersuchung der Vorhersagewahrscheinlichkeiten darauf zurückführen, dass DeepForest eher konservativ segmentiert. Die Diagramme zeigen, wie die TP und FP-Werte bei einem `predicion_score` von 0.5 liegen. Dies äußert sich dadurch, dass das Modell recht hohe Unsicherheiten bei der Klassifizierung als Baum hat und die `prediction_scores` hinter denen des Basismodells zurückfallen (siehe Tabelle 5.2-Schwellenwert). Eine Auswirkung dieses Verhaltens ist, dass die wenigen Vorhersagen mit hohem Score, die DeepForest macht, dann auch richtig sind, da sich das Modell sehr sicher ist, dass es sich bei dem betrachteten Segment um einen Baum handelt. Das Basismodell ist hierbei weniger konservativ und erreicht unter anderem aus diesem Grund häufiger weniger hohe True Positive Raten, da auch Segmente als Baum erkannt werden, die tatsächlich keine sind, da die Darstellung der TP und FP Diagramme mit einem `prediction_score` von 0.5 gemacht wurden, stehen diese Ergebnisse den Hypothesen **H1** und **H2** nicht entgegen. Sollten die in der Tabelle 5.2 angegebenen optimalen Schwellenwerte für die jeweiligen Modelle genutzt werden, würde sich die Verteilung der Werte innerhalb der Boxplots klar zugunsten des Basismodells verschieben. Zudem würden die Unsicherheiten deutlich abnehmen, was auf eine konsistentere und präzisere Leistung des Basismodells hinweisen würde.

Bei der genauen Betrachtung der Precision-Recall-Diagramme schneidet das Basismodell über alle Städte und Schwierigkeiten hinweg insgesamt etwas besser ab. Diese Ergebnisse unterstützen somit weiter **H1** und **H2**. Um eine genauere Untersuchung der Werte zu unternehmen, sind in den nachfolgenden Tabellen zusammengefasst, bei welchem prediction_score der höchste F1-Score erreicht wird und das pro Schwierigkeit und Modell.

Stadt	Modell	Korridor	Schwellenw.	Precision	Recall	F1
Hamburg	SAM	nicht_eng	0.95	0.4362	0.9318	0.5942
	DeepForest	nicht_eng	0.2	0.6552	0.4750	0.5507
	SAM	eng	0.9	0.4805	0.9487	0.6379
	DeepForest	eng	0.2	0.5484	0.5484	0.5484
	SAM	ineinander	0.95	0.4953	0.8833	0.6347
	DeepForest	ineinander	0.15	0.5455	0.7500	0.6316
Kapstadt	SAM	nicht_eng	0.8	0.7273	0.9231	0.8136
	DeepForest	nicht_eng	0.15	0.5870	0.8182	0.6835
	SAM	eng	0.95	0.7241	0.9545	0.8235
	DeepForest	eng	0.2	0.5500	0.8148	0.6567
	SAM	ineinander	0.95	0.6154	0.9600	0.7500
	DeepForest	ineinander	0.15	0.4706	0.9143	0.6214
San Francisco	SAM	nicht_eng	0.85	0.6552	0.7308	0.6909
	DeepForest	nicht_eng	0.0	0.8000	1.0000	0.8889
	SAM	eng	0.95	0.7826	0.9474	0.8571
	DeepForest	eng	0.15	0.7800	0.9750	0.8667
	SAM	ineinander	0.75	0.7692	0.9091	0.8333
	DeepForest	ineinander	0.15	0.9063	0.9355	0.9206
Tokio	SAM	nicht_eng	0.85	0.6552	0.7308	0.6909
	DeepForest	nicht_eng	0.0	0.8000	1.0000	0.8889
	SAM	eng	0.95	0.7826	0.9474	0.8571
	DeepForest	eng	0.15	0.7800	0.9750	0.8667
	SAM	ineinander	0.75	0.7692	0.9091	0.8333
	DeepForest	ineinander	0.15	0.9063	0.9355	0.9206

Tabelle 5.2: Precision-Recall Ergebnisse ohne Gesamtergebnis

In der obigen Tabelle ist ersichtlich, dass bei gegebenem Schwellenwert der F1-Score des Basismodells bei zwei von vier Städten besser ist als der des Spezialmodells. Außerdem ist zu sehen, dass die F1-Scores bei einer der Städte deutlich höhere Werte erlangten und nur knapp hinter dem Spezialmodell in San Francisco und Tokio liegen. Dass die Ergebnisse in trainingsfremden Umgebungen nur leicht hinter denen des Spezialmodells lagen, lässt den Rückschluss zu, dass das Training mit der Flora von Hamburg einen positiven Effekt auf andere fremde Florenerkennungen hat. Diese Überlegung stützt weiter **H2**. Hier sind

die F1-Scores beider Modelle gleichzeitig sehr hoch und übertreffen die Ergebnisse in Hamburg und Kapstadt deutlich.

Stadt	Modell	Precision	Recall	F1-Score
Hamburg	SAM	0.4707	0.9213	0.6223
	DeepForest	0.5830	0.5911	0.5769
Kapstadt	SAM	0.6889	0.9459	0.7957
	DeepForest	0.5358	0.8491	0.6539
San Francisco	SAM	0.7357	0.8624	0.7938
	DeepForest	0.8288	0.9702	0.8921
Tokio	SAM	0.7357	0.8624	0.7938
	DeepForest	0.8288	0.9702	0.8921

Tabelle 5.3: Precision-Recall Ergebnisse mit Gesamtergebnis

In der weiter konsolidierten Version der Tabelle, in der alle Schwierigkeitsgrade gemeinsam betrachtet werden, kann der eben erhaltene Eindruck weiter gefestigt werden. Die Modelle unterscheiden sich in ihren F1-Scores nicht in einem solchen Maße, dass die Aussage zutreffen würde, eines der beiden Modelle als Besser oder Schlechter zu bezeichnen. Die Leistungen sind abhängig von den betrachteten Städten, entweder auf der Seite des Basismodells (SAM) oder auf der des Spezialmodells (DeepForest) besser. Wichtig bei der Betrachtung der Tabellen ist, dass hier die Werte mit dem höchsten F1-Score abgetragen wurden, da diese als harmonisches Mittel zwischen hoher Precision und hohem Recall dienen. Der nicht existierende „große“ Unterschied spricht bei den unbekannten Floren für **H2**.

Bei näherer Betrachtung der ROC-Diagramme kann eine tabellarische Form gewählt werden, um diese anschließend in ihrer Güte zu beurteilen. Hierfür wird folgender Rahmen genutzt:

AUC-Wert	Güteklasse
1.00	Perfekt
0.90 - 0.99	Hervorragend
0.80 - 0.89	Exzellent
0.70 - 0.79	Akzeptabel
0.50 - 0.69	Mittelmäßig

Tabelle 5.4: Einordnung der AUC-Werte in Güteklassen[22]

Hierbei ist ein Wert, der bei oder unter 0.5 liegt, faktisch geraten. Hieraus ergibt sich folgende Tabelle.

Stadt	Modell	Korridor	AUC	Güteklasse
Hamburg	SAM	nicht_eng	0.84	Exzellent
		eng	0.76	Akzeptabel
		ineinander	0.74	Akzeptabel
	DeepForest	nicht_eng	0.66	Mittelmäßig
		eng	0.65	Mittelmäßig
		ineinander	0.76	Akzeptabel
Kapstadt	SAM	nicht_eng	0.93	Hervorragend
		eng	0.94	Hervorragend
		ineinander	0.88	Exzellent
	DeepForest	nicht_eng	0.77	Akzeptabel
		eng	0.81	Exzellent
		ineinander	0.67	Mittelmäßig
San Francisco	SAM	nicht_eng	0.87	Exzellent
		eng	0.96	Hervorragend
		ineinander	0.91	Hervorragend
	DeepForest	nicht_eng	0.87	Exzellent
		eng	0.86	Exzellent
		ineinander	0.90	Hervorragend
Tokio	SAM	nicht_eng	0.75	Akzeptabel
		eng	0.81	Exzellent
		ineinander	0.88	Exzellent
	DeepForest	nicht_eng	0.91	Hervorragend
		eng	0.69	Mittelmäßig
		ineinander	0.73	Akzeptabel

Tabelle 5.5: AUC-Werte und Güteklassen der Experimente

Es wird ersichtlich, dass keines der Modelle schlechte Ergebnisse liefert. Außerdem kann beobachtet werden, dass das fein abgestimmte Basismodell niemals unter einen guten AUC-Wert (Area Under the Curve[37]) fällt (0.74). Insgesamt zeigen die AUC-Werte beider Modelle, dass sie relativ gut darin sind, Baumkronen zu segmentieren, auch wenn beide Modelle erwartungsgemäß etwas Schwierigkeiten damit haben, in den höheren Schwierigkeitskategorien solide Ergebnisse zu liefern. Die Auswertung der AUC-Werte bestätigt außerdem die Erkenntnisse der F1-Scores, nach denen das Basismodell in un-

terschiedlichen Umgebungen mit unterschiedlicher Flora gute Segmentierungsergebnisse liefert. Das ROC-Modell (Receiver Operating Characteristic[37]) untersucht die Leistung des Modells zur Klassifikation, in dem es die Beziehung zwischen der True Positive Rate (TPR) (auch bekannt als Sensitivität oder Recall) und der False Positive Rate (FPR) über verschiedene Schwellenwerte hinweg betrachtet. Mit dieser Eigenschaft kann ein genaueres Gesamtbild der Leistungsfähigkeit der Modelle geschaffen werden, welches die vorangegangenen Eindrücke unterstützt.

5.2.1 Konsistenz der Diagrammergebnisse

Nach Analyse der Precision-Recall-Werte und der AUC-Werte wird ersichtlich, dass die Diagramme und Experimente eine konsistente Aussage zu der Leistungsfähigkeit der beiden untersuchten Modelle abgeben. Diese Aussage ist über verschiedene Städte und damit Floren, sowie zu unterschiedlichen Bedingungen bestätigt worden.

Alle Ergebnisse lassen den Schluss zu, dass das SAM in den meisten Fällen eine höhere Leistungsfähigkeit als DeepForest besitzt und somit **H1** und **H2** gleichermaßen nahelegen. Dies wird sowohl durch die höheren AUC-Werte als auch die besseren Precision- und Recall-Werte in mindestens der Hälfte der untersuchten Fälle in den unterschiedlichen Schwierigkeitsgraden beobachtet. In den Städten *Kapstadt* und *San Francisco* erzielt SAM sogar exzellente AUC-Werte von bis zu 0.96 (*eng*), somit findet hier eine hervorragende Trennung von positiven und negativen Instanzen statt. Diese Ergebnisse werden durch ähnlich gute Ergebnisse bei den Precision- und Recall-Werten in besagten Schwierigkeitsgraden noch weiter unterstützt.

In *Hamburg* sind die Ergebnisse für das SAM-Modell ebenfalls konsistent, wenn auch unterdurchschnittlich, im Vergleich zu den anderen Städten. Dieses auffällige Verhalten wird in Abschnitt 5.2.2 näher betrachtet. Dennoch sind die Ergebnisse hier mit AUC-Werten von 0.74 bis 0.84 weiterhin sehr gut, auch wenn die Precision- und Recall-Werte nicht über 0.63 hinausgehen.

Im Vergleich dazu zeigt das DeepForest-Modell teilweise gemischte Ergebnisse bei der Segmentierung. Während es in der Segmentierung von Baumkronen weiterhin stark ist, weist es in den höheren Schwierigkeitsgraden eine etwas weniger stabile Leistung auf. Auch bei diesem Modell sind die Precision-Recall-Werte und die AUC-Werte konsistent zueinander. Sehr hohe Precision-Recall- und AUC-Werte von bis zu 0.90 beispielsweise in *San Francisco* in dem Schwierigkeitsgrad *ineinander* zeigen, dass das Spezialmodell

nicht nur in einfacheren Segmentierungsaufgaben gute bis sehr gute Leistungen erreichen kann, sondern auch in schwer zu segmentierenden Abschnitten gute Leistungen erreicht. DeepForest bleibt jedoch in anderen Städten, wie *Hamburg* und *Kapstadt*, hinter SAM zurück, was sowohl in den AUC-Werten als auch in den niedrigeren Precision- und Recall-Werten deutlich wird. Die Ergebnisse aus den ROC-Kurven (AUC) und den Precision-Recall-Werten unterstützen sich gegenseitig in ihrer Aussage, dass SAM tendenziell besser darin ist, Baumkronen zu segmentieren, besonders in schwierigeren Korridoren wie *eng* und *ineinander*. DeepForest zeigt zwar in spezifischen Fällen eine solide Leistung, ist jedoch im Allgemeinen weniger konsistent in seiner Klassifikationsgenauigkeit, was sowohl durch die AUC-Werte als auch durch die Precision-Recall-Werte bestätigt wird.

5.2.2 Einfluss der Feinabstimmung auf die Ergebnisse

Bei der Experimentdurchführung und Auswertung der Daten ist neben dem Vergleich von SAM und DeepForest auch der Vergleich innerhalb von SAM und zwischen den Städten ein bemerkenswerter Aspekt. Sam wurde in allen Feinabstimmungsprozessen nur mit Daten aus der Stadt Hamburg verbessert. Das Ziel war es, eine Verbesserung mindestens für diese Flora zu erhalten. Trotz dieser Tatsache sind die Precsion-Recall-Werte und die AUC-Werte in dem Bereich von Hamburg schlechter als die der anderen Städte. Das schlechtere Abschneiden der Umgebung Hamburg deutet darauf hin, dass es andere lokale Gegebenheiten in dieser Umgebung gibt, die das Segmentierungsergebnis maßgeblich beeinflussen.

Ein weiterer Schluss ist, dass die Feinabstimmung mit Daten aus Hamburg auch die Segmentierungen in anderen Umgebungen positiv beeinflusst hat (**H2**). Dafür spricht, dass in allen untersuchten Städten, einschließlich *Kapstadt*, *San Francisco* und *Tokio*, mit dem Basismodell gute bis sehr gute Ergebnisse erzielt wurden. Dies könnte darauf zurückzuführen sein, dass die Unterscheidung verschiedener Floren mittels Spektralbänder (Multispektral- oder Hyperspektralbilder) erfolgt, die in den verwendeten RGB-Bildern nicht enthalten sind. Die einzige Unterscheidung erfolgt daher über den Grünton (stark beeinflusst von der Sonneneinstrahlung) und die Form der Pflanze, da auch Höheninformationen nicht zur Verfügung stehen. Wenn nun also die Pflanzen der unterschiedlichen Floren sich in Form und Farbe ähneln, so kann davon ausgegangen werden, dass die Daten von Hamburg zur Feinabstimmung auch in den anderen Städten für bessere Segmentierungsergebnisse gesorgt haben und die Hypothese **H2** weiter stützen.

Eine weitere mögliche Ursache für das relative schlechte Abschneiden der Segmentierung innerhalb von Hamburg, trotz lokalem Trainingsdatensatz, ist die ggf. schwierige Umgebung und die Tatsache, dass Hamburg eine *grüne Stadt* ist und somit viele Grünflächen, Bäume u.ä. besitzt. Hierin unterscheidet sich Hamburg klar von den anderen Testumgebungen, die alle einen sehr spärlichen urbanen Wald besitzen (innerhalb der untersuchten geographischen Ausdehnung). Zudem könnte die urbane Struktur, wie die Höhe der Gebäude, deren Schatten oder eine besonders bunte Vielfalt von Straßenfahrzeugen, Einfluss auf die Ergebnisse haben, ebenso wie weitere mögliche Faktoren, die in Abschnitt 5.2.3 beschrieben wurden. Die geringeren Precision- und Recall-Werte sowie die AUC-Werte in Hamburg könnten auf diese externen Einflüsse zurückzuführen sein, die die Feinabstimmung in dieser spezifischen Stadt weniger wirksam gemacht haben.

5.2.3 Einfluss weiterer Faktoren auf die Segmentierungsergebnisse

Bei Betrachtung der Ergebnisse beider Modelle über alle Schwierigkeitskategorien hinweg wird ersichtlich, dass die Ergebnisse nicht in allen Fällen mit steigender Schwierigkeit auch schlechter werden, wie zu erwarten gewesen wäre. Dies ist einerseits auf die Tatsache zurückzuführen, dass es sich bei der Wahl der Schwierigkeitsgrade um eine vereinfachte Darstellung handelt und eine Menge Einflussfaktoren unbeachtet lässt. Dies war nötig, da die Aufnahme aller möglichen Einflussfaktoren auf das Segmentierungsergebnis den Rahmen dieser Arbeit überstiegen hätte. Zudem wäre dies nicht ausschlaggebend für die Beantwortung der aufgestellten Hypothesen **H1** und **H2**. Hätte der Anspruch bestanden, alle möglichen Einflussfaktoren zu berücksichtigen und einen sehr dedizierten Vergleich unter Einbeziehung sämtlicher Faktoren zu unternehmen, so wäre es unumgänglich gewesen, die betrachteten Koordinaten visuell zu überprüfen und eine ausreichende Menge an Daten mit den spezifischen Gegebenheiten zu akquirieren. Die stichpunktartige Auswertung der Anomalien ist jedoch durchaus umsetzbar und wurde im Folgenden unternommen. Nachfolgend werden alle identifizierten messbaren Einflussfaktoren im Rahmen der unternommenen Segmentierungen erläutert:

- Schatten (von Bäumen und Gebäuden): Besonders hohe Gebäude werfen lange Schatten, die die Segmentierung erschweren. Andererseits konnte festgestellt werden, dass gerade flache Häuser eine Segmentierung maßgeblich verbessern. In der unteren Abbildung 5.2 sind im linken Bild Büsche nicht von Bäumen zu unterscheiden, außerdem sind im rechten Bild komplexe Schattenverhältnisse vorhanden, welche die Segmentierung deutlich erschweren.

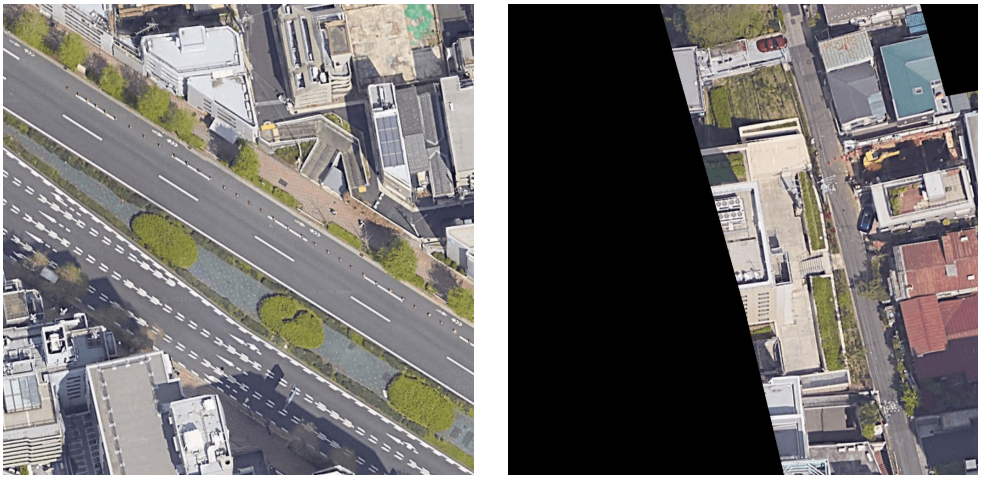

Abbildung 5.2: Satellitenbildausnahmen zur Verdeutlichung der komplexen Schattenstrukturen.

- Sonneneinstrahlung: Starke Sonneneinstrahlung, wie in Kapstadt, führte zu Reflexionen, welche die Modelle herausforderten. Es konnte beobachtet werden, dass eine Unterscheidung von verschiedenen Floren nicht über die Grünsättigung unternommen werden kann (siehe Abbildung: 5.3), da diese beträchtlich von der Sonneneinstrahlung zum Zeitpunkt der Aufnahme des Satellitenbildes abhängt.

Abbildung 5.3: Satellitenbildaufnahmen zur Verdeutlichung der Farbveränderungen durch unterschiedliche Lichteinstrahlungswinkel.

- Farbe der Baumkronen: Helle oder ungewöhnliche Farben führten zu Fehlklassifikationen. In allen Städten konnte beobachtet werden, dass ein Blätterdach, welches nicht grün ist, beide Modelle vor immense Herausforderungen stellte. Dies ließe sich mit Sicherheit durch die Anpassung des entsprechenden Trainingsdatensatzes verbessern.

Abbildung 5.4: Satellitenbildaufnahmen zur Verdeutlichung der Herausforderung von andersfarbigen Blätterdächern.

In der Abbildung 5.4 ist deutlich auf dem linken und rechten Bild zu erkennen, dass viele der Baumkronen nicht grün sind. Zusätzlich ist zusehen, wie in einem vorangegangenen Punkt besprochen, dass die Schattenverhältnisse die Segmentierung weiter verkomplizieren.

- Form und Dichte der Flora: Unregelmäßige Baumkronen oder dichte Baumgruppen erschwerten die Erkennung, insbesondere in Tokio und San Francisco. Bei Bäumen ohne sichtbares Blätterdach konnte deutlich erkannt werden, dass eine Segmentierung als Baum für keines der Modelle möglich war. Diese Problematik ist voraussichtlich auch nicht lösbar mit der ausschließlichen Nutzung von RGB-Bildern. LiDAR-Daten könnten hier die Segmentierungsergebnisse verbessern. Beachtlich ist auch, dass DeepForest, obwohl mit LiDAR-Daten trainiert, genauso erhebliche Schwierigkeiten bei der Segmentierung dieser Bäume hatte.

Abbildung 5.5: Satellitenbildaufnahmen zur Verdeutlichung der Herausforderung von sehr spärlichen Blätterdächern.

In Abbildung 5.5 ist unten links im linken Bild ein Baum zu sehen, der sich im Vergleich zu den anderen Baumkronen des Ausschnitts kaum vom Hintergrund absetzt. Außerdem ist im rechten Bild eine ganze Reihe von Bäumen, welche die gleiche Eigenschaft aufweisen. Durch unregelmäßige Verteilungen solcher Sonderfälle muss es folglich zu Schwankungen innerhalb der Segmentierungsergebnisse kommen.

- Objekte wie Autos: Farblich ähnliche Objekte führten zu falschen Erkennungen. Eine grundsätzliche Herausforderung in urbanen Wäldern ist das Vorhandensein von diversen fremden Objekten. Hierzu gehören Autos, Dachfenster, Dächer, Fahrbahnmarkierungen, etc., welche abhängig von Form und Farbe einen deutlichen Einfluss auf die Segmentierungsergebnisse beider Modelle haben.

- Fehlende Höheninformationen: Da nur 2D-Bilder verwendet wurden, war es schwer, Bäume in verschiedenen Ebenen korrekt zu segmentieren (dies wird auch deutlich in Abb.: 5.2). Auch wenn DeepForest mit LiDAR-Daten trainiert wurde, konnte es diese Stärke in den unternommenen Experimenten nicht ausspielen, da die Eingabedaten aus 2-D-RGB-Bildern bestanden. Grünflächen, wie Rasen oder Büsche, wurden fälschlicherweise häufiger als Baumkronen identifiziert.

- Patch-Zerlegung und Überlappung: Je nach gewählter Überlappung in der Bildaufteilung variieren die Ergebnisse. Gerade die Überlappung von Bildern kann zu deutlichen Unterschieden in den Segmentierungsergebnissen führen, daher wurden

nach Möglichkeit Bildausschnitte gewählt, die in ihrer Größe als ein Bild verarbeitet werden konnten.

- Insgesamt sind Bäume in städtischen Umgebungen keine homogene Masse wie in Wäldern, was zusätzliche Herausforderungen für die Segmentierung mit sich bringt. Hier besteht ein grundsätzlicher Unterschied in den Herausforderungen zwischen einem urbanen und einem klassischen Wald. Ein klassischer Wald ist eine homogene Masse von Baumkronen, wohingegen ein urbaner Wald eine sehr heterogene Masse von Objekten darstellt. Beide Gebiete haben hier spezifische Herausforderungen für ein Segmentierungsmodell.

Nach Überprüfung der zusätzlichen Einflussfaktoren innerhalb der Experimente lässt sich sagen, dass die spezifischen Faktoren beide Modelle in einem ähnlichen Ausmaß gestört haben sollten. Sie sollten somit kein Kriterium sein, nach dem eines der beiden untersuchten Modelle einen Nachteil oder einen Vorteil bei der Beurteilung ihrer Leistungsfähigkeit hat. Diese Erkenntnis hat die Implikation, dass der Einfluss dieser Anomalien den Hypothesen **H1** und **H2** nicht entgegenstehen.

6 Fazit

Die Entwicklung des Frameworks und die Durchführung der Experimente hatten das Ziel, herauszufinden, ob ein fein abgestimmtes generisches Modell durch Transfer Learning in der Lage ist, vergleichbare oder bessere Ergebnisse als ein spezialisiertes Modell in seiner Domäne zu liefern. Hierfür wurde ein experimentelles Design und dazugehörige Metriken gewählt, welche ein umfassendes Bild der Leistungen der beiden untersuchten Modelle in Hinblick auf die aufgestellten Hypothesen liefern.

6.1 Forschungsziel

Das Hauptanliegen ist hierbei die Leistungsfähigkeit in der Domäne der Erkennung von Baumkronen in urbanen Wäldern mit hochauflösenden Satellitenbildern. Diese Domäne wurde gewählt, da es sich hierbei um eine herausfordernde Umgebung für Segmentierungsmodelle handelt und es ein aktuelles großes Interesse an dieser Domäne gibt, was eine Vielzahl von neuen Veröffentlichungen in diesem Bereich stützen. Es wurde experimentell untersucht, ob SAM nach Feinabstimmung als Basismodell genauso leistungsstark wie DeepForest (ein spezialisiertes Modell) ist. Damit soll zur Schließung einer Forschungslücke beigetragen werden, indem diese Arbeit einen direkten Vergleich aufzeigt. Viele Veröffentlichungen der jüngeren Zeit entscheiden sich für die Feinabstimmung mittels Transfer Learning von Basismodellen oder die Implementierung spezieller Modelle für ihre Domäne. Beispiele der erwähnten Veröffentlichungen sind *GeoSAM*[49] und *Tree-GPT*[11], die Ansätze der Feinabstimmung von großen Basismodellen implementieren oder *Individual tree-crown detection*[5], welches sich für ein spezialisiertes Modell entschieden hat. Viele dieser Veröffentlichungen erreichen beeindruckende Ergebnisse, wenn auch nicht perfekt. Der Schritt vor der Implementierung des Modells, in dem die Frage nach dem optimalen Vorgehen gestellt wird, bleibt in vielen Veröffentlichungen

oft unbeachtet. Angesichts dessen wurde die hier beschriebene Arbeit und das implementierte System geschrieben, um mithilfe unterschiedlicher Schwierigkeitskategorien in unterschiedlichen Floren ein fundiertes Ergebnis zu dieser Frage zu liefern.

6.2 Methodik und Entscheidungsgrundlage

Die gewählte Methodik bestand darin, ein sehr bekanntes und erfolgreiches Basismodell mithilfe eines bekannten und erfolgreichen Spezialmodells umfassend in ihren Leistungsfähigkeiten über ein ganzheitliches experimentelles Design gegenüberzustellen. Der Grund für die Wahl von SAM[27] liegt in seiner Vielseitigkeit und seiner Fähigkeit, durch Feinabstimmung in verschiedene Domänen übertragen zu werden. DeepForest[54] hingegen wurde speziell für die Segmentierung von Baumkronen entwickelt und bietet daher einen idealen Vergleichsmaßstab. Um den Rahmen dieser Arbeit nicht zu sprengen, wurde die Leistungsbeurteilung auf die Segmentierungsfähigkeiten beschränkt, während Merkmale wie z.B. die Rechenzeit, Stromverbrauch, Trainingszeit usw. nicht betrachtet wurden. Die Segmentierungsfähigkeiten wurden umfassend anhand verschiedener Floren weltweit untersucht (Hamburg, Kapstadt, San Francisco und Tokio), und es wurde eine Kategorisierung der Schwierigkeitsgrade (nicht_eng, eng, ineinander) entwickelt.

6.3 Schlussfolgerung

Die hier gemachten Experimente und die anschließende Untersuchung zeigen, dass sowohl das Spezialmodell als auch das fein abgestimmte Basismodell eine angemessene Leistung erbringen können. In Übereinstimmung mit der ersten These **H1** lässt sich festhalten, dass das fein abgestimmte Basismodell in einer Vielzahl von Szenarien eine vergleichbare oder sogar bessere Leistung als das Spezialmodell erreichen konnte. Zudem bin ich davon überzeugt, dass eine diversere Datenlage und verbessertes Training des CLIP-Modells die Leistung des Basismodells noch einmal signifikant verbessern können.

Die zweite Hypothese **H2**, dass die Feinabstimmung des Basismodells auf RGB-Bildern einen positiven Effekt auf Segmentierungen in anderen Floren hat, konnte ebenfalls durch die Experimente bestätigt werden. Das Basismodell zeigte in allen betrachteten Floren eine vergleichbare Leistung wie das Spezialmodell. Diese Beobachtungen deuten darauf

hin, dass die Unterschiede zwischen den Floren auf RGB-Bildern nicht signifikant genug sind, um die Segmentierungsleistung beträchtlich zu beeinflussen.

Zusammenfassend legt diese Arbeit den Schluss nahe, dass generische Modelle wie SAM durch die Feinabstimmung in die Lage versetzt werden, Spezialmodelle in ihren Domänen abzulösen.

6.4 Ausblick

Durch die Schaffung eines Frameworks, das es Nutzern ermöglicht, globale hochauflösende Satellitenbilder für unterschiedliche Forschungsfragen zu akquirieren, können zahlreiche weitere Fragestellungen angegangen werden. Dies ist besonders relevant, da die Akquirierung großer Datenmengen im Bereich der tiefen neuronalen Netze oft eine kaum zu überwindende Hürde darstellt und potenziell interessierte Forscher und Studenten hiervon abgeschreckt werden könnten. Ein interessanter Aspekt dieser Arbeit ist die Untersuchung, inwieweit eine Verbesserung des Trainingsdatensatzes zu einer gesteigerten Segmentierungsleistung führen kann. Außerdem könnte eine Untersuchung der Baumgesundheit durchgeführt werden, sobald Google die angekündigte Erweiterung der Google Earth API mit Daten der letzten 80 Jahre zur Verfügung stellt. Dies würde es ermöglichen, die Entwicklung der Gesundheit urbaner Wälder weltweit zu analysieren. Sobald diese Fragestellungen bearbeitet sind, könnte auf dieser Grundlage eine Artenbestimmung implementiert werden, welche die differenzierte Beobachtung von urbanen Wäldern deutlich verbessern würde.

Literaturverzeichnis

[1] Alrasheedi, Fahad ; Zhong, Xin ; Huang, Pei-Chi: Padding module: Learning the padding in deep neural networks. In: *IEEE Access* 11 (2023), S. 7348–7357

[2] Angoca: *Map Features*. Jul 2024. – URL https://wiki.openstreetmap.org/wiki/Map_features

[3] Arora, Raman ; Basu, Amitabh ; Mianjy, Poorya ; Mukherjee, Anirbit: Understanding deep neural networks with rectified linear units. In: *arXiv preprint arXiv:1611.01491* (2016)

[4] Aubry-Kientz, Mélaine ; Laybros, Anthony ; Weinstein, Ben ; Ball, James G. ; Jackson, Toby ; Coomes, David ; Vincent, Grégoire: Multisensor data fusion for improved segmentation of individual tree crowns in dense tropical forests. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 14 (2021), S. 3927–3936

[5] Beloiu, Mirela ; Heinzmann, Lucca ; Rehush, Nataliia ; Gessler, Arthur ; Griess, Verena C.: Individual tree-crown detection and species identification in heterogeneous forests using aerial RGB imagery and deep learning. In: *Remote Sensing* 15 (2023), Nr. 5, S. 1463

[6] Berger, Marcel: *Geometry i*. Springer Science & Business Media, 2009

[7] Bharati, Puja ; Pramanik, Ankita: Deep learning techniques—R-CNN to mask R-CNN: a survey. In: *Computational Intelligence in Pattern Recognition: Proceedings of CIPR 2019* (2020), S. 657–668

[8] Chaolin, GU ; Liya, WU ; Cook, Ian: Progress in research on Chinese urbanization. In: *Frontiers of Architectural Research* 1 (2012), Nr. 2, S. 101–149

[9] Dimoudi, A ; Kantzioura, A ; Zoras, S ; Pallas, C ; Kosmopoulos, P: Investigation of urban microclimate parameters in an urban center. In: *Energy and Buildings* 64 (2013), S. 1–9

[10] Dosovitskiy, Alexey ; Beyer, Lucas ; Kolesnikov, Alexander ; Weissenborn, Dirk ; Zhai, Xiaohua ; Unterthiner, Thomas ; Dehghani, Mostafa ; Minderer, Matthias ; Heigold, Georg ; Gelly, Sylvain u. a.: An image is worth 16x16 words: Transformers for image recognition at scale. In: *arXiv preprint arXiv:2010.11929* (2020)

[11] Du, Siqi ; Tang, Shengjun ; Wang, Weixi ; Li, Xiaoming ; Guo, Renzhong: Tree-GPT: Modular Large Language Model Expert System for Forest Remote Sensing Image Understanding and Interactive Analysis. In: *arXiv preprint arXiv:2310.04698* (2023)

[12] Fu, Cheng-Yang ; Shvets, Mykhailo ; Berg, Alexander C.: RetinaMask: Learning to predict masks improves state-of-the-art single-shot detection for free. In: *arXiv preprint arXiv:1901.03353* (2019)

[13] Georgi, Neratzia J. ; Zafiriadis, K: The impact of park trees on microclimate in urban areas. In: *Urban Ecosystems* 9 (2006), S. 195–209

[14] Haase, Dagmar ; Güneralp, Burak ; Dahiya, Bharat ; Bai, Xuemei ; Elmqvist, Thomas u. a.: Global urbanization. In: *The Urban Planet: Knowledge Towards Sustainable Cities* 19 (2018), S. 326–339

[15] Hajela, Ruchi: Shortage of skilled workers: A paradox of the Indian economy. (2012)

[16] He, Kaiming ; Chen, Xinlei ; Xie, Saining ; Li, Yanghao ; Dollár, Piotr ; Girshick, Ross: Masked autoencoders are scalable vision learners. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, S. 16000–16009

[17] He, Kaiming ; Chen, Xinlei ; Xie, Saining ; Li, Yanghao ; Dollár, Piotr ; Girshick, Ross: Masked autoencoders are scalable vision learners. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, S. 16000–16009

[18] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, S. 770–778

[19] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: *Deep Residual Learning for Image Recognition*. 2016. – URL http://image-net.org/challenges/LSVRC/2015/

[20] Henderson, Vernon: Urbanization in developing countries. In: *The world bank research observer* 17 (2002), Nr. 1, S. 89–112

[21] Hosang, Jan ; Benenson, Rodrigo ; Schiele, Bernt: Learning non-maximum suppression. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, S. 4507–4515

[22] Hosmer Jr, David W. ; Lemeshow, Stanley ; Sturdivant, Rodney X.: *Applied logistic regression*. John Wiley & Sons, 2013

[23] Hossain, Mohammad D. ; Chen, Dongmei: Segmentation for Object-Based Image Analysis (OBIA): A review of algorithms and challenges from remote sensing perspective. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 150 (2019), S. 115–134

[24] Ishtiaque, Asif ; Masrur, Arif ; Rabby, Yasin W. ; Jerin, Tasnuba ; Dewan, Ashraf: Remote sensing-based research for monitoring progress towards SDG 15 in Bangladesh: A review. In: *Remote sensing* 12 (2020), Nr. 4, S. 691

[25] Ke, Lei ; Ye, Mingqiao ; Danelljan, Martin ; Tai, Yu-Wing ; Tang, Chi-Keung ; Yu, Fisher u. a.: Segment anything in high quality. In: *Advances in Neural Information Processing Systems* 36 (2024)

[26] Kirchenbauer, Vera: *Straßenbaumkataster Hamburg*. Jun 2024. – URL https://metaver.de/trefferanzeige?docuuid=C1C61928-C602-4E37-AF31-2D23901E2540

[27] Kirillov, Alexander ; Mintun, Eric ; Ravi, Nikhila ; Mao, Hanzi ; Rolland, Chloe ; Gustafson, Laura ; Xiao, Tete ; Whitehead, Spencer ; Berg, Alexander C. ; Lo, Wan-Yen ; Dollár, Piotr ; Girshick, Ross: Segment Anything. (2023), 4. – URL https://arxiv.org/abs/2304.02643v1

[28] Koonce, Brett ; Koonce, Brett: ResNet 50. In: *Convolutional neural networks with swift for tensorflow: image recognition and dataset categorization* (2021), S. 63–72

[29] Kraus, Karl ; Pfeifer, Norbert: Advanced DTM generation from LIDAR data. In: *International Archives Of Photogrammetry Remote Sensing And Spatial Information Sciences* 34 (2001), Nr. 3/W4, S. 23–30

[30] Lakshmanan, Valliappa ; Görner, Martin ; Gillard, Ryan: *Practical machine learning for computer vision*. Ö'Reilly Media, Inc.", 2021

[31] Lin, Tsung-Yi ; Dollár, Piotr ; Girshick, Ross ; He, Kaiming ; Hariharan, Bharath ; Belongie, Serge: Feature pyramid networks for object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, S. 2117–2125

[32] Lin, Tsung Y. ; Goyal, Priya ; Girshick, Ross ; He, Kaiming ; Dollar, Piotr: Focal Loss for Dense Object Detection. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42 (2017), 8, S. 318–327. – URL https://arxiv.org/abs/1708.02002v2. – ISSN 19393539

[33] Llames, Hellen G.: *Understanding zoom level in maps and imagery*. Jun 2024. – URL https://support.plexearth.com/hc/en-us/articles/6325794324497-Understanding-Zoom-Level-in-Maps-and-Imagery

[34] Lv, Jinna ; Shen, Qi ; Lv, Mingzheng ; Li, Yiran ; Shi, Lei ; Zhang, Peiying: Deep learning-based semantic segmentation of remote sensing images: a review. In: *Frontiers in Ecology and Evolution* 11 (2023), S. 1201125

[35] Lyons, Angela ; Grable, John ; Zeng, Ting: Infrastructure, urbanization, and the financial inclusion of Chinese households. In: *Available at SSRN 3012453* (2017)

[36] Makinde, Olusola O.: Urbanization, housing and environment: Megacities of Africa. In: *International Journal of Development and Sustainability* 1 (2012), Nr. 3, S. 976–993

[37] Mandrekar, Jayawant N.: Receiver operating characteristic curve in diagnostic test assessment. In: *Journal of Thoracic Oncology* 5 (2010), Nr. 9, S. 1315–1316

[38] Michel, Jean-Pierre ; Ecarnot, Fiona: The shortage of skilled workers in Europe: its impact on geriatric medicine. In: *European geriatric medicine* 11 (2020), Nr. 3, S. 345–347

[39] Okwuashi, Onuwa ; McConchie, Jack ; Nwilo, Peter ; Eyo, Etim: The challenge of urbanization. In: *Proceedings of DevNet Conference, Wellington, New Zealand*, 2008

[40] Perkins, S. ; Walker, A. ; Wolfart, E.: *Image Synthesis - Noise generation*. 2003. – URL https://homepages.inf.ed.ac.uk/rbf/HIPR2/noise.htm

[41] Radford, Alec ; Kim, Jong W. ; Hallacy, Chris ; Ramesh, Aditya ; Goh, Gabriel ; Agarwal, Sandhini ; Sastry, Girish ; Askell, Amanda ; Mishkin, Pamela ; Clark, Jack u. a.: Learning transferable visual models from natural language supervision. In: *International conference on machine learning* PMLR (Veranst.), 2021, S. 8748–8763

[42] Research, Wolfram: *CosineDistance*. Oct 2007. – URL https://reference.wolfram.com/language/ref/CosineDistance.html

[43] Ross, T-YLPG ; Dollár, GKHP: Focal loss for dense object detection. In: *proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, S. 2980–2988

[44] Sivanandam, Poornima ; Lucieer, Arko: Tree Detection and Species Classification in a Mixed Species Forest Using Unoccupied Aircraft System (UAS) RGB and Multispectral Imagery. In: *Remote Sensing* 14 (2022), Nr. 19, S. 4963

[45] Slater, James A. ; Malys, Stephen: Wgs 84—past, present and future. In: *Advances in Positioning and Reference Frames: IAG Scientific Assembly Rio de Janeiro, Brazil, September 3–9, 1997*. Springer, 1998, S. 1–7

[46] Spence, Michael ; Annez, Patricia C. ; Buckley, Robert M.: *Urbanization and growth*. World Bank Publications, 2008

[47] Sudre, Carole H. ; Li, Wenqi ; Vercauteren, Tom ; Ourselin, Sebastien ; Jorge Cardoso, M: Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3* Springer (Veranst.), 2017, S. 240–248

[48] Sukanya, R ; Tantia, Veerta: Urbanization and the impact on economic development. In: *New Perspectives and Possibilities in Strategic Management in the 21st Century: Between Tradition and Modernity*. IGI Global, 2023, S. 369–408

[49] Sultan, Rafi I. ; Li, Chengyin ; Zhu, Hui ; Khanduri, Prashant ; Brocanelli, Marco ; Zhu, Dongxiao: GeoSAM: Fine-tuning SAM with sparse and dense visual prompting for automated segmentation of mobility infrastructure. In: *arXiv preprint arXiv:2311.11319* (2023)

[50] Tharwat, Alaa: Classification assessment methods. In: *Applied computing and informatics* 17 (2021), Nr. 1, S. 168–192

[51] TrickyFoxy: *API v0.6 - OpenStreetMap*. Oct 2022. – URL https://wiki.openstreetmap.org/wiki/API_v0.6

[52] Vailshery, Lionel S. ; Jaganmohan, Madhumitha ; Nagendra, Harini: Effect of street trees on microclimate and air pollution in a tropical city. In: *Urban forestry & urban greening* 12 (2013), Nr. 3, S. 408–415

[53] Vaswani, A: Attention is all you need. In: *Advances in Neural Information Processing Systems* (2017)

[54] Weinstein, Ben G. ; Marconi, Sergio ; Aubry-Kientz, Mélaine ; Vincent, Gregoire ; Senyondo, Henry ; White, Ethan P.: DeepForest: A Python package for RGB deep learning tree crown delineation. In: *Methods in Ecology and Evolution* 11 (2020), 12, S. 1743–1751. – URL https://onlinelibrary.wiley.com/doi/full/10.1111/2041-210X.13472. – ISSN 2041-210X

[55] Weiss, Karl ; Khoshgoftaar, Taghi M. ; Wang, DingDing: A survey of transfer learning. In: *Journal of Big data* 3 (2016), S. 1–40

[56] Weisstein, Eric: *Mercator Projection*. Aug 2024. – URL https://mathworld.wolfram.com/MercatorProjection.html

[57] Willis, Katherine J. ; Petrokofsky, Gillian: The natural capital of city trees. In: *Science* 356 (2017), Nr. 6336, S. 374–376

[58] Wołk, Krzysztof ; Tatara, Marek: A Comprehensive Review of Semantic Segmentation and Instance Segmentation in Forestry: Advances, Challenges, and Applications. (2024)

[59] Yuan, Xiaohui ; Shi, Jianfang ; Gu, Lichuan: A review of deep learning methods for semantic segmentation of remote sensing imagery. In: *Expert Systems with Applications* 169 (2021), S. 114417

[60] Zhao, Zhuoyi ; Fan, Chengyan ; Liu, Lin: *Geo SAM: A QGIS plugin using Segment Anything Model (SAM) to accelerate geospatial image segmentation*. Juli 2023. – URL https://doi.org/10.5281/zenodo.8191039

A Anhang

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Masterarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
LaTeX	Erstellung des Dokuments
QGIS	Analyse räumlicher Daten
Python	Datenverarbeitung und Modellentwicklung
Google Colab	Cloud-basierte Entwicklung
Wolfram Mathematica	Mathematische Berechnungen
Writefull	Textkorrektur
Segment Anything Model (SAM)	Segmentierung von Baumkronen
CLIP	Verbesserung der Segmentierung
DeepForest	Segmentierung von Baumkronen
DeepL	Übersetzungen von Texten
OSMnx	Abfrage geographischer Daten
Pandas	Datenanalyse
Matplotlib	Visualisierungen
Scikit-learn	Berechnung von Metriken
GeoPandas	Geodatenverarbeitung
PyTorch	Modellausführung auf GPU
Git	Versionskontrolle
Overleaf	Erstellung des LaTeX-Dokuments
Grammarly	Unterstützung bei Formulierung
Language Tool	Unterstützung bei Formulierung
Wikipedia	Klärung von Verständnisfragen
Medium	Klärung von Verständnisfragen
Google One	Speicherung von Ergebnissen
Google Scholar	Hilfe bei der Suche nach Literatur
OpenRead Search	Hilfe bei der Suche nach Literatur
GoThesis	Hilfe bei Strukturierung und Literaturrecherche
Smodin	Plagiatscheck
Scribbr	Plagiatscheck und Formulierungshilfe
draw.io	Erstellung von Diagrammen
mentorium	Lektorat und Plagiatscheck
OpenAi	Lektorat und Unterstützung der Textstruktur

A.2 Ergebnisse aller Diagramme

In diesem Abschnitt werden die Ergebnisse der Diagramme (TP%, FP%, Precision-Recall, ROC) für die beiden Modelle SAM und DeepForest in verschiedenen Städten und unter verschiedenen Schwierigkeitsgraden einzeln dargestellt.

Die Tabellen stellen die Ergebnisse der beiden Modelle SAM und DeepForest in vier Städten (Hamburg, Kapstadt, San Francisco und Tokio) unter drei Schwierigkeitsgraden (nicht eng, eng, ineinander) getrennt nach TP, FP, Precision-Recall und ROC dar.

A.2.1 Tabelle für TP

Tabelle A.2: Zusammenfassung der TP Ergebnisse für SAM und DeepForest in verschiedenen Städten

Stadt	Modell	Schwierigkeitsgrad	TP
Hamburg	**SAM**	Nicht eng	Median höher, Mittelwert gleich
		Eng	Median und Mittelwert deutlich besser
		Ineinander	Median und Mittelwert deutlich besser
	DeepForest	Nicht eng	Median niedriger, Mittelwert gleich
		Eng	Median und Mittelwert deutlich schlechter
		Ineinander	Median und Mittelwert deutlich schlechter
Kapstadt	**SAM**	Nicht eng	Schlechter
		Eng	Schlechter
		Ineinander	Deutlich schlechter
	DeepForest	Nicht eng	Besser
		Eng	Besser
		Ineinander	Deutlich besser
San Francisco	**SAM**	Nicht eng	Etwas schlechter
		Eng	Fast gleich, weniger Streuung
		Ineinander	Schlechter, breite Streuung
	DeepForest	Nicht eng	Etwas besser, breite Streuung
		Eng	Fast gleich
		Ineinander	Besser
Tokio	**SAM**	Nicht eng	Besser, geringe Streuung
		Eng	Besser, hohe Streuung
		Ineinander	Besser
	DeepForest	Nicht eng	Schlechter
		Eng	Schlechter
		Ineinander	Schlechter

A.2.2 Tabelle für FP

Tabelle A.3: Zusammenfassung der FP Ergebnisse für SAM und DeepForest in verschiedenen Städten

Stadt	Modell	Schwierigkeitsgrad	FP
Hamburg	**SAM**	Nicht eng	Median und Mittelwert deutlich besser
		Eng	Median gleich, Mittelwert schlechter
		Ineinander	Median und Mittelwert besser
	DeepForest	Nicht eng	Median und Mittelwert schlechter
		Eng	Median gleich, Mittelwert besser
		Ineinander	Median und Mittelwert schlechter
Kapstadt	**SAM**	Nicht eng	Besser
		Eng	Besser
		Ineinander	Fast gleich
	DeepForest	Nicht eng	Schlechter
		Eng	Schlechter
		Ineinander	Fast gleich
San Francisco	**SAM**	Nicht eng	Etwas schlechter, geringe Streuung
		Eng	Fast gleich, geringe Streuung
		Ineinander	Etwas schlechter
	DeepForest	Nicht eng	Etwas besser
		Eng	Fast gleich, größere Streuung
		Ineinander	Etwas besser
Tokio	**SAM**	Nicht eng	Etwas schlechter
		Eng	Besser
		Ineinander	Schlechter
	DeepForest	Nicht eng	Etwas besser
		Eng	Schlechter, hohe Streuung
		Ineinander	Besser

A.2.3 Tabelle für Precision-Recall

Tabelle A.4: Zusammenfassung der Precision-Recall Ergebnisse für SAM und DeepForest in verschiedenen Städten

Stadt	Modell	Schwierigkeitsgrad	Precision-Recall
Hamburg	**SAM**	Nicht eng	Etwas besser
		Eng	Etwas besser
		Ineinander	Etwas besser, fast gleich
	DeepForest	Nicht eng	Schlechter
		Eng	Schlechter
		Ineinander	Schlechter
Kapstadt	**SAM**	Nicht eng	Besser
		Eng	Besser
		Ineinander	Besser
	DeepForest	Nicht eng	Schlechter
		Eng	Schlechter
		Ineinander	Schlechter
San Francisco	**SAM**	Nicht eng	Deutlich schlechter
		Eng	Fast gleich
		Ineinander	Fast gleich, schlechter bei hohem Recall
	DeepForest	Nicht eng	Deutlich besser
		Eng	Fast gleich, stabiler
		Ineinander	Fast gleich, stabiler bei hohem Recall
Tokio	**SAM**	Nicht eng	Schlechter
		Eng	Besser
		Ineinander	Fast gleich
	DeepForest	Nicht eng	Besser
		Eng	Schlechter
		Ineinander	Fast gleich

A.2.4 Tabelle für ROC

Tabelle A.5: Zusammenfassung der ROC Ergebnisse für SAM und DeepForest in verschiedenen Städten

Stadt	Modell	Schwierigkeitsgrad	ROC
Hamburg	**SAM**	Nicht eng	Deutlich besser, steilerer Anstieg
		Eng	Deutlich besser
		Ineinander	Etwas besser bei höheren FPR
	DeepForest	Nicht eng	Deutlich schlechter
		Eng	Deutlich schlechter
		Ineinander	Etwas schlechter
Kapstadt	**SAM**	Nicht eng	Besser, steilerer Anstieg
		Eng	Besser, steilerer Anstieg
		Ineinander	Besser, steilerer Anstieg
	DeepForest	Nicht eng	Schlechter
		Eng	Schlechter
		Ineinander	Schlechter
San Francisco	**SAM**	Nicht eng	Fast gleich
		Eng	Besser
		Ineinander	Fast gleich
	DeepForest	Nicht eng	Fast gleich
		Eng	Schlechter
		Ineinander	Fast gleich
Tokio	**SAM**	Nicht eng	Schlechter
		Eng	Besser
		Ineinander	Besser
	DeepForest	Nicht eng	Besser
		Eng	Schlechter
		Ineinander	Schlechter

A.3 Ergebnisse CLIP

Das hier abgebildete Bild stellt die Ergebnisse des CLIP-Modells nach der Feinabstimmung auf *Baum* und *kein Baum* dar.

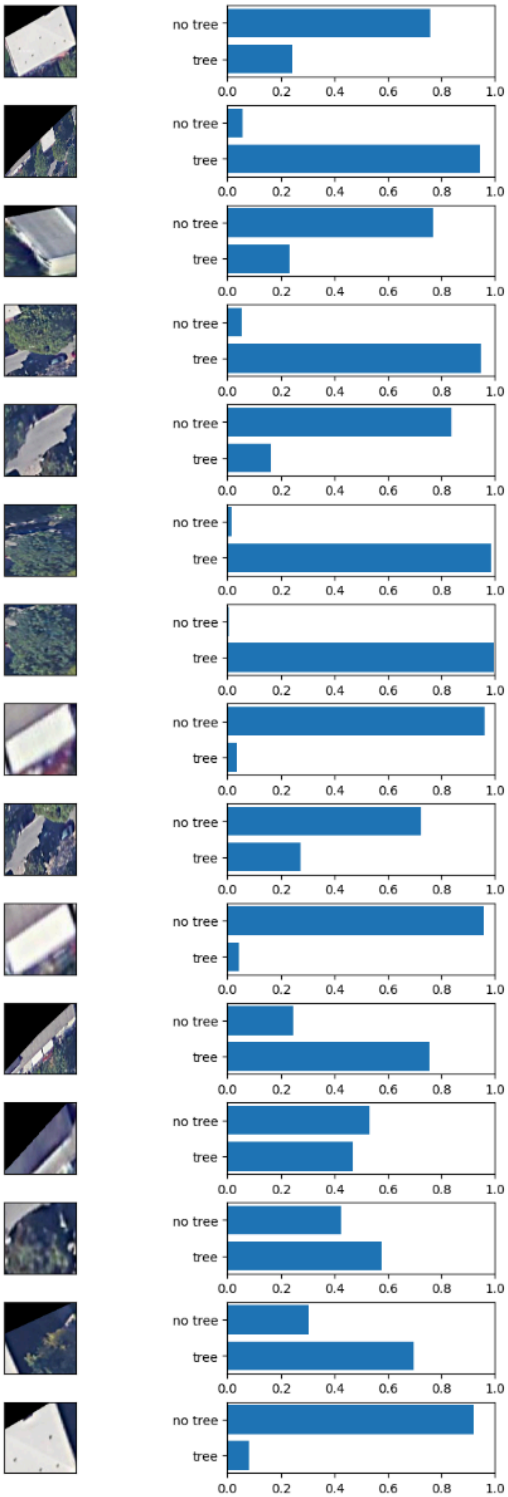

Abbildung A.1: Ergebnisse der Vorhersagewahrscheinlichkeiten bei einer zufälligen Auswahl an Segmentierungen innerhalb des Testgebiets.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Hamburg	29. Oktober 2024	
Ort	Datum	Unterschrift im Original