

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Objektbasierte und pixelbasierte Erkennung sowie Klassifizierung von
Photovoltaikanlagen auf einem hochauflösenden Orthomosaik mithilfe von
Deep Learning-Ansätzen

verfasst von | submitted by

Tünay Ismail BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt |
Degree programme code as it appears on the
student record sheet:

UA 066 856

Studienrichtung lt. Studienblatt | Degree
programme as it appears on the student
record sheet:

Masterstudium Kartographie und Geoinformation

Betreut von | Supervisor:

Ass.-Prof. Mag. Dr. Andreas Riedl

Nutzung geschlechterneutraler Sprache

In dieser Masterarbeit wird auf das Gendern im Sinne sprachlicher Sonderzeichen oder Doppelnennungen bewusst verzichtet. Stattdessen wird das generische Maskulinum verwendet. Diese Entscheidung dient dem Ziel, den Lesefluss zu fördern und die sprachliche Klarheit zu wahren. Die gewählte Formulierung soll eine neutrale, gut verständliche Darstellung ermöglichen, bei der der Fokus auf dem wissenschaftlichen Inhalt liegt. Selbstverständlich sind alle Geschlechter gleichermaßen mitgemeint.

Kurzfassung

Diese Masterarbeit untersucht den Vergleich zwischen Objekterkennung und Bildsegmentierung zur automatisierten Identifikation von Photovoltaikanlagen. Ziel ist es, herauszufinden, welcher Ansatz, basierend auf Deep Learning, eine höhere Erkennungsgenauigkeit bietet. Dafür werden zwei verschiedene Convolutional Neural Networks (CNNs) eingesetzt: Faster R-CNN für die Objekterkennung und Mask R-CNN für die Instanzsegmentierung. Beide Modelle werden jeweils auf zwei Datensätzen trainiert - einem Einzelklassendatensatz (nur Photovoltaikanlagen) sowie einem kombinierten Datensatz (Photovoltaik- und Solarthermieranlagen). Als Grundlage für Trainings- und Testdaten dient ein Orthomosaik mit einer Bodenauflösung von 0,2m, das in ArcGIS Pro verarbeitet wurde. Die Ergebnisse werden mithilfe einer visuellen Analyse, Konfusionsmatrizen und der Intersection over Union (IoU) bewertet. Dabei zeigt sich, dass segmentierungsbasierte Ansätze insgesamt präzisere Ergebnisse liefern – insbesondere bei der Erkennung von Photovoltaikanlagen. Die Erkennung von Solarthermieranlagen erwies sich hingegen als weniger zuverlässig. Die Arbeit liefert einen praxisnahen Vergleich aktueller Deep Learning-Modelle zur Fernerkundung solarer Infrastrukturen und zeigt auf, welche Modellart und Trainingsstrategie besonders geeignet ist, um Photovoltaikanlagen automatisiert und zuverlässig zu erkennen.

Abstract

This master's thesis investigates the comparison between object detection and image segmentation for the automated identification of photovoltaic systems. The objective is to determine which deep learning-based approach offers higher detection accuracy. Two different convolutional neural networks (CNNs) are employed: Faster R-CNN for object detection and Mask R-CNN for instance segmentation. Both models are trained on two datasets – a single-class dataset (photovoltaic systems only) and a combined dataset (photovoltaic and solar thermal systems). An orthomosaic with a ground resolution of 0.2m, processed in ArcGIS Pro, serves as the basis for training and testing. The results are evaluated using visual analysis, confusion matrices, and Intersection over Union (IoU). The findings show that segmentation-based approaches generally provide more accurate results – particularly in the detection of photovoltaic systems. In contrast, the detection of solar thermal systems proved to be less reliable. This work provides a practical comparison of current deep learning models for the remote sensing of solar infrastructure and highlights which model type and training strategy are especially suitable for the automated and reliable detection of photovoltaic systems.

Danksagung

An dieser Stelle möchte ich mich herzlich bei meinem Betreuer, Professor Andreas Riedl, bedanken. Durch seine Unterstützung, Geduld und Expertise war er mir stets eine wertvolle Hilfe bei der Erstellung dieser Masterarbeit und nahm sich auch immer die Zeit, meine Fragen zu beantworten. Besonders schätze ich, dass er mir stets mit Verständnis und Ruhe zur Seite stand, wodurch ich die Arbeit in meinem eigenen Tempo abschließen konnte. Dafür bin ich sehr dankbar.

Inhaltsverzeichnis

Kurzfassung	iv
Abstract	v
Inhaltsverzeichnis	vii
Abbildungsverzeichnis	xi
Tabellenverzeichnis	xiv
Abkürzungsverzeichnis	xv
1 Einleitung	1
1.1 Motivation, Problemstellung und Hintergrund	1
1.2 Stand der Forschung	2
1.3 Ziel der Arbeit, Forschungsfrage und begleitende Arbeitsfragen	3
1.4 Methodische Vorgehensweise	4
1.5 Aufbau der Arbeit	5
2 Energieversorgung und -industrie in Österreich	6
2.1 Ziele und Strategien für eine nachhaltige Energieversorgung	6
2.2 Marktentwicklung der Photovoltaik im Jahr 2023	7
2.2.1 Anlagen-, Montageart und Solarzellentypen	8
2.3 Zukünftige Aussichten	9
3 Künstliche Intelligenz: Ein Überblick	10
3.1 Geschichtliche Entwicklung der Künstlichen Intelligenz	11
3.1.1 Die 1940er-Jahre	11
3.1.2 Die Jahre 1950 bis 1970	11
3.1.3 Die 1960er- bis 1980er-Jahre	12
3.1.4 Mitte der 1980er-Jahre	13
3.1.5 Die 1990er-Jahre	14
3.1.6 2006 bis zur Gegenwart	14
3.1.7 Künstliche Intelligenz in der Geographie	16
3.2 Grundlagen von Machine Learning	16
3.2.1 Lernmethoden im Machine Learning	18
3.2.2 Überwachtes Lernen im Detail	19
4 Artificial Neural Networks (ANN)	21
4.1 Aufbau von Artificial Neural Networks	21
4.2 Entwicklung und Fortschritte in der Technologie	23
4.2.1 Erfolgsfaktoren für Artificial Neural Networks und Deep Learning	24
5 Deep Learning	27
5.1 Funktionsweise von Deep Learning	27
5.2 Convolutional Neural Networks (CNN)	28

5.3	Computer Vision	29
5.3.1	Bildklassifizierung	29
5.3.2	Objekterkennung	30
5.3.3	Semantische Segmentierung und Instanzsegmentierung	31
5.4	Digitale Bilddarstellung in der Computer Vision	32
5.4.1	Graustufenbilder	33
5.4.2	RGB-Bilder	33
5.5	Architektur eines Convolutional Neural Networks	34
5.5.1	Input Layer	35
5.5.2	Convolutional Layer	35
5.5.3	Pooling Layer	36
5.5.4	Fully-Connected Layer	37
5.6	Der Lernprozess eines Künstlichen Neuronales Netzes (KNN)	38
5.6.1	Forward Propagation, Backpropagation und Verlustfunktion	38
5.6.2	Aktivierungsfunktionen	39
5.6.3	Batch, Iteration und Epoche	39
5.6.4	Gradientenverfahren	39
5.7	Zu beachtende Herausforderungen	40
5.7.1	Overfitting	41
5.7.2	Underfitting	41
5.7.3	Verschwindende und explodierende Gradienten	41
5.8	Forschung und verwandte Arbeiten	42
5.8.1	Ableitende Erkenntnisse aus den vorgestellten Forschungen	43
6	Praktische Herangehensweise	46
6.1	Begründung der Softwarewahl	46
6.2	Datengrundlage	47
6.2.1	Untersuchungsgebiet	47
6.2.2	Orthofotos	48
6.3	Eingesetzte Modelle und Netzwerkarchitekturen	49
6.3.1	Faster R-CNN	51
6.3.1.1	Architektur von Faster R-CNN	51
6.3.1.2	Vor- und Nachteile von Faster R-CNN	52
6.3.2	Mask R-CNN	52
6.3.2.1	Architektur von Mask R-CNN	52
6.3.2.2	Vor- und Nachteile von Mask R-CNN	53
6.4	Grundvoraussetzungen für die praktische Umsetzung	53
6.5	Klassifikation von Trainingsdaten	55
6.5.1	Export der erzeugten Trainingsdaten	60
6.6	Datenvorverarbeitung	63
6.6.1	Datenaugmentationsmethode Rotation	63
6.6.2	Vorverarbeitung der Trainingsdaten	65
6.6.3	Erste Arbeitsschritte in ArcGIS Pro	67
6.6.4	Datenaugmentationsmethoden in prepare_data	67
6.7	Training der Daten	68
6.7.1	Faster R-CNN – Datensatz 1	69
6.7.1.1	Faster R-CNN: Training über 10 Epochen ohne Transformationen	72

6.7.1.2	Hyperparameter-Tuning: Training über 10 Epochen inkl. Transformationen	75
6.7.1.3	Hyperparameter-Tuning: Training über 10 Epochen inkl. Transformationen und Lernrate von 0.001	76
6.7.1.4	Hyperparameter-Tuning für Faster R-CNN: Training über 30 Epochen ohne Transformationen	77
6.7.1.5	Hyperparameter-Tuning für Faster R-CNN: Training über 30 Epochen inkl. Transformationen	79
6.7.1.6	Hyperparameter-Tuning für Faster R-CNN: Training über 30 Epochen inkl. Transformationen und Lernrate von 0.001	81
6.7.2	Faster R-CNN – Kombierter Datensatz	82
6.7.2.1	Faster R-CNN: Training über 10 Epochen ohne Transformationen	84
6.7.2.2	Hyperparameter-Tuning für Faster R-CNN: Training über 10 Epochen inkl. Transformationen	88
6.7.3	Mask R-CNN – Datensatz 1	89
6.7.3.1	Mask R-CNN: Training über 10 Epochen ohne Transformationen	91
6.7.3.2	Hyperparameter-Tuning für Mask R-CNN: Training über 10 Epochen inkl. Transformationen	94
6.7.3.3	Hyperparameter-Tuning für Mask R-CNN: Training über 30 Epochen ohne Transformationen	95
6.7.3.4	Hyperparameter-Tuning für Mask R-CNN: Training über 30 Epochen inkl. Transformationen und Lernrate von 0.001	97
6.7.4	Mask R-CNN – Kombierter Datensatz	98
6.7.4.1	Mask R-CNN: Training über 10 Epochen ohne Transformationen	99
6.7.4.2	Hyperparameter-Tuning für Mask R-CNN: Training über 10 Epochen inkl. Transformationen	102
6.7.4.3	Hyperparameter-Tuning für Mask R-CNN: Training über 10 Epochen inkl. Transformationen und Lernrate von 0.001	103
6.7.5	Zusätzliche Erhebung von Informationen zur Grafikkarte	105
7	Trainingsresultate – Erkennung der Solaranlagen	106
7.1	Vorstellung des Testgebiets	108
7.1.1	Erste visuelle Erkennungsergebnisse	111
7.2	Auswertung der Modellleistung	114
7.2.1	Precision, Recall und F1-Score	115
7.2.1.1	Precision (dt.: Präzision)	115
7.2.1.2	Recall (dt.: Trefferquote)	115
7.2.1.3	F1-Score (dt.: F1-Maß)	115
7.2.2	Evaluierung der Modellleistung mittels Konfusionsmatrix	115
7.2.2.1	Mask R-CNN – Kombierter Datensatz	116
7.2.2.2	Mask R-CNN – Datensatz 1	117
7.2.2.3	Faster R-CNN – Kombierter Datensatz	117
7.2.2.4	Faster R-CNN – Datensatz 1	118
7.2.3	Evaluierung der Modellleistung mit Intersection over Union (IoU)	120
7.2.3.1	Durchführung der Bewertung in ArcGIS Pro	120
7.2.3.2	Vergleich der IoU-Werte auf dem Datensatz 1	121
7.2.3.3	Vergleich der IoU-Werte auf dem kombinierten Datensatz	122
7.3	Modellbewertung und Ergebnisinterpretation	123
7.3.1	Analyse der Klassifikationsmetriken (Konfusionsmatrix)	123
7.3.2	Analyse der Modelle anhand von IoU-Werten	124
7.3.3	Zusammenfassung der Ergebnisse	124
8	Diskussion der Ergebnisse	126
8.1	Beantwortung der Forschungsfrage	126

8.2	Beantwortung der Arbeitsfragen	126
8.2.1	Geeignete Deep Learning-Modelle	127
8.2.2	Entwicklungsumgebung und Umsetzung	127
8.2.3	Einflussfaktoren auf die Modellleistung	128
8.3	Stärken und Schwächen	129
8.3.1	Stärken und Schwächen der Modelle	129
8.3.2	Stärken und Schwächen der CV-Ansätze Objekterkennung und Bildsegmentierung	129
8.3.3	Methodenkritik und Grenzen der Arbeit	129
8.4	Positionierung im Forschungsfeld	130
8.4.1	Methodische Zielsetzung und Rahmenbedingungen der vorliegenden Masterarbeit	131
8.4.1.1	Vergleich der Klassifikationsergebnisse mit verwandten Studien	132
8.5	Forschungsempfehlungen und Ausblick	133
9	Fazit	136
	Literaturverzeichnis	138
	Hilfsmittelverzeichnis	143
	Anhang	144
	Anhang A: Umbenennen von Bildchips und Annotationen für Faster R-CNN	144
	Anhang B: Export von Trainingsdaten für Mask R-CNN (Kombinierter Datensatz)	147
	Anhang C: Vorbereiten und Trainieren von Trainingsdaten für Faster R-CNN Datensatz 1 (30 Epochen und Dataaugmentation)	149
	Anhang D: Vorbereiten und Trainieren von Trainingsdaten für Mask R-CNN – Kombiniertes Datensatz (10 Epochen und Dataaugmentation)	152
	Anhang E: Erkennen von Objekten mittels Deep Learning – Faster R-CNN (Datensatz 1)	155
	Anhang F: Modellbewertung mittels Konfusionsmatrix für Mask R-CNN	157
	Anhang G: Balkendiagramm für PVA aus den Ergebnissen der Konfusionsmatrix erstellen (Kombinierter Datensatz und Datensatz 1)	159
	Anhang H: Modellbewertung mittels Intersection over Union (IoU) für Faster R-CNN	160
	Anhang I: Balkendiagramme aus den Ergebnissen von IoU erstellen (Datensatz 1)	161

Abbildungsverzeichnis

Abb. 1: Monokristalline PV-Zellen	7
Abb. 2: Polykristalline PV-Zellen	7
Abb. 3: PV-Leistung in den Jahren 2000 – 2022	8
Abb. 4: Montageart der installierten PV-Anlagen der Jahre 2013 – 2022	8
Abb. 5: Solarzellentypen der installierten PV-Anlagen in den Jahren 2010 – 2022	8
Abb. 6: Künstliche Intelligenz und ihre Anwendungsbereiche	10
Abb. 7: Perzeptron	12
Abb. 8: ImageNet	15
Abb. 9: Machine Learning und seine Zweige	17
Abb. 10: Entwicklung von Machine Learning und Fernerkundung über die Jahre	19
Abb. 11: Vier-Schichtiges neuronales Netz	22
Abb. 12: Rekurrentes neuronales Netz	22
Abb. 13: NVIDIA-Grafikkarte	25
Abb. 14: Convolutional Neural Networks	28
Abb. 15: Bildklassifizierung inklusive Lokalisierung	30
Abb. 16: Objekterkennung	30
Abb. 17: Semantische Segmentierung	31
Abb. 18: Instanzsegmentierung	32
Abb. 19: Average-Pooling bei Grauwertbildern	33
Abb. 20: Average-Pooling bei RGB-Bildern	34
Abb. 21: Beispiel für Average-Pooling	34
Abb. 22: Average-Pooling	37
Abb. 23: MAX-Pooling	37
Abb. 24: Darstellung einer Aktivierungsfunktion	39
Abb. 25: Verlustfunktion mit Kurve	40
Abb. 26: Blöcke für die Luftbildaufnahme	48
Abb. 27: Orthomosaik-Ausschnitt „Nordburgenland“ aus dem Jahr 2022, eigene Bearbeitung	49
Abb. 28: Vergleich von ein- und zweistufigen Netzwerken in der Objekterkennung	51
Abb. 29: Architektur von Faster R-CNN	52
Abb. 30: Architektur von Mask R-CNN	53
Abb. 31: Erweiterungen Image Analyst und Spatial Analyst	54
Abb. 32: NVIDIA-Befehl zur Abfrage von GPU-Informationen	55
Abb. 33: Workflowdiagramm	57
Abb. 34: Gesamttrainingsgebiet, eigene Darstellung, erstellt mit ArcGIS Pro 3.2	58
Abb. 35: Werkzeug für die Klassifikation	59
Abb. 36: Unterschied zwischen Photovoltaik- und Solarthermieranlagen	60
Abb. 37: Beispiel eines Gebäudes mit PVA und dazu die Annotationen (v.l.n.r.)	62
Abb. 38: Beispiel von Aufdach-PVA und dazu die Annotationen (v.l.n.r.)	62
Abb. 39: Beispiel von Gebäuden mit PVA sowie STA und dazu die Annotationen (v.l.n.r.)	62
Abb. 40: Beispiel eines Gebäudes mit STA und dazu die Annotationen (v.l.n.r.)	63
Abb. 41: Rotation einer STA mit 0° (links oben), 90° (rechts oben), 180° (links unten) und 270° (rechts unten)	64
Abb. 42: Rotation einer PVA mit 0° (links) und 180° (rechts)	65
Abb. 43: Nachträgliche Anpassung der EMD-Datei für den kombinierten Datensatz	66
Abb. 44: Originalbild (oben links, unten links), Vertikaler Flip (oben mittig, unten mittig) und Helligkeitsanpassung (oben rechts, unten rechts)	68
Abb. 45: Stichprobenartige Anzeige der Annotationen auf den Bilddaten in Datensatz 1 (FasterRCNN, 10 Epochen, ohne Transformationen)	70
Abb. 46: Lernratenkurve von FasterRCNN in Datensatz 1 (10 Epochen, ohne Transformationen)	71
Abb. 47: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, ohne Transformationen)	72
Abb. 48: Verlustdiagramm von FasterRCNN von Datensatz 1I (10 Epochen, ohne Transformationen)	73
Abb. 49: Funktion show_results() für FasterRCNN mit Ground Truth (l.) und Vorhersagen (r.) (Datensatz 1, 10 Epochen, ohne Transformationen)	74

Abb. 50: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen) ...	75
Abb. 51: Verlustdiagramm von FasterRCNN von Datensatz 1 (10 Epochen, mit Transformationen)	75
Abb. 52: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen, Lernrate).....	76
Abb. 53: Verlustdiagramm von FasterRCNN von Datensatz 1 (30 Epochen, ohne Transformationen, Lernrate)	77
Abb. 54: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (30 Epochen, ohne Transformationen) .	78
Abb. 55: Verlustdiagramm von FasterRCNN von Datensatz 1 (30 Epochen, ohne Transformationen)	79
Abb. 56: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (30 Epochen, mit Transformationen) ...	80
Abb. 57: Verlustdiagramm von FasterRCNN von Datensatz 1 (30 Epochen, mit Transformationen)	80
Abb. 58: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (Lernrate 0.001, 30 Epochen, mit Transformationen)	81
Abb. 59: Verlustdiagramm von FasterRCNN von Datensatz 1 (Lernrate 0.001, 30 Epochen, mit Transformationen)	82
Abb. 60: Stichprobenartige Anzeige der Annotationen auf den Bilddaten im kombinierten Datensatz (FasterRCNN, 10 Epochen, ohne Transformationen).....	83
Abb. 61: Lernratenkurve von FasterRCNN im kombinierten Datensatz (10 Epochen, ohne Transformationen)	84
Abb. 62: FasterRCNN-Modellergebnis nach Training mit kombiniertem Datensatz (10 Epochen, ohne Transformationen)	84
Abb. 63: Verlustdiagramm von FasterRCNN vom kombinierten Datensatz (10 Epochen, ohne Transformationen)	85
Abb. 64: Average Precision-Ergebnis für beide Klassen (FasterRCNN, 10 Epochen, ohne Transformationen) ..	86
Abb. 65: Funktion show_results() für FasterRCNN mit Ground Truth (l.) und Vorhersagen (r.) (Kombinierter Datensatz, 10 Epochen, ohne Transformationen).....	87
Abb. 66: FasterRCNN-Modellergebnis nach Training mit kombiniertem Datensatz (10 Epochen, mit Transformationen)	88
Abb. 67: Verlustdiagramm von FasterRCNN vom kombinierten Datensatz (10 Epochen, mit Transformationen)	88
Abb. 68: Average Precision-Ergebnis für beide Klassen (FasterRCNN, 10 Epochen, mit Transformationen)	89
Abb. 69: Stichprobenartige Anzeige der Annotationen auf den Bilddaten in Datensatz 1 (MaskRCNN, 10 Epochen, ohne Transformationen)	90
Abb. 70: Lernratenkurve von MaskRCNN in Datensatz 1 (10 Epochen, ohne Transformationen).....	91
Abb. 71: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, ohne Transformationen) ..	91
Abb. 72: Verlustdiagramm von MaskRCNN vom Datensatz 1 (10 Epochen, ohne Transformationen).....	92
Abb. 73: Funktion show_results() für MaskRCNN mit Ground Truth (l.) und Vorhersagen (r.) (Datensatz 1, 10 Epochen, ohne Transformationen)	93
Abb. 74: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen)	94
Abb. 75: Verlustdiagramm von MaskRCNN vom Datensatz 1 (10 Epochen, mit Transformationen)	94
Abb. 76: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (30 Epochen, ohne Transformationen) ..	95
Abb. 77: Verlustdiagramm von MaskRCNN vom Datensatz 1 (30 Epochen, ohne Transformationen).....	96
Abb. 78: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen, Lernrate).....	97
Abb. 79: Verlustdiagramm von MaskRCNN von Datensatz 1 (30 Epochen, ohne Transformationen, Lernrate)	98
Abb. 80: MaskRCNN-Modellergebnis nach Training mit kombinierten Datensatz (10 Epochen, ohne Transformationen)	99
Abb. 81: Verlustdiagramm von MaskRCNN vom kombinierten Datensatz (10 Epochen, ohne Transformationen)	99
Abb. 82: Average Precision-Ergebnis für beide Klassen (MaskRCNN, 10 Epochen, ohne Transformationen) .	100
Abb. 83: Funktion show_results() für MaskRCNN mit Ground Truth (l.) und Vorhersagen (r.) (Kombinierter Datensatz, 10 Epochen, ohne Transformationen).....	101
Abb. 84: MaskRCNN-Modellergebnis nach Training mit kombinierten Datensatz (10 Epochen, mit Transformationen)	102
Abb. 85: Verlustdiagramm von MaskRCNN vom kombinierten Datensatz (10 Epochen, mit Transformationen)	102

Abb. 86: Average Precision-Ergebnis für beide Klassen (MaskRCNN, 10 Epochen, mit Transformationen)	103
Abb. 87: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen, Lernrate).....	103
Abb. 88: Verlustdiagramm von MaskRCNN von Datensatz 1 (30 Epochen, ohne Transformationen, Lernrate)	104
Abb. 89: Average Precision-Ergebnis für beide Klassen (MaskRCNN, 10 Epochen, mit Transformationen, Lernrate).....	105
Abb. 90: Ermittlung von GPU-Informationen.....	105
Abb. 91: Testgebiet „Hennersdorf“, eigene Darstellung, erstellt mit ArcGIS Pro 3.2.....	108
Abb. 92: Ground Truth-Daten 1/2.....	110
Abb. 93: Ground Truth-Daten 2/2.....	111
Abb. 94: Falsch als Photovoltaikanlage detektierte Objekte (Faster R-CNN).....	112
Abb. 95: Richtig als Photovoltaikanlage detektierte Objekte (Faster R-CNN).....	112
Abb. 96: Unzureichende Erkennung von Photovoltaikanlagen (Mask R-CNN).....	113
Abb. 97: Erkennung von Photovoltaikanlagen im kombinierten Datensatz (Mask R-CNN)	114
Abb. 98: Erkennung von Solarthermieranlagen im kombinierten Datensatz (Mask R-CNN).....	114
Abb. 99: Konfusionsmatrix des Modells Mask-RCNN (Kombinierter Datensatz)	116
Abb. 100: Konfusionsmatrix des Modells Mask-RCNN (Datensatz 1)	117
Abb. 101: Konfusionsmatrix des Modells Faster-RCNN (Kombinierter Datensatz)	118
Abb. 102: Konfusionsmatrix des Modells Faster-RCNN (Datensatz 1)	118
Abb. 103: Darstellung vom Vergleich der drei Metriken, bezogen auf PVA	119
Abb. 104: Darstellung der IoU-Ergebnisse der Modelle im Vergleich (Datensatz 1)	122
Abb. 105: Darstellung der IoU-Ergebnisse der Modelle im Vergleich (Kombinierter Datensatz)	123

Tabellenverzeichnis

Tab. 1: Vergleich verwandter Forschungen	44
Tab. 2: Vergleich unter zur Auswahl stehender, vortrainierter Modelle	50
Tab. 3: Gerät und Spezifikationen	53
Tab. 4: Genutzte Software und Versionen	54
Tab. 5: Software, die Mindestversionen erfordern sowie installierte Versionen	54
Tab. 6: Für das Training festgelegte Klassen	55
Tab. 7: Klassennamen und -werte	59
Tab. 8: Anzahl an Annotationen und Bildern	60
Tab. 9: Erläuterung der Parameter des Werkzeugs „Export Training Data for Deep Learning“	62
Tab. 10: Veranschaulichung der Anzahl an Bildchips und Annotationen beider Klassen	64
Tab. 11: Anzahl an Annotationen und Bildern nach der Anpassung als Endresultat	66
Tab. 12: Übersicht der Datenformate für Trainingsdaten	69
Tab. 13: Gegenüberstellung der Modellergebnisse	106
Tab. 14: Erläuterung der Parameter des Werkzeugs „Detect Objects Using Deep Learning“	108
Tab. 15: Ergebnisse der Klassifikationsmetriken für Mask R-CNN (Kombinierter Datensatz)	116
Tab. 16: Ergebnisse der Klassifikationsmetriken für Mask R-CNN (Datensatz 1)	117
Tab. 17: Ergebnisse der Klassifikationsmetriken für Faster R-CNN (Kombinierter Datensatz)	117
Tab. 18: Ergebnisse der Klassifikationsmetriken für Faster R-CNN (Datensatz 1)	118
Tab. 19: Gesamtergebnis der Klassifikationsmetriken (für alle Modelle)	119
Tab. 20: IoU-Ergebnisse (Datensätze 1)	121
Tab. 21: IoU-Ergebnisse (Kombinierte Datensätze)	121
Tab. 22: Vergleich der Studienergebnisse	132
Tab. 23: Hilfsmittelverzeichnis	143

Abkürzungsverzeichnis

AI	Artificial Intelligence
ANN	Artificial Neural Networks
AP	Average Precision
BEV	Bundesamt für Eich- und Vermessungswesen
CART	Classification and Regression Trees
CIR	Color Infrared
CNN	Convolutional Neural Networks
COG	Cloud Optimized GeoTIFF
CUDA	Compute Unified Device Architecture
CV	Computer Vision
DETReg	Detection with Region Priors
DL	Deep Learning
EAG	Erneuerbaren-Ausbau-Gesetz
FFNR	Feedforward Neural Networks
GAN	Generative Adversarial Networks
GEE	Google Earth Engine
HPC	High Performance Computing
ILSVRC	Imagenet Large Scale Visual Recognition Challenge
IoU	Intersection over Union
KI	Künstliche Intelligenz
KNN	Künstliche Neuronale Netze
mAP	Mean Average Precision
ML	Machine Learning
MLP	Multilayer Perceptron
OSM	OpenStreetMap
PVA	Photovoltaikanlage
PV-Anlage	Photovoltaikanlage
RGB	Red, Green, Blue
RNN	Recurrent Neural Networks
RoIAlign	Region-of-Interest Align
RPN	Region Proposal Network
SAE	Stacked Autocoder
SSD	SingleShotDetector
STA	Solarthermieanlage
VLSI	Very-Large-Scale Integrated Circuit
YOLO	You Only Live Once

1 Einleitung

Diese Arbeit behandelt die automatisierte Erkennung und Klassifikation von Photovoltaikanlagen mithilfe von Deep Learning-Verfahren im Bereich der Geoinformationsverarbeitung. Die Einleitung soll die Motivation, die Problemstellung, den Forschungsstand, die Zielsetzung und die Methodik der Arbeit präsentieren sowie einen Überblick über den Aufbau der Arbeit geben.

1.1 Motivation, Problemstellung und Hintergrund

Geoobjekte und Raumeinheiten spielen eine zentrale Rolle in der Geoinformationsverarbeitung. Sie ermöglichen die Identifikation geographischer Strukturen, sei es in urbanen Gebieten, in vom Klimawandel betroffenen Regionen oder für das Katastrophenmanagement, etwa zur Erkennung von Hochwasserzonen. Ebenso lassen sich Bergwerke, archäologische Stätten oder andere Objekte lokalisieren. Dank der zunehmenden Offenlegung von Daten, insbesondere Bilddaten, für die Öffentlichkeit wächst das Phänomen Big Data stetig. Diese Fülle an geographischen Informationen kann für verschiedene Aufgaben der Künstlichen Intelligenz (KI) genutzt werden, insbesondere in den Bereichen Machine Learning (ML), Deep Learning (DL) und Computer Vision (CV).

Im Zeitalter von Big Data gewinnt die effiziente Datenverarbeitung zunehmend an Bedeutung. Maschinen können Prozesse automatisieren und Arbeitsweisen erleichtern, wodurch sich der menschliche Aufwand erheblich reduziert. Künstliche Intelligenz (KI) spielt dabei eine immer größere Rolle – oft unbewusst im Alltag integriert, aber auch kontrovers diskutiert. Ein zentraler Begriff in diesem Zusammenhang ist Deep Learning, ein Teilgebiet des Machine Learning (ML), das sich auf künstliche neuronale Netze (KNN; Artificial Neural Networks, ANN) stützt. Eine spezielle Form dieser Netzwerke sind Convolutional Neural Networks (CNN), deren Funktionsweise vom menschlichen Gehirn inspiriert ist. CNNs finden Anwendung in der Analyse von Audio-, Bild- und Textdaten (Aggarwal, 2018; Jiang et al., 2019).

Ein Teilgebiet der KI ist das eingangs erwähnte Feld Computer Vision (CV; dt.: Maschinelles Sehen), das in der deutschen Fachsprache oft als Bildverarbeitung bezeichnet wird. Zu den Kernbereichen gehören die Bildklassifizierung (engl.: Image Classification), Objekterkennung (engl.: Object Detection) und Bildsegmentierung (engl.: Image Segmentation). Die Bildsegmentierung kann weiter unterteilt werden in Semantische Segmentierung (engl.: Semantic Segmentation), Instanzsegmentierung (engl.: Instance Segmentation) und Panoptische Segmentierung (engl.: Panoptic Segmentation) (Kumar & Renuka, 2023). Die Objekterkennung geht über die einfache Bildlokalisierung hinaus. Sie bestimmt nicht nur die Position eines oder mehrerer Objekte mittels *Bounding Boxes*, sondern ordnet sie auch einer bestimmten Klasse zu. Dies ist besonders für die schnelle Echtzeiterkennung beweglicher Objekte (z. B. Fahrzeuge) relevant. Die Bildsegmentierung hingegen identifiziert nicht nur Objekte, sondern ordnet jedem Pixel eine spezifische Kategorie zu. Dadurch entstehen detaillierte Masken für jedes Objekt im Bild. Bei der semantischen Segmentierung wird jedes Pixel eines Bildes einer Klasse zugeordnet, ohne zwischen einzelnen Instanzen zu unterscheiden. Eine Instanz bezeichnet die konkrete Ausprägung eines Objekts innerhalb einer Klasse. Die Instanzsegmentierung hingegen differenziert zwischen Objekten derselben Klasse und behandelt sie als separate Instanzen. Sie kombiniert somit Elemente der semantischen Segmentierung und der Objekterkennung (Aggarwal, 2018; Bandyopadhyay, 2023).

Der Fokus dieser Masterarbeit liegt auf der Identifikation von Photovoltaikanlagen (PV-Anlagen, PVA), die als erneuerbare Energiequelle stark nachgefragt sind. Aufgrund der hohen Anschaffungskosten gibt es in Österreich Förderprogramme, die von der Bevölkerung stark genutzt werden (Frei, 2023). Laut einem Bericht des Bundesamts für Klima, Umwelt, Energie, Mobilität, Innovation und Technologie aus dem Jahr 2022 stieg der Anteil der Photovoltaik an der österreichischen Stromerzeugung auf über 6%. Besonders im Jahr 2021 wurden viele neue Anlagen ausgebaut und installiert, wodurch die kumulierte Gesamtleistung auf 3,8 GW_{peak} anstieg. Das Ausbauziel für 2030 liegt bei 13GW_{peak} – ein Zeichen für das hohe Potenzial der Photovoltaik in der Energiewende (Bundesministerium für Klimaschutz, 2023).

In Österreich gibt es bislang keinen zentralen, flächendeckenden Datensatz mit präzisen Geokoordinaten installierter Photovoltaikanlagen. Zwar liegen im Kartenviewer „STATatlas“ der Statistik Austria aggregierte Daten zur installierten Photovoltaikleistung auf Gemeindeebene vor, jedoch fehlt ein konkreter räumlicher Bezug auf Objektebene, etwa in Form von genauen Adressen oder spezifischen Geometrien der Anlagen. Diese Lücke in der Datenerfassung unterstreicht die Notwendigkeit präziserer Lokalisierungsmethoden, wie sie im Rahmen dieser Arbeit angewendet werden (Statistik Austria, 2024). In einem E-Mail-Kontakt mit einem Mitarbeiter des BEV wurde bestätigt, dass im Rahmen einer flächenhaften Durchsicht, die alle drei Jahre erfolgt, Photovoltaikanlagen manuell erfasst werden – jedoch nicht zu Forschungszwecken, sondern „aufgrund der Notwendigkeit, die in diesem Bereich besteht“. Dabei werden sichtbare Anlagen eingezeichnet, da auf kommunaler Ebene oftmals kein vollständiger Überblick über den Anlagenbestand besteht. Daraus ergibt sich ein klarer Bedarf an Methoden zur automatisierten Lokalisierung und Erfassung von Photovoltaikanlagen mit Raumbezug, der für Verwaltung, Planung und Forschung relevant ist (BEV, persönliche Mitteilung, 22. Dezember 2022).

1.2 Stand der Forschung

Die Erkennung und Klassifizierung von Photovoltaikanlagen war bereits in der Vergangenheit Gegenstand der Forschung, insbesondere bevor Convolutional Neural Networks (CNN) in den Vordergrund rückten. Methoden zur Objekterkennung und Bildsegmentierung lassen sich grundsätzlich in zwei Gruppen einteilen: Solche, die auf neuronalen Netzen basieren, und solche, die auf nicht-neuronalen Verfahren beruhen (Tang et al., 2021). Letztere werden auch als traditionelle bzw. flache (Machine Learning-)Methoden bezeichnet und gelten in der Regel als ausgereift und bewährt. Dazu zählen beispielsweise Support Vector Machines (SVM), Random Forest oder die Lineare Regression (Talluri, 2017; Zhu et al., 2017). Wichtige Vertreter der auf neuronalen Netzen basierenden Methoden sind Faster R-CNN, Mask R-CNN, YOLO (You Only Look Once) und RetinaNet. Immer mehr auf CNN basierende Modelle finden Einzug in die Informationstechnologie. Dabei tragen nicht zuletzt einflussreiche Unternehmen wie Microsoft, Facebook und Google sowie Universitäten und engagierte Privatpersonen zur Entwicklung neuer Ansätze bei (Sugomori et al., 2017).

Durch den technologischen Fortschritt und insbesondere durch die Entwicklung leistungstärkerer Hardware für High Performance Computing (HPC) wurde CNN-basierte Bildverarbeitung zunehmend praktikabler. Verbesserungen in der Rechenleistung, Speicherkapazität, Energieeffizienz, Bildsensorauflösung sowie leistungstärkere Grafikkarten haben die Performance und Kosteneffizienz von Computer Vision-Methoden und Deep Learning-Modellen erheblich optimiert. Dennoch erfordern insbesondere Deep Learning-Modelle eine große Menge an Trainingsdaten. Werden diese

bereitgestellt, zeichnen sich diese Modelle durch eine hohe Lern- und Verarbeitungsleistung sowie eine hohe Genauigkeit aus (Ardeshir et al., 2014).

Verschiedene Studien untersuchen sowohl objekt- als auch pixelbasierte Klassifizierungsansätze zur Identifizierung von PV-Anlagen (siehe Kapitel 5.8). Dabei kommen unterschiedliche Datenquellen mit variierenden geometrischen Auflösungen zum Einsatz. Beispielsweise klassifizierte Vasku (2019) PV-Anlagen anhand von Sentinel-1- und Sentinel-2-Satellitenbildern mit einer geometrischen Auflösung von 10m. Ge et al. (2022) entwickelten Deep Learning-Methoden zur Erkennung und Klassifizierung von Photovoltaikanlagen basierend auf Satellitenbildern mit einer Bodenauflösung von 16m. Petrik und Hamme (2020) hingegen arbeiteten mit Orthofotos, die frei von Verzerrungen sind und eine deutlich höhere Auflösung aufweisen. Perry und Campos (2023) konzentrierten sich zusätzlich auf die Analyse von Montage- und Installationstypen von Photovoltaikanlagen. Darüber hinaus gibt es bereits Forschung zur Identifizierung defekter oder verschmutzter PVA. Infrarotbilder, insbesondere Thermalbilder, können hierbei helfen. Mit diesem Thema befassten sich beispielsweise Díaz et al. (2020) in ihrer Publikation.

1.3 Ziel der Arbeit, Forschungsfrage und begleitende Arbeitsfragen

Ziel dieser Masterarbeit ist es, zu untersuchen, ob sich die Objekterkennung oder die Bildsegmentierung besser für die automatische Erkennung und Klassifikation von Photovoltaikanlagen eignet. Dabei liegt der Fokus auf modernen Deep-Learning-Ansätzen, die auf Convolutional Neural Networks (CNN) basieren. Klassische bzw. traditionelle Machine Learning-Methoden bleiben bewusst unberücksichtigt.

Die zentrale **Forschungsfrage** lautet:

- „Ist in der Computer Vision die Objekterkennung oder die Bildsegmentierung zur Erkennung und Klassifizierung von Photovoltaikanlagen die qualifiziertere Methode?“

Zur Beantwortung dieser Frage wird ein Vergleich zwischen zwei Teilgebieten der Computer Vision angestellt: Zwischen objekt- und pixelbasierten Klassifizierungsansätzen - der Objekterkennung und der Bildsegmentierung. Im Bereich der Bildsegmentierung werden sowohl die semantische als auch die Instanzsegmentierung berücksichtigt. In den Kapiteln 5.8 und 5.8.1 wird die bestehende Literatur sowie die daraus abgeleiteten Erkenntnisse näher erläutert. Die Modelle werden auf ihre Fähigkeit hin untersucht, Photovoltaikanlagen zuverlässig zu erkennen und von ähnlichen Objekten – insbesondere Solarthermieranlagen – zu unterscheiden.

Folgende **begleitende Arbeitsfragen** dienen zur Strukturierung der Analyse und letztendlich auch für die Beantwortung der Forschungsfrage:

- „Welche Deep Learning-Modelle können jeweils für die Objekterkennung sowie die Bildsegmentierung in Betracht gezogen werden?“
- „Welche Entwicklungsumgebung eignet sich für die praktische Umsetzung der Aufgabenstellung?“
- „Gibt es bestimmte Parameter oder Indikatoren für die einzelnen Modelle, die die Erkennbarkeit und Klassifizierbarkeit von Photovoltaikanlagen verbessern kann?“

Zur Beantwortung dieser Fragen werden vier Modelle herangezogen: Faster R-CNN (Objekterkennung) und Mask R-CNN (Instanzsegmentierung), jeweils trainiert auf einem Einzelklassendatensatz (nur Photovoltaikanlagen) sowie auf einem kombinierten Datensatz (Photovoltaik- und Solarthermieanlagen). Damit werden sowohl unterschiedliche Modellansätze als auch unterschiedliche Trainingsdaten berücksichtigt. Auffällig war während der Literaturrecherche, dass der Fokus bisher primär auf der Objekterkennung und der semantischen Segmentierung lag. Die Instanzsegmentierung – insbesondere mithilfe von Mask R-CNN – wurde bislang kaum im Kontext der automatisierten Erkennung von Photovoltaikanlagen thematisiert. Deshalb wird in dieser Arbeit gezielt auch die Instanzsegmentierung einbezogen, um zu untersuchen, mit welchen Deep-Learning-Ansätzen und letztlich mit welchen konkreten Modellen Photovoltaikanlagen am zuverlässigsten identifiziert werden können.

Das Untersuchungsgebiet liegt in Österreich – eine Region, in der bislang keine vergleichbare Forschung bekannt ist. Als Datengrundlage dienen hochauflösende Orthofotos mit einer Bodenauflösung von 0,2m.

1.4 Methodische Vorgehensweise

Zunächst wird mit einer fundierten Literaturrecherche in Fachartikeln, Büchern, Blogs und wissenschaftlichen Online-Plattformen wie *ResearchGate*, *MDPI* (Multidisciplinary Digital Publishing Institute) und *IEEE* (Institute of Electrical and Electronics Engineers) begonnen. Diese Recherche dient dazu, aktuelle Entwicklungen im Bereich Deep Learning und der automatisierten Objekterkennung und -segmentierung in der Fernerkundung nachzuvollziehen. Darauf aufbauend werden die wichtigsten theoretischen Grundlagen erarbeitet: Die Funktionsweise von Convolutional Neural Networks, die Struktur von Trainings- und Validierungsdatsätzen, die Bedeutung der Modellvalidierung sowie die Rolle von Hyperparametern wie Lernrate und Epochenanzahl.

Im praktischen Teil der Arbeit werden zwei verschiedene Deep Learning-Ansätze verfolgt:

- Objekterkennung mithilfe von Faster R-CNN
- Bildsegmentierung bzw. Instanzsegmentierung mithilfe von Mask R-CNN

Beide Modelltypen werden jeweils mit zwei verschiedenen Trainingsdatensätzen trainiert:

- Einem Einzelklassendatensatz (Datensatz 1), der ausschließlich Photovoltaikanlagen enthält.
- Einem Kombinationsdatensatz (Kombinierter Datensatz), der sowohl Photovoltaik- als auch Solarthermieanlagen umfasst.

Auf diese Weise ergeben sich insgesamt vier Modellvarianten, die hinsichtlich ihrer Erkennungsgenauigkeit und ihrer Fähigkeit zur Unterscheidung zwischen ähnlichen Objekten verglichen werden. Die Modelle werden mithilfe der Python-Bibliothek `arcgis.learn` in ArcGIS Pro trainiert. Die Trainingsdaten bestehen aus manuell annotierten Photovoltaik- und Solarthermieanlagen auf hochauflösenden Orthofotos (0,2m), die vom Bundesamt für Eich- und Vermessungswesen (BEV) bereitgestellt und in der GIS-Software ArcGIS Pro erstellt werden. Die Annotationen erfolgen in einem definierten Referenz- bzw. Untersuchungsgebiet und werden später auf ein Testgebiete übertragen. Ziel ist es, die Vorhersagen visuell und quantitativ auszuwerten. Dazu werden zwei Evaluationsmethoden eingesetzt:

- Eine klassifikatorische Bewertung auf Basis von Precision, Recall und F1-Score.
- Eine räumliche Bewertung auf Basis der Intersection over Union (IoU).

Die Integration von Solarthermieranlagen dient dabei nicht nur der Klassifikation, sondern auch als Kontrollmechanismus zur Vermeidung von Fehlklassifikationen aufgrund optischer Ähnlichkeiten. Weitere relevante Bewertungskriterien sind unter anderem Rechenaufwand, benötigte Hardware, Trainingszeit und Modellgenauigkeit.

1.5 Aufbau der Arbeit

Die vorliegende Arbeit ist in neun Kapitel gegliedert. Nach der Einleitung in Kapitel 1, die unter anderem die Motivation, Zielsetzung und methodische Vorgehensweise beschreibt, folgt in Kapitel 2 ein Überblick über die Energieversorgung in Österreich mit besonderem Fokus auf die Photovoltaik und ihrer Marktentwicklung in den letzten Jahren. Kapitel 3 und 4 widmen den theoretischen Grundlagen der künstlichen Intelligenz und künstlicher neuronaler Netze (KNN). Anschließend wird an Deep Learning und seiner bekanntesten Architektur, Convolutional Neural Networks, angeknüpft. In Kapitel 5 werden zentrale Konzepte des Deep Learning sowie der Computer Vision erläutert. Darauf aufbauend beschreibt Kapitel 6 die praktische Umsetzung, beginnend bei der Datengrundlage über die eingesetzten Modelle bis hin zum Training. Die Resultate dieser Trainings werden in Kapitel 7 vorgestellt und analysiert. Kapitel 8 dient der übergeordneten Diskussion der Ergebnisse im Kontext der Forschungsfrage sowie der methodischen Einbettung. Abschließend fasst Kapitel 9 die zentralen Erkenntnisse der Arbeit zusammen und gibt einen kurzen Ausblick.

2 Energieversorgung und -industrie in Österreich

Die Verwendung fossiler Brennstoffe in der Energieerzeugung hat u. a. zwei bedeutende Nachteile: Einerseits führt sie zu einer verstärkten Abhängigkeit von Staaten, die Erdöl exportieren, was insbesondere in weltpolitischen Krisenzeiten problematisch sein kann. Andererseits tragen die Emissionen von Schadstoffen zum Klimawandel bei (Groiß et al., 2013). Ein stetiger Bedarf an diesen Ressourcen hat zunehmend negative Folgen auf das Ökosystem und stellt parallel das Wohlbefinden der Erdbevölkerung infrage. Der Rohstoffabbau und die -verarbeitung schädigen das Klima, nicht zuletzt durch die Umwandlung in Gebrauchs- und Verbrauchsprodukte, die wiederum neue Rohstoffe (sogenannte Sekundärrohstoffe) und Energie erfordern. Auch den Menschen bzw. den Lokalbevölkerungen kann unmittelbar Schaden zugefügt werden, etwa wenn nichtorganische Rohstoffe abgebaut und verarbeitet werden, dabei Chemikalien mit Wasser in Berührung kommen und Landschaften sowie Trinkwasser verpestet werden (Scherhauser et al., 2020).

Natürliche Ressourcen bzw. Primärrohstoffe stehen uns nur noch begrenzt zur Verfügung, und die Auswirkungen auf die Umwelt machen deutlich, dass mehr Nachhaltigkeit notwendig ist. Daher sollte der Weg zu erneuerbaren Ressourcen eingeschlagen werden. Gleichzeitig streben viele Staaten eine Energieautarkie an, d. h. eine energiewirtschaftliche Unabhängigkeit von Drittstaaten (Scherhauser et al., 2020). Ein Übergang zu nachhaltigen Erzeugungsquellen ist somit entscheidend für die zukünftige Energieversorgung.

2.1 Ziele und Strategien für eine nachhaltige Energieversorgung

Um sich den Zielen des Pariser Klimaschutzübereinkommens für das Jahr 2030 anzunähern, ist ein zügiger Ausstieg aus fossilen Brennstoffen unerlässlich. In diesem Kontext plant Österreich, bis 2030 die gesamte Stromgewinnung ausschließlich aus erneuerbaren Energiequellen zu beziehen. Zur Umsetzung dieses Ziels ist eine Erhöhung der erneuerbaren Produktion von Strom bis 2030 um 27 Terawattstunden (TWh) geplant. Von dem Gesamtverbrauch sollen 11 TWh durch Photovoltaik, 10 TWh durch Windkraft, 5 TWh durch Wasserkraft und 1 TWh durch Biomasse bereitgestellt werden. Insbesondere die Photovoltaik soll stark ausgebaut werden. Bei diesem ambitionierten Ziel für Photovoltaik wird der Beitrag der Prosumenten von großer Bedeutung sein (Kettner et al., 2022). Prosumenten sind Verbraucher, die zugleich Konsumenten und Produzenten sind oder umgekehrt (DWDS, 2021). Die Beteiligung verschiedener Haushaltsgruppen am Strommarkt als Prosumenten wird je nach Ausstattung der Haushalte variieren, besonders in Bezug auf Gebäudetyp und Haushaltseinkommen. Diese Unterschiede werden Auswirkungen auf die Verteilung der Elektrizitätswende haben. Durch die Beteiligung als Prosumenten können Haushalte ihren Stromverbrauch mindern und dadurch Kosten sparen (Kettner et al., 2022).

Für die Energiewende hat Österreich eine Strategie präsentiert. Diese umfasst, mit dem Ziel, bis 2040 klimaneutral zu werden, nicht nur Maßnahmen zur Energieeinsparung, sondern auch die Erhöhung der Stromerzeugung aus regenerativen Energien. Photovoltaikanlagen (PVA) und Windkraftanlagen (WKA) haben beim Ausbau der österreichischen Stromproduktion das größte Potenzial (Scherhauser et al., 2020).

2.2 Marktentwicklung der Photovoltaik im Jahr 2023

Vor dem internationalen Hintergrund der nachhaltigen Entwicklung und der Verpflichtung der Staaten, eine gemeinsame Zukunft für die Menschheit zu gestalten, bietet die Solarenergie erhebliche Vorteile. Dazu zählen ihre Unerschöpflichkeit, der Wegfall von Transport- und -kosten (und damit verbundenen Emissionen) sowie ihre Umweltfreundlichkeit. Als wichtige erneuerbare Energiequelle zur Abmilderung der Energiekrise ist die Solarenergie zunehmend in den Fokus der Forschung gerückt. Insbesondere entwickelt sich die Photovoltaikindustrie und der Markt wird kontinuierlich gefördert (Guo, 2018).



Abb. 1: Monokristalline PV-Zellen
(Homelab233, 2021)



Abb. 2: Polykristalline PV-Zellen
(Riemer, 2009)

Durch das Einfangen von Sonnenlicht wird mithilfe der Photovoltaik elektrische Energie gewonnen. Die speziellen Solarzellen (auch Solarpanels genannt) wandeln die Lichtenergie in elektrische Energie um, die zum Betrieb von Geräten oder zur Einspeisung ins Stromnetz verwendet werden kann. Der Photovoltaikmarkt hat in der Vergangenheit einige Höhen und Tiefen durchlaufen. Nach einem Rekordzuwachs im Jahr 2013 stabilisierte sich das Marktwachstum zwischen 2014 und 2018 mit jährlichen Zuwachsraten zwischen 150 MWpeak und 190 MWpeak. Dieser Anstieg wurde durch eine außergewöhnliche Fördermaßnahme ausgelöst (Biermayr et al., 2023). Der Zusatz „Peak“ bedeutet „Spitze“ und drückt in der Photovoltaik die maximale Leistung von Solarzellen aus. Er wird an die üblichen Energieeinheiten angehängt, etwa Watt Peak (Wpeak), Kilowatt Peak (kWpeak), Megawatt Peak (MWpeak) und Gigawatt Peak (GWpeak) (Kesselheld, 2018).

In den darauffolgenden Jahren nahm die Leistung der Neuinstallationen kontinuierlich zu. Im Jahr 2021 wurde mit 739,7 MWpeak ein bemerkenswerter Zuwachs erzielt, der 2022 ein weiteres Mal übertroffen wurde. Erstmals überschritt die neu installierte Photovoltaikleistung 2022 die Grenze von 1 GWpeak und erreichte 1009,1 MWpeak – ein Anstieg um 36,4% im Vergleich zu 2021. Ende 2022 waren insgesamt 3791,7 MWpeak installiert (kumulierten Stromgesamtleistung), was einem Zuwachs von 36,3% entspricht (siehe Abbildung 3) (Biermayr et al., 2023). Der Anteil der Photovoltaik an der österreichischen Stromerzeugung ist in den letzten Jahren deutlich gestiegen und beträgt derzeit rund 6% (BMK, 2023).

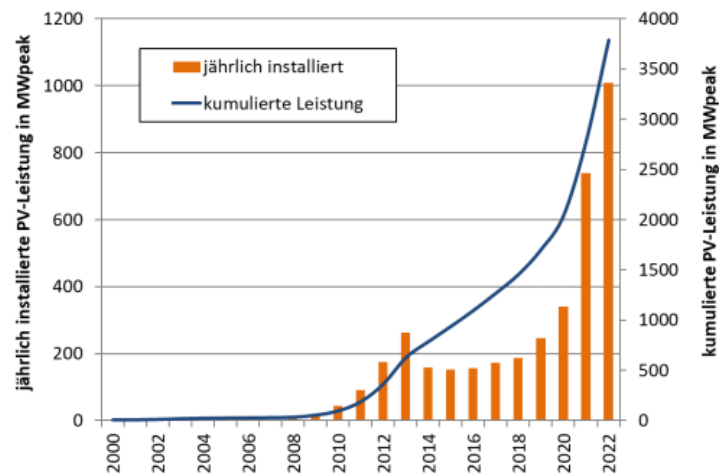


Abb. 3: PV-Leistung in den Jahren 2000 – 2022
bis 2006 (Faninger, 2007), ab 2007 (Technikum Wien, 2023)

2.2.1 Anlagen-, Montageart und Solarzellentypen

83,7% der neu installierten Photovoltaikleistung entfällt mehrheitlich auf Aufdachanlagen, während freistehende Anlagen 14,9% ausmachen. Fassaden- und dachintegrierte Photovoltaikanlagen spielen eine untergeordnete Rolle. Im Jahr 2022 wurden fast ausschließlich monokristalline Solarzellen eingesetzt (Biermayr et al., 2023).

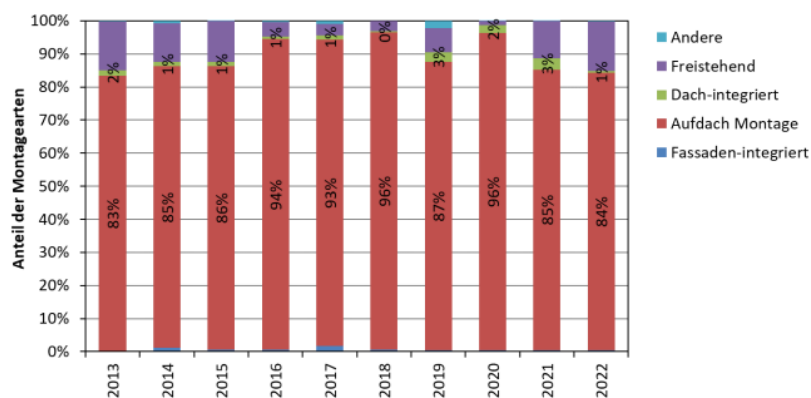


Abb. 4: Montageart der installierten PV-Anlagen der Jahre 2013 – 2022
(Technikum Wien, 2023)

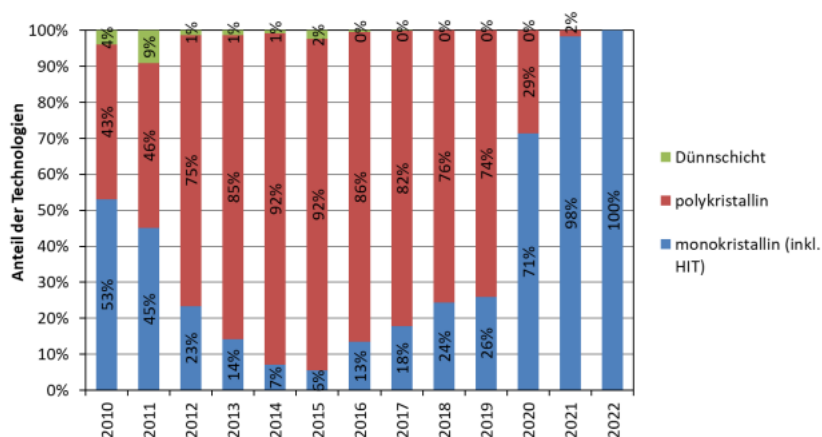


Abb. 5: Solarzellentypen der installierten PV-Anlagen in den Jahren 2010 – 2022
(Technikum Wien, 2023)

2.3 Zukünftige Aussichten

Die neu installierte Photovoltaikleistung birgt großes Potenzial und spielt in der österreichischen Energiewende eine entscheidende Rolle. Die Bundesregierung hat sich das Ziel gesetzt, bis 2030 den gesamten Strom aus erneuerbaren Energien zu erzeugen und eine installierte PV-Leistung von knapp 11 GW_{peak} zu erreichen. Das entspricht einer zusätzlichen Energieerzeugung von 11 TWh aus Photovoltaik. Auch liegt Österreich mit einer jährlichen PV-Stromproduktion von ca. 6,6% noch unter dem EU-Durchschnitt von 8,7%, was weiteres Entwicklungspotenzial bietet (Biermayr et al., 2023; BMK, 2023).

Die Klima- und Energieziele des Pariser Klimaschutzübereinkommens sind mit der bisherigen Strategie bis 2030 voraussichtlich nicht erreichbar. Auch spielt die Photovoltaik für die Umsetzung des heimischen Erneuerbaren-Ausbau-Gesetzes (EAG) eine zentrale Rolle, da bis 2030 die gesamte Stromversorgung durch regenerative Energiequellen gedeckt werden soll. Dennoch ist der bisherige Photovoltaikausbau nicht ausreichend. Daher sollte der Fokus verstärkt auf die EU-Klimaneutralitätsziele für 2040 gelegt werden. Der wissenschaftliche Beirat der Europäischen Union (EU) für den Klimawandel fordert eine Reduktion der Treibhausemissionsgase von 90% bis 95% bis 2040 (Biermayr et al., 2023).

3 Künstliche Intelligenz: Ein Überblick

Künstliche Intelligenz (KI; engl.: Artificial Intelligence, AI) ist ein Teilbereich der Informatik (engl.: Computer Science), der darauf abzielt, intelligente menschliche Aktivitäten durch die Simulation mithilfe von Computern und Programmen zu realisieren (Poonkuntran et al., 2022). Künstliche Intelligenz bezeichnet Computersysteme, die Aufgaben übernehmen können, die normalerweise Menschen ausführen. Bereits heute erzielt sie Erfolge in Bereichen wie visueller Wahrnehmung, Spracherkennung und Sprachübersetzung sowie Entscheidungsfindung (Goswami, 2020). Weitere Anwendungsfelder sind die Cybersicherheit – etwa zur Erkennung von Bedrohungen im Netzwerkverkehr – sowie das Gesundheitswesen, wo sie Mustern in der Genexpression analysiert, um Krankheiten frühzeitig zu erkennen (Kalita, 2022). In den letzten Jahren ist KI sowohl ein bedeutsames als auch ein kontroverses Thema geworden, nicht zuletzt durch den Einfluss vom Chatbot ChatGPT.

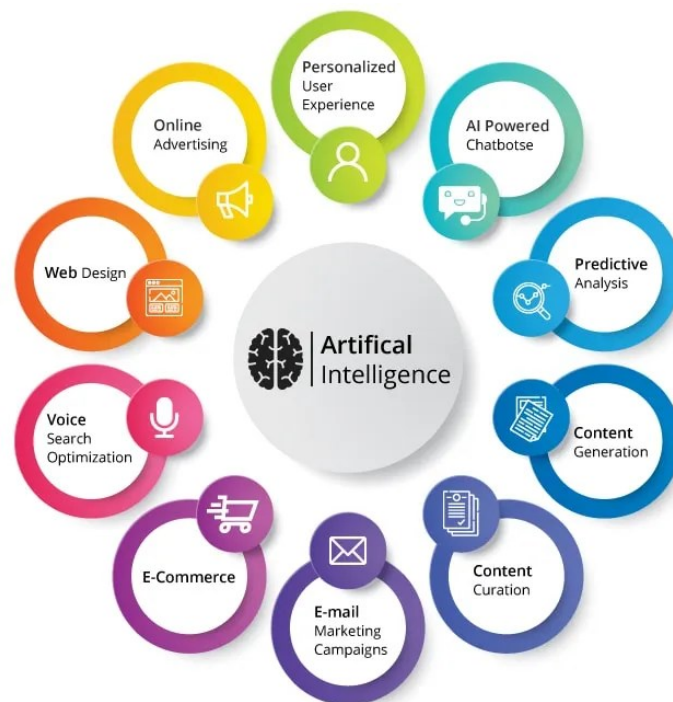


Abb. 6: Künstliche Intelligenz und ihre Anwendungsbereiche
(SuryaCreatX, 2023)

Der aktuelle Aufschwung ist bereits der dritte Zyklus intensiver Forschung. Seit den 1940er-Jahren durchläuft die KI-Forschung Phasen der Hoch- und Tiefpunkte, bekannt als „AI spring“ und „AI winter“ (Grunau, 2022). Der jüngste Aufschwung, oft als „Deep Learning-Boom“ bezeichnet, unterscheidet sich grundlegend von früheren Entwicklungen. Deep Learning hat eine entscheidende Neuerung hervorgebracht: Die Fähigkeit von Computern, Merkmale (engl.: Features) eigenständig aus Daten zu extrahieren und daraus zu lernen – ein Prozess, der zuvor manuelle Eingriffe erforderte („Feature extraction“) (Poonkuntran et al., 2022). Diese Errungenschaft ermöglicht es Maschinen, eigenständig neue Konzepte und Lösungsansätze zu entwickeln, wodurch sie gewissermaßen "Intelligenz erwerben" (Sugomori et al., 2017).

Bevor jedoch auf „Machine Learning“, „Artificial Neural Networks“ und das Paradigma des „Deep Learning“ eingegangen wird, folgt zunächst ein Überblick über die Funktionsweise des maschinellen Lernens sowie die ersten Errungenschaften der KI.

3.1 Geschichtliche Entwicklung der Künstlichen Intelligenz

Der Begriff „Künstliche Intelligenz“ (KI) wurde erstmals 1956 auf einer Konferenz am Dartmouth College in den USA von John McCarty geprägt. Dort wurden verschiedene Forschungsansätze vorgestellt, darunter auch neuronale Netzwerke (Kelleher, 2019). Im Laufe der Zeit entwickelten sich sowohl die Hardware als auch neue Algorithmen weiter, sodass zunehmend Expertenwissen mithilfe formaler Regeln und Formeln zur Lösung komplexer Aufgaben genutzt wurde. Später kamen datengetriebene Modelle hinzu, die aus Beobachtungen lernen. Ein eng mit KI verknüpftes Forschungsfeld ist die „Data Science“, deren Ziel es ist, Wissen und Erkenntnisse aus Daten zu gewinnen. Data Science umfasst dabei Disziplinen wie Machine Learning und Statistik (Grunau, 2022).

3.1.1 Die 1940er-Jahre

Viele forschungsbegeisterte Menschen haben sich Gedanken über die Möglichkeit mechanischer Intelligenz gemacht. Die Idee, Hardware- und Steuerprogramme zu kombinieren, um denkende, künstlich-intelligente Maschinen zu schaffen, die mit dem menschlichen Verstand konkurrieren können, war und ist äußerst attraktiv. Experimentelle Forschung auf diesem Gebiet konnte jedoch erst beginnen, als in den 1950er-Jahren elektronische Digitalcomputer breiter verfügbar wurden. Ein früheres Ziel der KI-Forschung war die Entwicklung von Maschinen mit allgemeiner Intelligenz. Turing argumentierte bereits 1936, dass es oft einfacher sei, einer Maschine spezifische Aufgaben beizubringen, anstatt ihr das gesamte Intelligenzniveau eines Menschen zu vermitteln. Erste Erfolge wurden erzielt, als der Fokus auf intelligentes Verhalten in eng begrenzten Anwendungsbereichen gelegt wurde (Openshaw & Openshaw, 1997).

Ein wichtiger Meilenstein war das 1943 von McCulloch und Pits entwickelte mathematische Modell eines Neurons, das als Ausgangspunkt für viele spätere Forschungsarbeiten diente. Dieses Modell markiert den Beginn der Geschichte des Machine Learning (Openshaw & Openshaw, 1997; Vasuki & Govindaraju, 2017). Ihr Ziel war es, die Funktionsweise biologischer Neuronen nachzubilden. Ihre Arbeit hatte großen Einfluss, und seither sind **Künstliche Neuronale Netze** (KNN; engl.: Artificial Neural Networks, ANN) – auch als Neuroinformatik (engl.: Neurocomputing) bekannt – ein zentrales Forschungsfeld. Eine weiterer bedeutender Fortschritt gelang 1949, als Donald Olding Hebb auf der Grundlage von Tierversuchen Lernmechanismen postulierte, die zum Trainieren neuronaler Netze genutzt werden (Openshaw & Openshaw, 1997). In künstlichen neuronalen Netzen passen Lernalgorithmen die Verbindungsgewichte zwischen Neuronen an. Dieses Prinzip wurde als *Hebb'sche Lernregel* bekannt: Das Gewicht zwischen zwei Neuronen wird verstärkt, wenn beide gleichzeitig aktiv sind. Auf diese Weise lernt das Netzwerk die Wechselwirkung zwischen den Neuronen (Alpaydin, 2021). Hebb's Erkenntnisse inspirierten viele weitere Forscher und legten den Grundstein für die Neuroinformatik, die ab den 1980er-Jahren erneut an Bedeutung für die KI-Forschung gewann (Openshaw & Openshaw, 1997).

3.1.2 Die Jahre 1950 bis 1970

Nicht nur das menschliche Gehirn diente als Inspirationsquelle für KI-Forschung, sondern auch andere biologische Systeme – insbesondere die Mechanismen, durch die sich Spezies entwickeln und an ihre Umwelt anpassen (Openshaw & Openshaw, 1997).

Ein entscheidender Fortschritt war die Entwicklung des „Perceptron“ (dt.: Perzeptron), des ersten erfolgreichen Neurocomputers, der in der Lage war, einfache Muster zu erkennen. Dieses künstliche neuronale Netz 1957 vom Informatiker und Psychologen Frank Rosenblatt entwickelt. Seine Erfindung löste große Begeisterung aus. Das Perceptron war ein selbstorganisierender Automat, der auf einem vereinfachten Modell der Netzhaut des menschlichen Auges basierte (Openshaw & Openshaw, 1997; Wlodarczak, 2019). Es bestand aus einem einzigen Neuron mit einem binären Zustand (passiv = 0, aktiv = 1), der aus der gewichteten Summe der Eingangsvariablen und einem Schwellenwert berechnet wurde (Grunau, 2022). Rosenblatt gilt als Begründer der modernen Neuroinformatik und neuronaler Netze (Openshaw & Openshaw, 1997).



Abb. 7: Perzeptron
(Lefkowitz, 2019)

Damals nahm man an, dass ein solches Netzwerk mit künstlichen Neuronen nach ausreichendem Training nahezu alles leisten könnte. Dabei wurde jedoch vernachlässigt, dass das menschliche Gehirn über etwa 100 Milliarden bis 100 Billionen Nervenzellen verfügt (Paál, 2024; Wissen.de, o. J.). Zum Vergleich: 1997 ging man noch von etwa 10 Milliarden Neuronen aus. Selbst mit damaliger Technologie war es nicht annähernd möglich, diese Komplexität nachzubilden. In den 1950er-Jahren waren sowohl die analogen Röhrenschtaltungen als auch die digitalen Computer zu groß, zu langsam und speicherkapazitätsbedingt unzureichend für realistische KI-Experimente. Zudem waren die verfügbaren Softwaretools noch sehr rudimentär (Openshaw & Openshaw, 1997).

Ein entscheidender Wendepunkt kam 1969 mit dem Werk von Minsky und Papert, welches das Ende der ersten Ära neuronaler Netze markierte. Ihre theoretische Analyse zeigte, dass einfache, einschichtige neuronale Netze bestimmte Aufgaben nicht lösen konnten – insbesondere das exklusive ODER-Problem (XOR; auch als „ausschließende Disjunktion“ bekannt) (Openshaw & Openshaw, 1997). Während ein Perceptron in der Lage war, logische Operationen wie das inklusive ODER (auch als „nicht-ausschließende Disjunktion“ bekannt) darzustellen, scheiterte es am exklusiven ODER. Dies führte zu einer langen Phase der Stagnation in der Forschung zu neuronalen Netzen (Alpaydin, 2021).

3.1.3 Die 1960er- bis 1980er-Jahre

Als Folge der vorherigen Entwicklungen kam es zu einer Rückkehr zum grundlegenden Problem der Modellierung menschlichen Denkens und der Repräsentation von Wissen. Expertensysteme (engl.: Expert Systems) verfolgten einen völlig anderen Ansatz, um die Prozesse menschlicher Intelligenz zu

modellieren oder nachzubilden. Sie basierten auf der Annahme, dass menschliche Experten eine Vielzahl von Faustregeln anwenden, die auf Wissen und Erfahrungen beruhen und für bestimmte Aufgaben relevant sind. Expertensysteme wurden entwickelt, um dieses Modell der Intelligenz operational zu machen. Man erkannte bald, dass Computer intelligentes Verhalten zeigen können, wenn der Bereich, in dem sie tätig sind, ausreichend eingegrenzt wird. In den frühen 1980er-Jahren wurden die „Intelligent Knowledge-Based Systems“ (IKBS) populär. Diese Systeme kodierten Wissen in Form von Regeln, auf die einfache Argumentationsverfahren angewendet wurden. Diese Regeln repräsentieren das Wissen von Experten (Openshaw & Openshaw, 1997).

Expertensysteme wurden häufig als Technologie betrachtet, die den Menschen in bestimmten Bereichen ersetzen könnte, und als potenzielle Lösung für den Fachkräftemangel. Allerdings war ihre grundlegende Schwäche, dass sie nur das Wissen menschlicher Experten nachbildeten und nicht in der Lage waren, über dieses hinaus selbstständig zu lernen oder zu innovieren. In vieler Hinsicht wünschte man sich jedoch superintelligente Systeme, die mehr leisten könnten als nur das Nachahmen menschlicher Intelligenz (ebd.).

Eine wettbewerbsfähige industrielle und wirtschaftliche Bedrohung, insbesondere aus Japan, war ein wichtiger Faktor, der einen Ansturm auf neue Forschungsmittel für diese Art von KI auslöste und die Forschung weiter vorantrieb. In den späten 1980er- und 1990er-Jahren wurde die Definition von IKBS erweitert, sodass sie nun auch Neuroinformatik und alle anderen Formen des Machine Learning umfasste. Dadurch wurden Expertensysteme in ihrer ursprünglichen Form reformiert, jedoch in einer völlig neuen Richtung (Openshaw & Openshaw, 1997).

3.1.4 Mitte der 1980er-Jahre

Die Zweifel an IKBS, die auf Expertensystemen basieren, fielen mit dem Aufschwung des Machine Learning im Allgemeinen und insbesondere der neuronalen Netze zusammen, die jedoch zunächst in einer langsamen Stagnation steckten (Alpaydin, 2021). Die Wissensbasis von Expertensystemen ist statisch, und die Systeme besitzen keine automatische Lernfähigkeit. Neuronale Netze hingegen bieten ein praktikables Paradigma für Machine Learning, das die Möglichkeit bietet aus großen Mengen von Daten zu lernen. Sobald die Implementierungsprobleme überwunden waren, eröffnete diese Technologie ein breites Spektrum an möglichen Anwendungen (Openshaw & Openshaw, 1997).

Die früheren Probleme rund um das Perceptron wurden Mitte der 1980er-Jahre überwunden. Besonders die Arbeit von Rumelhart und McClelland zur parallel-verteilten Verarbeitung im Jahr 1986 trug wesentlich zum Fortschritt der neuronalen Netze bei. Der von ihnen entwickelte **Backpropagation-Algorithmus** ermöglichte ein praktisches Mittel, um mehrschichtige neuronale Netze zu trainieren (Alpaydin, 2021; Openshaw & Openshaw, 1997). Dieser Algorithmus ermöglichte es, dass neuronale Netze statistische Muster aus großen Mengen von Beispieldaten erlernten und Vorhersagen für neue Instanzen treffen konnten. Dadurch konnten neuronale Netze auch komplexe Lernaufgaben bewältigen. Im Vergleich zu regelbasierten Systemen haben sich lernbasierte Systeme, wie neuronale Netze, in realen Anwendungen als wettbewerbsfähiger erwiesen. Gleichzeitig wurden seit Mitte der 1980er-Jahre auch zwei unterschiedliche Ansätze im maschinellen Lernen beobachtet: „Shallow Learning“ und „Deep Learning“ (Jiang et al., 2019). Ein weiteres entscheidendes Ereignis war das neue Interesse an diesem Thema in vielen Ländern, das auf die Entwicklung leistungsfähiger und preiswerter PCs (Personalcomputer) zurückzuführen war. Darüber hinaus gab es viele praktische und kommerzielle

Anwendungen, die den Wert neuronaler Netze unter Beweis stellten. Diese technologischen Fortschritte führten zu der Erkenntnis, dass neuronale Netzwerke aufgrund ihrer praktischen Anwendungen und ihrer vielversprechenden Fortschritte eine zentrale Rolle in der KI-Forschung spielten (Openshaw & Openshaw, 1997).

Künstliche neuronale Netze sind auch deshalb von großer Bedeutung, weil sie den Anwendungsbereich der Informatik auf Disziplinen ausgeweitet haben, die zuvor als außerhalb des klassischen Bereichs der Computermodellierung und -analyse galten (Openshaw & Openshaw, 1997). Dazu gehören u. a. Physik, Statistik, Psychologie, Kognitionswissenschaften, Neurowissenschaften und Sprachwissenschaften sowie Informatik, Elektrotechnik und adaptive Steuerung (Alpaydin, 2021). Die Fähigkeit neuronaler Netze, verrauschte Daten sowie nichtlineare Beziehungen zu verarbeiten, macht sie auch für Geographen besonders relevant. Der wichtigste Beitrag zur Forschung an neuronalen Netzen ist die Interdisziplinarität und das Zusammenspiel dieser verschiedenen Disziplinen (Openshaw & Openshaw, 1997).

3.1.5 Die 1990er-Jahre

Bereits in den frühen 1990er-Jahren wurden zwei erfolgreiche Anwendungen neuronaler Netze vorgestellt: LeCuns „LeNet“-Netz zur Erkennung handgeschriebener Ziffern und Tesauros „TD-Gammon“-Netz, das Backgammon spielte (Alpaydin, 2021; Sewak et al., 2018).

In den Anfangsjahren der 1990er-Jahre gab es jedoch nur wenige bedeutende Fortschritte im Bereich neuronaler Netze. Das Hauptaugenmerk lag auf unüberwachten Lernverfahren. Dabei wurde eine Technik verwendet, die versuchte, die Anzahl der Merkmale oder Dimensionen in einem Datensatz zu reduzieren, während die wesentliche Struktur und die Beziehungen der Daten erhalten blieben (Sufyan, 2023). Ein weiteres Problem bei neuronalen Netzen mit vielen Schichten war das sogenannte „verschwindende Gradienten“-Problem, das ihre Funktionsfähigkeit einschränkte. Daher wurden ab Mitte der 1990er-Jahre eine Reihe von Lernansätzen auf der Grundlage von Shallow Learning-Modellen populär, darunter Support Vector Machines (SVM), logistische Regression (LR) und Random Forests (Zhu et al., 2017). Diese Modelle werden auch als flache Modelle bezeichnet, da sie entweder keine oder nur eine Zwischenschicht besitzen. In Theorie und Praxis erwiesen sich diese Ansätze als äußerst erfolgreich. Sie sind einfach zu trainieren und eignen sich besonders gut für Aufgaben mit kleinen Stichproben. Im Gegensatz dazu sind neuronale Netze aufgrund ihrer Komplexität schwieriger zu trainieren. Auch wenn neuronale Netze bei einigen Aufgaben weiterhin gute Ergebnisse lieferten, galten sie zur damaligen Zeit als zu rechenintensiv und in ihrem Potenzial für viele praktische Anwendungen begrenzt (Jiang et al., 2019).

Ab dem Jahr 2000, mit der Weiterentwicklung der Internettechnologien, dominierten flache Modelle, insbesondere Support Vector Machines, in vielen praktischen Anwendungen wie Werbeempfehlungen, Informationssuche und Spam-Filtersystemen. Der Begriff „Deep Learning“ wurde erstmals von Aizenberg et al. im Jahr 2000 eingeführt (Jiang et al., 2019).

3.1.6 2006 bis zur Gegenwart

Der Durchbruch für Deep Learning (DL; dt.: Tiefes Lernen) kam erstmals 2006, als Geoffrey Hinton et al. von der Universität Toronto eine Methode namens Deep Belief Networks (DBN) vorstellten, die eine Erweiterung der neuronalen Netze darstellt (Bhardwaj et al., 2018; Sugomori et al., 2017).

2012 revolutionierte das von Krizhevsky, Sutskever und Hinton entwickelte Deep-Learning-Verfahren „SuperVision“ (auch bekannt als „AlexNet“) die Welt der Informatik und KI. Bei der „Imagenet Large Scale Visual Recognition Challenge“ (ILSVRC), einem Wettbewerb zur automatischen Bilderkennung aus dem Bereich der Computer Vision, setzte sich SuperVision durch, indem es die Konkurrenz bei der Erkennung von Tieren, Fahrzeugen und alltäglichen Objekten deutlich übertraf. Der Wettbewerb verwendete etwa 1,2 Millionen Trainingsbilder und 150 000 Testbilder. Verschiedene Methoden traten gegeneinander an, um die niedrigste Fehlerrate und die höchste Präzision zu erreichen. Bisher basierte die Bilderkennung auf menschlich definierten Merkmalen im Machine Learning. SuperVision brach mit diesem Ansatz und erreichte eine Präzisionsverbesserung von 10,7 Prozentpunkten. AlexNet reduzierte die Fehlerrate der besten konkurrierenden Modelle von etwa 26% auf 15,3% und setzte damit einen neuen Maßstab in der Bildklassifikation (Bhardwaj et al., 2018; Sewak et al., 2018; Wang et al., 2019).



Abb. 8: ImageNet
(Russakovsky et al., 2015)

Nach diesem Erfolg verlagerte sich der Schwerpunkt von Deep Learning zunächst auf neue unüberwachte Lerntechniken. Doch mit der Zeit wandte sich der Fokus zu überwachten Lerntechniken, die von großen Datensätzen profitieren konnten (Jiang et al., 2019).

In den darauffolgenden zehn Jahren hat Deep Learning dank fortschrittlicher mathematischer Formeln, leistungsfähigerer Computerhardware und großer Datensätze viel Aufmerksamkeit auf sich gezogen und sich zu einer dominierenden Technologie entwickelt, die in einer Vielzahl von Bereichen Anwendung findet (Bhardwaj et al., 2018; Jiang et al., 2019). Insbesondere das exponentielle Wachstum von Big Data motivierte die Entwicklung innovativer Methoden zur Extraktion semantischer Informationen aus Rohdaten, wie etwa Videos, Bildern und Sprachsequenzen. Es wird erwartet, dass Deep Learning in Zukunft auch viele weitere Bereiche transformieren wird. Unter den Methoden hat sich das Convolutional Neural Network (CNN; dt.: Faltungsneuronale Netze) als Standard etabliert, da es in vielen Anwendungen im Bereich des Computer Vision (CV; dt.: Maschinelles Sehen) eine nahezu

menschliche Genauigkeit erreicht. Dazu gehören Anwendungen wie Klassifikation, Erkennung, Segmentierung und Spracherkennung (Hassaballah & Awad, 2020).

3.1.7 Künstliche Intelligenz in der Geographie

Machine Learning und Deep Learning haben in der Geographie bedeutende Fortschritte ermöglicht. Sie werden erfolgreich zur Analyse großer Mengen von Geodaten eingesetzt, insbesondere bei der Klassifikation der Landbedeckung, der Erkennung von Veränderungen auf der Erdoberfläche (auch bekannt als Change Detection) und der Segmentierung von Landschaftsmerkmalen (Lavallin & Downs, 2021). Bereits Openshaw und Openshaw (1997) betonten, dass künstliche Intelligenz das Potenzial hat, komplexe räumliche Muster und Beziehungen zu analysieren, die mit traditionellen Methoden schwer zu erkennen sind. Sie wiesen darauf hin, dass der Einsatz von KI nicht nur die Geschwindigkeit und Effizienz geographischer Analysen erhöht, sondern auch eine neue Perspektive bei der Modellierung und Vorhersage räumlicher Prozesse bietet. Diese frühe Arbeit legte den Grundstein für den heutigen Einsatz von Machine Learning- und Deep Learning-Techniken, die eine präzise Verarbeitung von Geodaten ermöglichen (Openshaw & Openshaw, 1997).

In der Fernerkundung verbessern Deep Learning-Modelle wie Convolutional Neural Networks (CNN) die Genauigkeit bei der Objekterkennung, etwa für die Identifikation von Gebäuden, Straßen oder landwirtschaftlichen Flächen in Satellitenbildern. Diese Technologien können auch verwendet werden, um z. B. OpenStreetMap (OSM) mit automatisch extrahierten geographischen Merkmalen zu aktualisieren und zu erweitern. Mithilfe von Deep Learning können fehlende oder ungenaue Daten in OSM identifiziert und ergänzt werden, was zu einer genaueren Kartierung führt. Zudem tragen ML-Algorithmen zur Klassifizierung und Segmentierung von Bildern dazu bei, die georeferenzierten Daten in OSM automatisch zu verbessern und die Qualität der geographischen Daten insgesamt zu erhöhen (Miloudi & Meguenni, 2023). ML- und DL-Techniken spielen auch eine Schlüsselrolle in der Vorhersage und Modellierung von Umweltprozessen, wie die schnelle Identifizierung von Umweltveränderungen, die Überwachung der Luftqualität und die Analyse von Ozeantemperaturen für die Klimaforschung. Darüber hinaus wird die Automatisierung von Prozessen, wie der Erstellung von Karten oder der Extraktion von topographischen Merkmalen, durch KI erheblich beschleunigt (Copernicus, 2024).

Dank ihrer Fähigkeit, komplexe Muster in großen und heterogenen Datenmengen zu erkennen, treiben solche Methoden Innovationen im Bereich der Kartographie und Geoinformation entscheidend voran.

3.2 Grundlagen von Machine Learning

Beim maschinellen Lernen (engl.: Machine Learning, ML) wird versucht, das menschliche Lernen auf einem Computersystem zu imitieren. Während menschliches Lernen meist auf Erfahrung basiert, stützt sich ML auf Daten. Diese dienen als Ausgangspunkt des Lernprozesses und werden zum Trainieren und Testen eines Modells verwendet (Wlodarczak, 2019). Anwendungen wie die Objekt- und Bilderkennung, Spracherkennung sowie Bilder- und Sprachsuche sind Beispiele für Bereiche, in denen Machine Learning über Jahre hinweg erfolgreich eingesetzt wurde. Besonders die Objekterkennung ist auch im geographischen Kontext relevant.

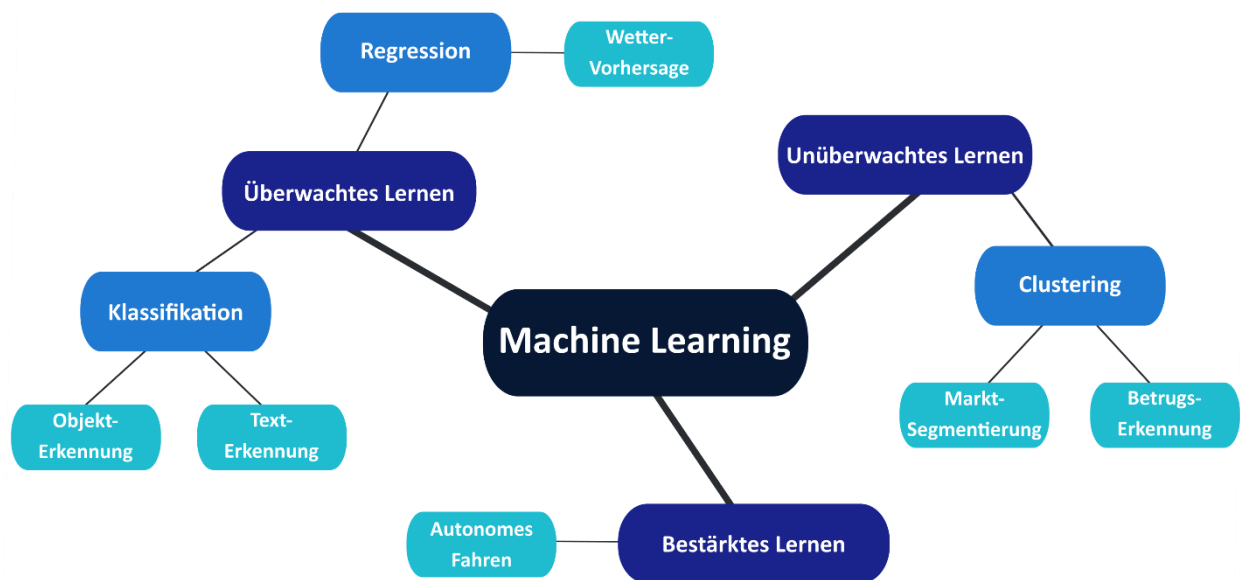


Abb. 9: Machine Learning und seine Zweige
 (Eigene Darstellung, erstellt mit Inkscape 1.2)

Machine Learning ist ein eigenständiges Studienggebiet, das eng mit Data Mining und der Datenanalyse verknüpft ist (Kalita, 2022). Seine Ursprünge reichen zurück bis 1943, als das erste mathematische Modell neuronaler Netze entwickelt wurde (Openshaw & Openshaw, 1997). Vor der Etablierung von ML waren Maschinen darauf beschränkt, Antworten aus einem zuvor gespeicherten Wissensrepertoire abzurufen. Der Schwerpunkt lag auf der Geschwindigkeit, mit der eine Maschine bekannte Fragen beantworten konnte. Mit der Entwicklung von ML sind Maschinen nun in der Lage, unbekannte Fragen basierend auf ihrem erlernten Wissen zu bewältigen. Fragen werden mit hoher Geschwindigkeit verarbeitet und es wird eine große Menge an Trainingsdaten genutzt. Machine Learning ist primär eine Methode der **Mustererkennung**. Ziel ist es, bestimmte Strukturen oder Muster in Daten zu identifizieren, zu klassifizieren und basierend darauf Vorhersagen zu treffen. Dies geschieht auf Grundlage eines probabilistischen statistischen Modells, das auf unterschiedlichen mathematischen Ansätzen basiert (Sugomori et al., 2017).

Während es für Menschen nahezu unmöglich ist, mehrdimensionale Daten oder Daten mit komplexen Mustern zu verarbeiten, kann ML diese Aufgaben effizient übernehmen. Vor der Weiterentwicklung von ML-Ansätzen waren Maschinen oft überfordert, doch dank maschineller Hilfe können sie jetzt auch Daten analysieren, die für Menschen schwer zugänglich sind. Wie im vorherigen Kapitel dargestellt, hat das Konzept von Machine Learning eine lange Geschichte. Allerdings war die Forschung lange Zeit durch den Mangel an verfügbaren Daten eingeschränkt. Durch die zunehmende Verbreitung des Internets und die fortschreitende Entwicklung von Open-Source-Datenbanken haben Forscher heute Zugriff auf eine Vielzahl von Datensätzen, die Experimente und die Weiterentwicklung von Algorithmen erleichtern. ML benötigt große Mengen an Daten, um Muster effektiv zu erkennen. Je mehr Daten und Muster verarbeitet werden, desto höher sind die Anforderungen an Datenmengen und Rechenkapazitäten. Frühere Technologien waren nicht in der Lage, solch hohe Anforderungen zu bewältigen (Sugomori et al., 2017).

Eine Maschine kann jedoch nicht selbst entscheiden, was sie lernen soll, wenn sie mit Daten gefüttert wird. Stattdessen benötigt sie sogenannte **Merkmale** (engl.: **Features**), die bestimmte Eigenschaften

eines Datenpunktes ausdrücken und als Eingabe für den Lernprozess dienen (Sugomori et al., 2017). Für ein beliebiges Objekt kann ein Merkmal beispielsweise dessen Name sein. Diese Merkmale werden in qualitative und quantitative Eigenschaften unterteilt. Beim Machine Learning wird eine Aufgabe durch die bereitgestellten Eingaben, wie z. B. Bilder, und die erwarteten Ergebnisse, wie Vorhersagen für bestimmte Klassen, definiert (Planche & Andres, 2019).

3.2.1 Lernmethoden im Machine Learning

Um ein Modell zu trainieren, gibt es drei grundlegende Ansätze: Überwachte, unüberwachte und teilüberwachte Lernmethoden. Die Wahl der Methode hängt von der Verfügbarkeit der Zielvariablen (engl.: Target Variable) sowie vom allgemeinen Ziel des Lernprozesses ab (Grunau, 2022).

Beim **überwachten Lernen** (engl.: Supervised Learning) müssen annotierte bzw. gelabelte (engl.: Label) Informationen in Form von Trainingsdaten vorliegen. Annotierte Daten sind solche, die mit korrekt zugewiesenen Tags versehen sind, wobei die gewünschten Ergebnisse bereits bekannt sind. Diese Methode ermöglicht es, gezielt spezifische Ergebnisse zu erzielen (Wlodarczak, 2019). Annotationen können aus einer endlichen Menge von Klassen bestehen, denen Beispiele zugeordnet werden, oder sie können komplexere Strukturen wie reelle Zahlen, Vektoren, Matrizen oder Graphen repräsentieren (Burkov, 2019). Annotierte Daten werden sowohl für das Training des Modells als auch die Bewertung seiner Genauigkeit verwendet (Wlodarczak, 2019).

Bekannte Algorithmen des überwachten Lernens sind (Wlodarczak, 2019):

- Naive Bayes
- Entscheidungsbäume (engl.: Decision Trees)
- Neuronale Netze
- Lineare Regression
- Logistische Regression
- Support Vector Machines (SVMs)
- K-Nearest Neighbor (k-NN)

Grunau (2022) nennt zudem spezifische Lernaufgaben wie Klassifikation, Regression, Objekterkennung sowie semantische Segmentierung und Instanzsegmentierung als Beispiele für überwachte Lernmethoden. Im Gegensatz dazu werden beim **unüberwachten Lernen** (engl.: Unsupervised Learning) keine annotierten Daten verwendet. Hier fehlen Zielvariablen, und das Hauptziel besteht darin, Muster oder Strukturen in den Daten zu erkennen. Ein bekanntes Verfahren im unüberwachten Lernen ist das Clustering, bei dem ähnliche Datenpunkte zu Gruppen zusammengefasst werden (Burkov, 2019; Grunau, 2022). Teilüberwachtes Lernen (engl.: Semi-Supervised Learning, SSL) kombiniert annotierte und nicht annotierte Daten, wobei die nicht annotierten Daten in der Regel überwiegen. Ziel dieser Methode ist es, das Beste aus beiden Datentypen herauszuholen, wobei sie dem Ansatz des überwachten Lernens ähnelt (Bhattacharyya et al., 2020; Burkov, 2019). Zu den teilüberwachten Methoden gehört auch das Reinforcement Learning (dt.: Bestärktes Lernen), bei dem das Modell durch positives oder negatives Feedback schrittweise verbessert wird. Positives Feedback belohnt korrekte Entscheidungen, während negatives Feedback Fehler signalisiert und so die Modellleistung optimiert (Burkov, 2019).

Im Laufe der Jahre haben sich die Methoden und Verfahren des Machine Learning kontinuierlich weiterentwickelt, wobei neue Ansätze und Anwendungen hinzugekommen sind. Die Entwicklung dieser Lernmethoden ist in der folgenden Abbildung dargestellt.

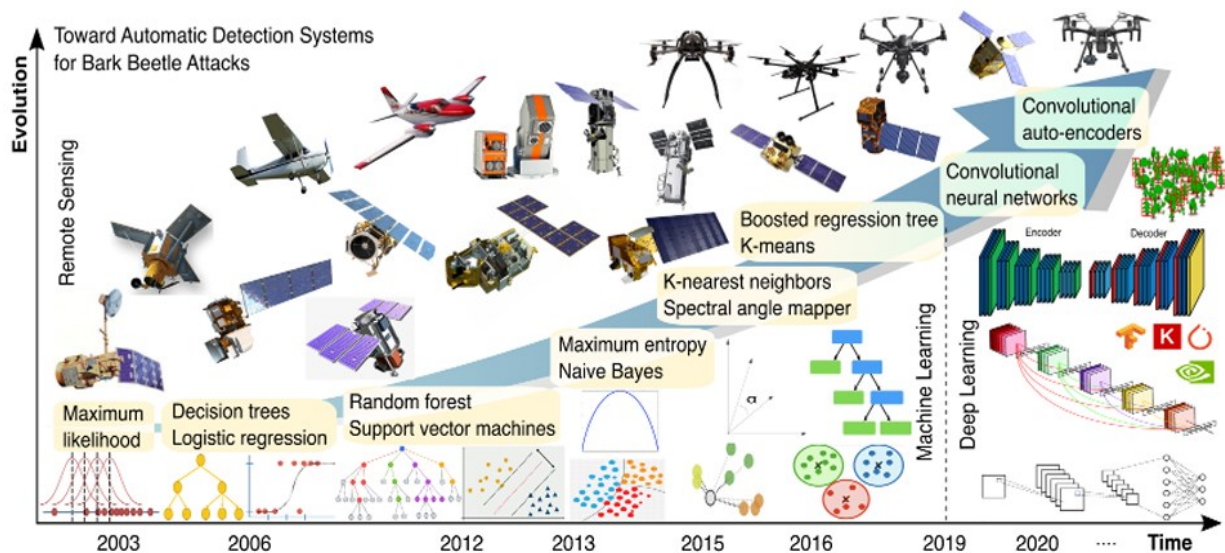


Abb. 10: Entwicklung von Machine Learning und Fernerkundung über die Jahre
(Marvasti-Zadeh et al., 2023)

3.2.2 Überwachtes Lernen im Detail

Im Kontext neuronaler Netze und unter Berücksichtigung des Fokus dieser Masterarbeit ist das überwachte Lernen besonders relevant, da für die Erkennung von Solaranlagen annotierte Daten verwendet werden. Dabei bestehen klare Zuordnungen zwischen den Eingabebildern und den gewünschten Ausgaben, z. B. Bounding Boxes oder Segmentierungen für Objekte. Das Modell lernt, Muster aus den Annotationen zu erkennen und diese korrekt zu klassifizieren. In diesem Fall bedeutet dies, dass das Modell auf Grundlage der bereitgestellten lernt, Photovoltaikanlagen entweder durch Objekterkennung oder Segmentierung zu identifizieren. Da die erwarteten Ergebnisse dieses Lernverfahrens im Voraus definiert sind (z. B. die genaue Position oder Form der Photovoltaikanlagen), können die Modelle anhand geeigneter Metriken bewertet werden.

Ein wesentlicher Unterschied zwischen Algorithmen des überwachten Lernens besteht darin, wie viel Verzerrung (engl.: Bias) und Varianz (engl.: Variance) sie erzeugen. Diese beiden Größen werden als Vorhersagefehler (engl.: Prediction Errors) bezeichnet. Bias beschreibt die Abweichung zwischen der erwarteten und der tatsächlichen, gemessenen Vorhersage. Varianz hingegen bezieht sich auf die Empfindlichkeit eines Algorithmus gegenüber einem bestimmten Satz von Trainingsdaten. Während des Trainingsprozesses werden sowohl Bias als auch Varianz reduziert, bis sie sich in einem optimalen Bereich befinden. Ein weiterer Vorhersagefehler, der irreduzierbarer Fehler (engl.: Irreducible Error), entsteht durch zufälliges Rauschen in den Daten. Dieser Fehler kann durch bessere Datenbereinigungsschritte verringert werden (Wlodarczak, 2019).

Das überwachte Lernen erfordert drei wesentliche Bestandteile: Input Features (dt.: Eingabevariablen), Target Features (dt.: Zielvariablen) und ein Maß zur Bewertung der Verbesserung. Die Input Features werden aus den Trainingsdaten extrahiert und dienen als Eingangsinformationen für das Modell. Die Target Features sind die Zielvariablen, also die erwarteten Ergebnisse, die das Modell vorhersagen soll.

Das Maß zur Bewertung der Verbesserung dient der Überprüfung und Bewertungen des Fortschritts oder die Optimierung des Modells in jedem Lernzyklus. Während des Trainings lernt das Modell, die Input Features auf die Target Features abzubilden. Der Trainingsprozess wird so lange angepasst, bis die gewünschte Genauigkeit erreicht ist. Häufig handelt es sich bei dieser Zuordnung jedoch um eine Annäherung, da vollständige Genauigkeit in der Regel nicht erreicht wird. Die Modellleistung wird mithilfe von Metriken wie z. B. Precision oder Recall gemessen (Burkov, 2019; Wlodarczak, 2019).

Ein vollständiger Trainingszyklus besteht in der Regel aus den folgenden Schritten (Wlodarczak, 2019):

1. Datenerfassung
2. Vorverarbeitung der Daten (engl.: Preprocessing)
3. Training eines Machine Learning-Algorithmus
4. Anpassung der Modell-Hyperparameter
5. Testen und Evaluierung des trainierten Modells
6. Berechnung des Fehlers

Ein **Algorithmus** ist ein wiederholbarer Prozess, der verwendet wird, um aus einer Menge von Daten ein trainiertes Modell zu erstellen. Jeder Algorithmus verhält sich unterschiedlich und kann je nach Problemstellung unterschiedlich effektiv sein. Das Trainieren eines solchen Algorithmus ist ein iterativer Prozess und durchläuft zahlreiche Zyklen, bis zufriedenstellende Ergebnisse erzielt werden. Die Genauigkeit, mit der ein Modell die realen Werte abbildet, wird unter anderem als Anpassungsgüte (engl.: Goodness of Fit) bezeichnet (Wlodarczak, 2019).

4 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANN; dt.: Künstliche Neuronale Netze, KNN) versuchen, die Aktivität der Neuronenschichten im Neokortex, einem Teil der Großhirnrinde, nachzuahmen. Das sind die faltigen 80% des Gehirns, in denen das Denken stattfindet (Bhardwaj et al., 2018; Vasuki & Govindaraju, 2017). ANN erlernen hierarchische Strukturen sowie die vielen Ebenen der Darstellung und Abstraktion, um die Muster von Daten zu verstehen, die aus verschiedenen Quellen stammen – seien es Bilder, Videos, Sprachsequenzen oder Texte (Bhardwaj et al., 2018; Bhattacharyya et al., 2020). Insbesondere das exponentielle Wachstum von Big Data motiviert die Entwicklung innovativer Methoden zur Extraktion von semantischen Informationen aus diesen Rohdaten (Hassaballah & Awad, 2020).

Inzwischen übertrumpfen künstliche neuronale Netze uns Menschen, nicht nur in ihrer Geschwindigkeit, sondern auch in der Genauigkeit ihrer Vorhersagen (Ng & Soo, 2018). Diese Erfolge fußen auf einigen wichtige Faktoren, die weiter in Kapitel 4.2 besprochen werden. Sie feiern Erfolge in verschiedenen Anwendungen der künstlichen Intelligenz, wie Computer Vision (CV), Natural Language Processing (NLP) und Spracherkennung (Bhardwaj et al., 2018).

4.1 Aufbau von Artificial Neural Networks

Zusammengefasst zeichnet sich ein künstliches neuronales Netzwerk aus durch (Goswami, 2020):

- Die Architektur: Das Gestaltungs- und Verbindungsmuster zwischen den Neuronen.
- Dem Algorithmus: Die Technik, mit der Gewichte (engl.: Weights) bestimmt werden.
- Die Aktivierungsfunktionen.

Ein künstliches neuronales Netz repräsentiert ein leistungsfähiges ML-Modell – ein Netzwerk, das durch eine Abfolge mathematischer Operationen charakterisiert ist. Ein solches neuronales Netz setzt sich aus **Neuronen** zusammen, die als Knoten in einem Graphen illustriert werden und Informationen in Form von Signalen verarbeiten (Grunau, 2022). Neuronen werden in **Schichten** (engl.: **Layers**) strukturiert, wobei mehrere solcher Schichten zustande kommen können (Jiang et al., 2019). Die Struktur eines neuronalen Netzes sieht wie folgt aus (Kelleher, 2019):

- Die Eingabeschicht (engl.: Input Layer), die die Daten repräsentiert.
- Mindestens zwei oder mehrere Zwischenschichten (engl.: Hidden Layers).
- Die Ausgabeschicht (engl.: Output Layer), die das Ergebnis darstellt.

Die Informationen wandern von der ersten Eingabeschicht zur letzten Ausgabeschicht, nachdem sie die dazwischenliegenden, versteckten Schichten mehrfach durchlaufen haben (Vasuki & Govindaraju, 2017). Ein oft für neuronale Netze verwendetes Schema stellt die Neuronen bzw. die verschiedenen Schichten in horizontaler Richtung dar. Je nach Anzahl der Schichten spricht man von z. B. einem vier-schichtigen Netz.

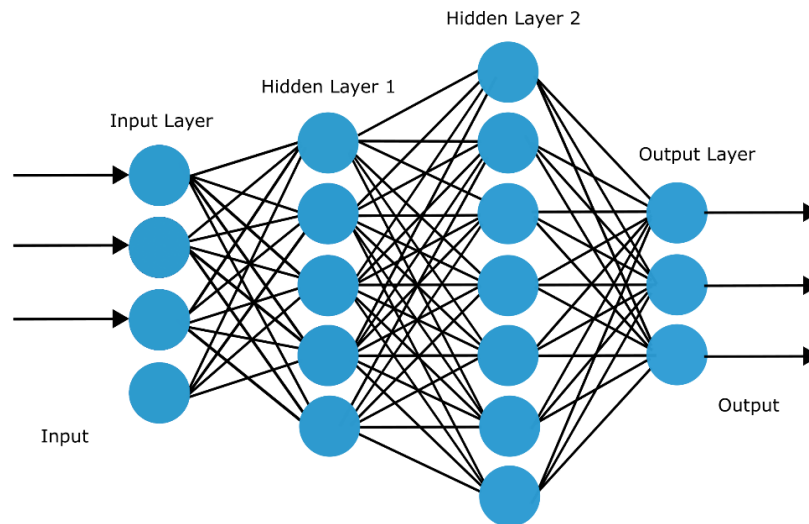


Abb. 11: Vier-Schichtiges neuronales Netz
(Eigene Darstellung, erstellt mit Inkscape 1.2)

Jedes der Neuronen nimmt eine Reihe von numerischen Werten als Input entgegen und bildet sie auf einen einzelnen Ausgabewert ab. Die Verbindung, die zwei Neuronen miteinander verknüpft, wird als Gewicht bezeichnet. Jedes dieser Gewichte ist in eine Richtung gerichtet, veranschaulicht durch Pfeile, was bedeutet, dass die Signale bzw. die Informationen in eine Richtung fließen – von der Eingabeschicht zur Ausgabeschicht (Kelleher, 2019). Wenn in einem Netzwerk die Signale in Vorwärtsrichtung laufen, werden diese „Feedforward“-Netzwerke (engl.: Feedforward Neural Networks, FFNR) bezeichnet. Hier gibt es keine sogenannte Rückkopplungsschleife (Bhattacharyya et al., 2020). Werden Informationen in umgekehrter Richtung weitergeleitet, spricht man von rekurrenten Netzen (engl.: Recurrent Neural Networks, RNN), die mindestens eine Rückkopplungsschleife aufweisen. Nach diesem Prinzip funktioniert u. a. der Backpropagation-Algorithmus (Vasuki & Govindaraju, 2017).

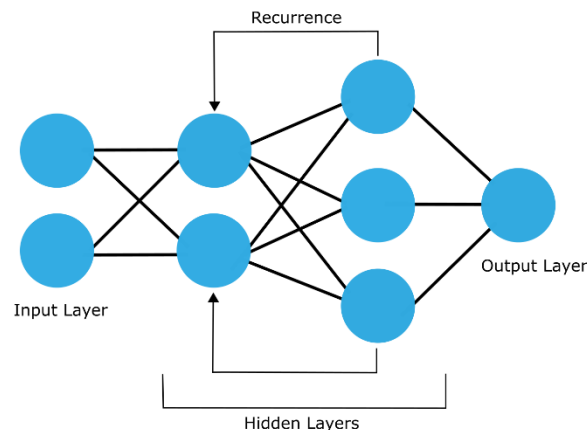


Abb. 12: Rekurrentes neuronales Netz
(Eigene Darstellung, erstellt mit Inkscape 1.2)

In den Neuronen der Eingabeschicht findet keine Verarbeitung von Informationen statt. Die Ausgabe jedes einzelnen Neurons entspricht dem Wert der gespeicherten Daten (Kelleher, 2019). Mit den vielen angeordneten Neuronen in einem Layer kann eine Funktion erstellt werden, und viele dieser Layer können (iterativ) gestapelt werden. Die ausgehenden Aktivierungen der Neuronen in einem Layer werden zu den Eingabesignalen für die Neuronen im nächsten Layer umgewandelt. Diese unterschiedlichen Verbindungsstrukturen werden als Architekturen eines neuronalen Netzes bezeichnet (Jiang et al., 2019; Roberts et al., 2022). Der Funktionswert dieser Funktionen, der auch als

Aktivierung bezeichnet wird, wird durch die Eingabe, Gewichtung der Eingabe und den Bias bestimmt und im Neuron gespeichert (Grunau, 2022).

In der ersten Verarbeitungsstufe erfolgt die Ermittlung einer gewichteten Summe der Eingaben für das Neuron. Diese Eingaben werden gewichtet, wobei jede Verbindung in einem Netzwerk ein Gewicht hat, das mit ihr verbunden ist. Die Verbindungsgewichte erhalten fortlaufend Werte bzw. einfache Zahlen, mit denen die entsprechenden Eingabe-Neuronen multipliziert werden. Auf diese Weise hat jede Eingabe eine unterschiedliche Wirksamkeit auf die nachfolgenden Neuronen in den darauffolgenden Schichten und darauf, wie diese Neuronen die Informationen verarbeiten, die sie entlang der Verbindungen erhalten (Grunau, 2022; Kelleher, 2019). Das Trainieren eines neuronalen Netzes besteht darin, den besten Satz von Gewichten zu ermitteln (Kelleher, 2019).

In der zweiten Verarbeitungsstufe, die innerhalb des Neurons liegt, durchläuft das Resultat der gewichteten Summe eine zweite Funktion - die Aktivierungsfunktion (Kelleher, 2019). Jedes Neuron kann ein Bias haben, das zur Aktivierung bzw. zum Funktionswert hinzuaddiert wird. Es erfolgt eine Anpassung des Funktionswertes eines Neurons, um die sogenannte Approximationsfähigkeit zu vergrößern. Dies hat den Nebeneffekt, dass der Wertebereich der Aktivierung beeinflusst wird (Grunau, 2022). In der Ausgabeschicht bestimmt die Aktivierungsfunktion, ob das Netz feuert oder nicht (Roberts et al., 2022). Hierbei können je nach Anforderungen an Einfachheit oder Komplexität unterschiedliche Funktionen verwendet werden. Bekannte Aktivierungsfunktionen sind die ReLu-Aktivierungsfunktion, die Sigmoidfunktion sowie logistische und Tanh-Aktivierungen. Seit deren Einführung werden hauptsächlich nichtlineare Aktivierungsfunktionen genutzt (Kelleher, 2019).

Die Anzahl und Typen der Neuronen sowie die Anzahl der in sich verketteten Schichten sind entscheidend dafür, wie komplex künstliche neuronale Netze im Deep Learning-Kontext sind (Grunau, 2022). Was solche Netze ausmacht, sind die vielen Zwischenschichten, die die eingespeisten Daten durchlaufen müssen. Es sind mindestens zwei erforderlich, damit von einem tiefen Netzwerk die Rede ist. Auch die Ausgabeschicht spielt in diesem Zusammenhang eine wichtige Rolle (Vasuki & Govindaraju, 2017). Es ist jedoch zu beachten, dass eine große Anzahl von Schichten nicht unbedingt die Genauigkeit der Prognose erhöht. Stattdessen ist mit einer hohen Rechenzeit zu rechnen, weshalb vorher überlegt werden sollte, ob für die Aufgabe einige wenige Zwischenschichten bereits ausreichen (Ng & Soo, 2018). Aktuelle Anwendungen neuronaler Netze weisen im Gegensatz zum Perceptron eine Vielzahl von Neuronen und Schichten auf, die als Deep Neural Networks (DNN) bezeichnet werden. Aufgrund der großen Anzahl an Parametern, genauer gesagt den Gewichten und dem Bias, sind umfangreiche Datensätze erforderlich, um diese Netze zu trainieren (Grunau, 2022). Die archetypische Architektur eines ANN, die auf dem Prinzip der Schichtung vieler Neuronen beruht, wird "mehrschichtiges Perzeptron" (engl.: Multilayer Perceptron, MLP) genannt (Roberts et al., 2022).

In Kapitel 5.6 wird näher auf den Lernprozess bzw. das Training von ANN eingegangen. Dort werden speziell Parameter betrachtet, auch Hyperparameter genannt, mit denen bessere Modellergebnisse erzielt werden können.

4.2 Entwicklung und Fortschritte in der Technologie

Neuronale Netze sind von besonderem Interesse, weil sie die Grundlage für eine drastische Erweiterung der Rechenfähigkeiten bietet, die von Hochgeschwindigkeitsarithmetik und Datenabfrage

bis hin zu grundlegenden, aber anspruchsvolleren Anforderungen in der Informationsverarbeitung reicht (Openshaw & Openshaw, 1997). Die Fortschritte in der VLSI-Technologie (Very-Large-Scale Integrated Circuit) ermöglichten den Bau paralleler Hardware mit Tausenden von Prozessoren. Künstliche neuronale Netze waren deshalb von Interesse, weil mit ihrer Hilfe Berechnungen auf eine große Anzahl von parallellaufenden Verarbeitungseinheiten verteilt werden konnte. Sie waren auch deshalb besonders interessant, weil sie aus Dateninputs lernen können (Alpaydin, 2021; Openshaw & Openshaw, 1997).

KNN werden daher nicht im herkömmlichen Sinne programmiert, sondern bringen sich das Programmieren selbst bei, indem sie eine bestimmte Aufgabe anhand von Beispielen erlernen. Bei der üblichen Computerprogrammierung weist der Programmierer den Computer an, eine Aufgabe anhand eines explizit vordefinierten Algorithmus zu lösen. Die Leistung des Programms hängt stark von den Fähigkeiten des Programmierers ab und ist eine Erweiterung seiner Intelligenz. Wenn der Algorithmus nicht funktioniert, schlägt das Programm fehl und der Code muss vom Programmierer abgeändert werden. Es gibt keinen Mechanismus, mit dem das Programm lernen kann, sich selbst zu reparieren oder seine Codes aus der Erfahrung heraus zu optimieren. Bei einem neuronalen Netz gibt es zwar noch ein Computerprogramm, aber das Programm enthält keine spezifischen Algorithmen, die für die Lösung eines bestimmten Problems von Bedeutung sein können. Das neuronale Netz lernt, wie eine bestimmte Aufgabe auszuführen ist. Dies ist eine grundlegend andere Technologie. Es muss kein einmaliges Programm geschrieben werden, da die Software für neuronale Netze anwendungsunabhängig ist. KNN sind nicht-algorithmisch und können Probleme lösen, für die es keine bekannten Algorithmen gibt. Als solche werden sie zur Lösung von Problemen als ein wichtiges neues Paradigma für die Programmierung von Computern angesehen (Alpaydin, 2021; Openshaw & Openshaw, 1997).

In den letzten zehn Jahren, in denen mehr Daten und Rechenleistung zur Verfügung standen, wurden viele beeindruckende Anwendungen von tiefen neuronalen Netzen bekannt, die manchmal aus Hunderten solcher Schichten bestanden. So werden z. B. Netze, die tiefer als LeNet sind, in der Gesichtserkennung eingesetzt, und ein Netz, das tiefer als das von TD-Gammon ist, hat gelernt, Go zu spielen (Alpaydin, 2021). Im Spiel Go sind deutlich mehr Spielkonfigurationen möglich als beispielsweise beim Schach. Dies liegt daran, dass das Spielbrett größer ist und zugleich die einfacheren Spielregeln mehr Brett- und somit auch mehr Spielkonfigurationen erlauben (Zhu et al., 2017). Das jüngste Interesse an Machine Learning ist weitgehend auf solche Erfolge mit Deep Learning-Netzen zurückzuführen.

4.2.1 Erfolgsfaktoren für Artificial Neural Networks und Deep Learning

Die Praktikabilität von Machine Learning hat sich bereits bewährt, und die Zahl der Anwendungen nimmt täglich in unterschiedlichen Bereichen zu. In den vergangenen Jahrzehnten hat sich die Computer-Hardware mit jeder neuen Mikroprozessor-Generation kontinuierlich weiterentwickelt. Beginnend mit 8- über 16- bis hin zu 32-Bit-Prozessoren ermöglichte jede neue Generation eine höhere Rechenleistung pro Zeiteinheit sowie die Nutzung von mehr Speicherplatz, was zu einer stetigen Verbesserung der Computer führte. In ähnlicher Weise erfolgte auch eine Weiterentwicklung der Computersoftware von einer Programmiersprache zur nächsten. Jede neue Sprache vereinfachte die Programmierung und eröffnete neue Möglichkeiten für komplexere Berechnungen. Parallel dazu

entwickelte sich die Computertechnologie von einem Betriebssystem zum nächsten. Jede aufkommende Version vereinfachte die Bedienung der Computer und ermöglichte die Unterstützung eines breiteren Spektrums an Anwendungen. Heute verlagert sich die Datenverarbeitung zunehmend auf intelligente Geräte und Anwendungen, die eine immer größere Rolle im Alltag spielen (Alpaydin, 2021).

Im Zuge der zunehmenden Digitalisierung verlagert sich unser Leben immer stärker in den digitalen Raum, was zu einer stetig wachsenden Menge an Daten führt. DL-Algorithmen zeigen bei der Verarbeitung großer Datenmengen deutlich bessere Ergebnisse als klassische Machine-Learning-Methoden (Bhattacharyya et al., 2020). Die ersten Festplatten für Personalcomputer (PC) hatten eine Kapazität von lediglich fünf Megabyte. Gegenwärtig verfügen gewöhnliche Computer über Speichergrößen von fünfhundert Gigabyte – eine Steigerung um das Hunderttausendfache innerhalb von etwa drei Jahrzehnten. Moderne Datenbanken können mittlerweile problemlos Hunderte von Terabyte speichern, wobei zunehmend auch das Petabyte als Maßeinheit verwendet wird. Neben dem enormen Anstieg der Speicherkapazität sind dank technologischer Fortschritte auch die Verarbeitungsgeschwindigkeit und die Kosten pro Recheneinheit drastisch verbessert worden. Dies ist sowohl auf schnellere Computerchips als auch auf parallele Architekturen zurückzuführen, die Tausende gleichzeitig ablaufender Prozesse ermöglichen. Jeder dieser Prozesse löst dabei einen Teil eines umfassenden Problems (Alpaydin, 2021; Ng & Soo, 2018).

Weitere wichtige Fortschritte betreffen **Hochleistungs-GPUs** (Grafikprozessoren), die den Lernprozess von neuronalen Netzen bei großen Datenmengen erheblich beschleunigen. Dank ihrer höheren Rechenleistung können GPUs die Trainingszeit deutlich verkürzen (Bhattacharyya et al., 2020). Grafikprozessoren sind dabei bis zu 150-mal schneller als herkömmliche CPU-Mikroprozessoren und sind daher ideal für Deep-Learning-Aufgaben geeignet (Ng & Soo, 2018). Es wird prognostiziert, dass sich die Entwicklung von vielseitigen PCs hin zu spezialisierten, intelligenten Geräten weiter an Geschwindigkeit gewinnen wird. Menschen besitzen heute nicht mehr ausschließlich einen zentralen PC, der sämtliche Daten speichert und verarbeitet. Stattdessen erfolgt die Speicherung der Daten zunehmend in der Cloud, wodurch ein ortsunabhängiger Zugriff von verschiedenen Geräten aus möglich wird. Machine Learning wird dazu beitragen, eine immer komplexer werdende Welt besser zu verstehen. Bereits in der Gegenwart stehen wir einer Datenflut gegenüber, die unsere sensorischen Fähigkeiten und kognitiven Kapazitäten übersteigt (Alpaydin, 2021).



Abb. 13: NVIDIA-Grafikkarte
(NVIDIA, o. J.)

Die Genauigkeit und Leistungsfähigkeit von Deep Learning-Algorithmen konnte durch den Einsatz neuer Werkzeuge und Methoden deutlich gesteigert werden. Im Vergleich zu klassischen Machine Learning-Ansätzen haben moderne, effizientere DL-Algorithmen bei identischen Problemstellungen und Datensätzen die Fehlerquote erheblich reduziert (Bhattacharyya et al., 2020). Zusammenfassend tragen verschiedene Faktoren zu diesen Fortschritten bei: Verbesserungen in der Hardware, die größere und effizientere Speicherkapazitäten bereitstellen, GPUs mit höherer Rechenleistung, die das Training beschleunigen, sowie der optimierte Umgang mit großen Datenmengen. Die Verfügbarkeit von offenen Daten (Open Data) spielt ebenfalls eine zentrale Rolle, da sie den Lernprozess intensiviert und den Zugang zu umfangreichen Datensätzen erleichtert.

5 Deep Learning

Deep Learning (dt.: Tiefes Lernen, DL) ist ein moderner Forschungszweig des Machine Learning (ML) und gehört zur künstlichen Intelligenz (KI). Sein Ziel es ist, ML stärker an die ursprünglichen Ziele der KI heranzuführen (Bhardwaj et al., 2018; Bhattacharyya et al., 2020). Deep Learning kann als eine Technik zur Implementierung von Machine Learning angesehen werden und hat den Bereich der künstlichen Intelligenz revolutioniert (Poonkuntran et al., 2022). Besonders im Bereich der Computer Vision hat DL mit seinen Fortschritten beeindruckende Ergebnisse erzielt (Zhang et al., 2023). Der Begriff tief bezieht sich auf die Komplexität der Modelle, insbesondere auf die Anzahl der Schichten im neuronalen Netz (Grunau, 2022).

Trotz der jüngsten Popularität von Deep Learning verfügt diese Technologie über eine lange Geschichte. Ihre Ursprünge reichen bis in die 1940er-Jahre zurück, und sie hat sich aus künstlichen neuronalen Netzen (KNN) im Bereich der künstlichen Intelligenz und des Machine Learning entwickelt (Jiang et al., 2019). Fortschritte in mathematischen Methoden, leistungsstärkere Computer und der Zugang zu großen Datensätzen haben dazu beigetragen, dass Deep Learning heute eine zentrale Rolle in technologischen Innovationen spielt und in zahlreichen Bereichen Anwendung findet (Bhardwaj et al., 2018; Poonkuntran et al., 2022). Ein wesentlicher Vorteil von Deep Learning gegenüber ML besteht darin, dass Merkmalsextraktion und Datenvorverarbeitung größtenteils automatisch erfolgen, wodurch der Aufwand für die manuelle Vorbereitung der Daten deutlich reduziert wird. Deep Learning-Algorithmen lernen aus Daten, ohne dass Expertenwissen in jedem Schritt erforderlich ist (Kumar & Renuka, 2023). Es wird erwartet, dass Deep Learning in Zukunft zahlreiche Lebens- und Arbeitsbereiche weiter erleichtert. Die meisten Architekturen basieren auf neuronalen Netzen, die von neurowissenschaftlicher Forschung inspiriert sind (Jiang et al., 2019).

5.1 Funktionsweise von Deep Learning

Herkömmliche Neural Networks bestehen in der Regel nur aus zwei oder drei Schichten, während DL-Netze Hunderte solcher Schichten haben können. Es gibt keine feste Definition, die festlegt, wie viele Layer ein Netz haben muss, um als tief zu gelten. Die Lerntechniken von Deep Learning basieren auf Ansätzen, die auch bei ML und Artificial Neural Networks verwendet werden (Bhattacharyya et al., 2020). Jiang et al. (2019) definieren Deep Learning als eine Klasse von ML-Algorithmen, die sich durch viele Schichten von nichtlinearen Verarbeitungseinheiten auszeichnen. Sie erläutern dazu:

„Deep learning approaches are a class of machine learning algorithms that use many layers of nonlinear processing units for representations and transformations. Each layer uses the output from the previous layer as input and the hierarchical representations can be obtained by different levels of abstraction. These algorithms may be trained in a supervised or an unsupervised way, and their applications include pattern analysis (unsupervised) and classification (supervised). Currently, the rapid development of deep learning has been accelerated by three key reasons, i.e., massive data, powerful computation ability, and novel algorithms.” (Jiang et al., 2019).

Deep-Learning-Algorithmen zeichnen sich dadurch aus, dass ihre Lernkapazität proportional zur verfügbaren Datenmenge steigt. Im Gegensatz dazu erreichen flache oder traditionelle Lernalgorithmen nach einer bestimmten Datenmenge ein Leistungslimit, ab dem keine signifikanten

Verbesserungen mehr erfolgen. Die Tiefe der Architektur bezieht sich auf die Anzahl der Ebenen der Zusammensetzung nichtlinearer Operationen. Zu diesen Operationen gehören die gewichtete Summe (engl.: Weighted Sum), das Produkt, ein einzelnes Neuron, der Kernel etc. Die meisten Lernalgorithmen entsprechen flachen Architekturen, die eins bis drei Ebenen haben (Bhardwaj et al., 2018).

Verschiedene Deep Learning-Architekturen wie „Deep Belief Networks“ (DBN), „Convolutional Neural Networks“ (CNN), „Recurrent Neural Networks“ (RNN), „Stacked Autocoder“ (SAE) und „Generative Adversarial Networks“ (GAN) erzielen bei vielen Aufgaben hervorragende Ergebnisse und finden daher in diversen Bereichen erfolgreich Anwendung (Jiang et al., 2019). Beispielsweise sind Recurrent Neural Networks besonders für Anwendungen geeignet, die sequenzielle Daten nutzen, wie etwa Zeitreihenanalysen, Übersetzungssoftware oder Sprachassistenten (Grunau, 2022). Die genannten Architekturen sind grundlegend und zeichnen sich durch spezifische Eigenschaften aus, insbesondere in Bezug auf die Anordnung und den Typ ihrer Schichten. Sie können erweitert oder miteinander kombiniert werden, um neue Anwendungsbereiche zu erschließen und komplexere Aufgabenstellungen zu bewältigen (Jiang et al., 2019).

5.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN; dt.: Faltungsneuronalen Netze) sind dem mehrschichtigen Perzeptron nachempfunden. Sie haben sich unter den Architekturen künstlicher neuronaler Netze als Standard etabliert, da sie bei zahlreichen Anwendungen im Bereich des Computer Vision eine annähernd menschliche Genauigkeit erreichen können (Kiranyaz et al., 2021). Neuronale Netze, einschließlich CNN, in vielfältigen Anwendungsbereichen eingesetzt, etwa in der Spracherkennung, bei maschineller Übersetzungen, in der Verarbeitung handgeschriebener Dokumente oder in der medizinischen Diagnostik (Hassaballah & Awad, 2020; Vasuki & Govindaraju, 2017). Besonders im Bereich Computer Vision erzielen CNN beeindruckende Ergebnisse, beispielsweise in der Bildklassifizierung, der Muster- und Objekterkennung sowie der Bildsegmentierung.

In den Zwischenschichten wird eine mathematische Operation namens Konvolution (dt.: Faltung) durchgeführt, bei der Filter auf die Eingabedaten angewendet werden. Dies ermöglicht es, räumliche Muster wie Kanten, Formen oder Texturen zu erkennen (Grunau, 2022).

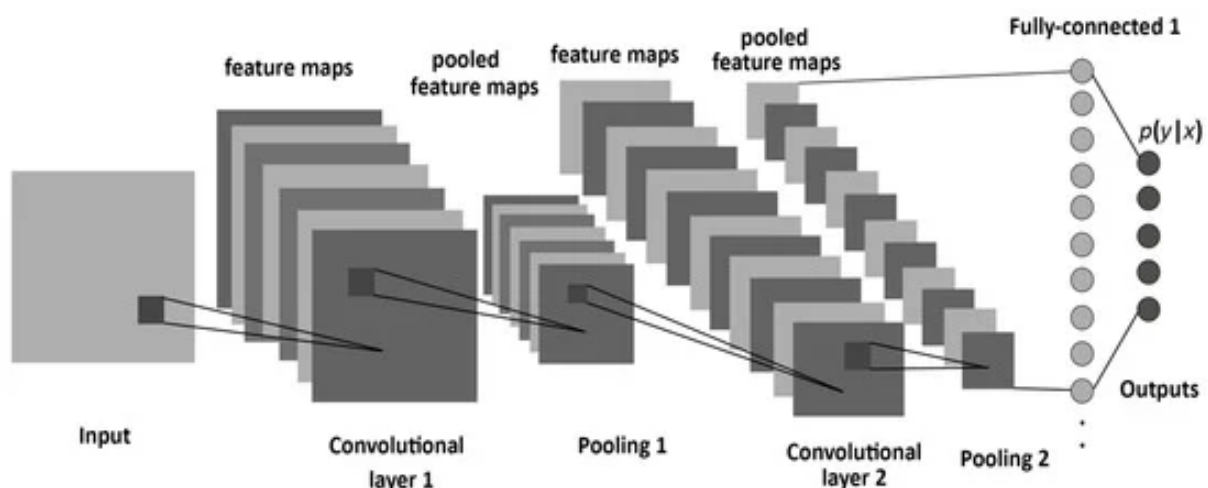


Abb. 14: Convolutional Neural Networks
(Albelwi & Mahmood, 2017)

Die hohe Anzahl an Schichten in CNN stellte in der Vergangenheit eine Herausforderung dar, da beim Training durch den Gradientenabstiegsalgorithmus Probleme wie der verschwindende oder explodierende Gradient auftraten. Dank moderner Optimierungsalgorithmen und Techniken, wie Batch-Normalisierung und Residual-Netzen, treten diese Probleme heutzutage kaum noch auf bzw. wurden sie stark verringert. Im Kontext von Deep Learning bezieht sich der Begriff auch auf das Training solcher Netzwerke unter Verwendung moderner, effektiver Lernalgorithmen, unabhängig von der Tiefe des Modells (Bhattacharyya et al., 2020; Burkov, 2019).

5.3 Computer Vision

Das menschliche Sehsystem besteht aus den Augen, die Licht aufnehmen, aus Neurorezeptoren, die die Weiterleitung ermöglichen, und aus dem visuellen Cortex (auch Sehrinde genannt), der diese Informationen verarbeitet (Kumar & Renuka, 2023). Die Fähigkeit des Gehirns, visuelle Reize zu entschlüsseln, ist außergewöhnlich, etwa Objekte spontan zu unterscheiden, Gesichter wiederzuerkennen oder visuelle Muster zu identifizieren. Computer hingegen betrachten Bilder zunächst als bloße Anordnung von Pixeln, also Matrizen aus Rot-Grün-Blau-Werten, denen keine Bedeutung zugeordnet ist (Planche & Andres, 2019). Computer Vision (dt.: Maschinelles Sehen, CV) ist ein Bereich der Informatik, der sich mit der Entwicklung digitaler Systeme zur Verarbeitung, Analyse und Interpretation visueller Daten wie Bilder und Videos befasst. Ziel ist es, Computern beizubringen, visuelle Eingaben ähnlich wie Menschen zu verstehen und semantische Informationen, wie Objekte, deren Positionen und Anzahl, aus den Pixelwerten extrahiert werden (Kerim et al., 2021; Kumar & Renuka, 2023; Poonkuntran et al., 2022). Computer Vision ist interdisziplinär und verbindet Erkenntnisse aus Informatik, Mathematik, Physik und Biologie. Diese Vielfalt ermöglicht es, komplexe visuelle Muster zu analysieren und zu interpretieren (Planche & Andres, 2019).

Computer Vision in Verbindung mit Deep Learning hat viele Anwendungen bzw. Problemstellungen (Aufgaben), darunter fallen u. a. Bildklassifizierung, Objekterkennung, Gesichtserkennung und Segmentierung (Poonkuntran et al., 2022). Die rasanten Fortschritte auf dem Gebiet von CV wurden maßgeblich durch lernbasierte Methoden vorangetrieben, insbesondere durch solche, die auf Deep Learning basieren. Die Verfügbarkeit großer Trainingsdatensätze ist jedoch ein wichtiger Faktor, um von solchen Ansätzen profitieren zu können. Ein Hindernis bei der Lösung vieler Computer Vision-Aufgaben ist das Fehlen genauer, vielfältiger sowie umfangreicher Datensätze. Das manuelle Zusammenstellen von Daten ist dagegen sehr arbeitsintensiv und fehleranfällig (Kerim et al., 2021). Im Kontext von Computer Vision sind Merkmale (Features) besonders wichtig für die Identifikation und Klassifizierung von Objekten. Diese Merkmale, die bereits in Kapitel 3.2 erwähnt wurden, sind in Computer Vision entscheidend, um detaillierte Informationen aus Bilddaten zu extrahieren und die Aufgabe der Bildanalyse zu unterstützen (Dadhich, 2018; Planche & Andres, 2019). Es gibt verschiedene Anwendungen, bei denen Bilddaten im Mittelpunkt stehen. Die folgenden Aufgaben sind der überwachten maschinellen Lernmethode zuzuschreiben. Ein Convolutional Neural Network ist als Deep Learning-Modell darauf ausgelegt, mit Bildmaterial zu arbeiten.

5.3.1 Bildklassifizierung

Unter Bildklassifizierung (engl.: Image Classification) versteht man den Prozess der Zuweisung einer Annotation zu einem Bild basierend auf dessen visuellem Inhalt. Das bedeutet, dass mit dem „Labeling“ eine Klassifizierung (Bildung von Kategorien) durchgeführt wird. Während des Lernprozesses wird dem

Eingabebild aus einem festen Satz von Annotationen die wahrscheinlichste Annotation zugewiesen und es somit einer Kategorie zugeordnet. Die Bildklassifizierung hat eine Vielzahl praktischer Anwendungen. Andere Bildverarbeitungsaufgaben, wie zum Beispiel Objekterkennung und Segmentierung, lassen sich methodisch auf die Bildklassifizierung herunterbrechen (Dey, 2020).

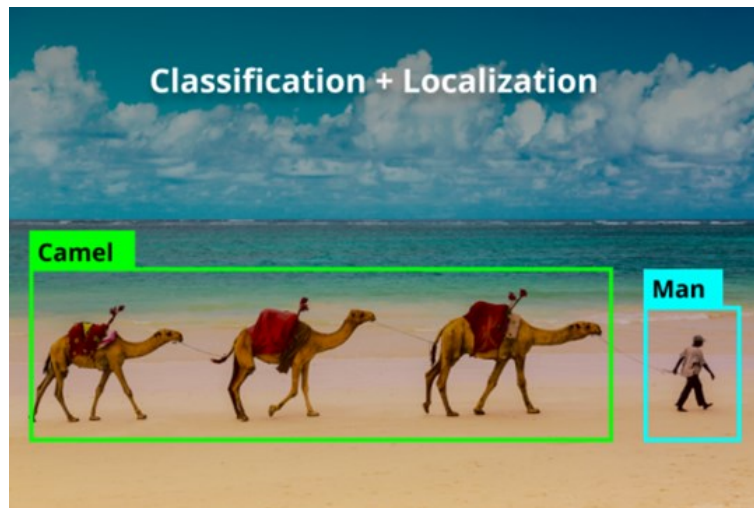


Abb. 15: Bildklassifizierung inklusive Lokalisierung
(DeepLobe, 2021)

5.3.2 Objekterkennung

Objekterkennung ist eine Aufgabe der Computer Vision, bei der semantische Instanzen von Objekten eines bestimmten Typs in Bildern erkannt werden. Besonders relevant sind in diesem Kontext z. B. Menschen, Fahrzeuge und Gebäude.

Um die Objekterkennung zu verstehen, müssen zuvor die Begriffe Bildklassifizierung und Lokalisierung verstanden werden. Die Bildklassifizierung zielt darauf ab, eine Annotation für ein Bild vorherzusagen, die am ehesten zutrifft, während die Objektlokalisierung darauf abzielt, die Positionen von Objekten zu identifizieren und Bounding Boxes rund um diese Objekte zu zeichnen. Bounding Boxes haben eine rechteckige Form (siehe Abbildung 16). Objekterkennung kombiniert also die Aufgaben der Klassifizierung und Lokalisierung. Sie zeichnet eine Bounding Box um jedes Objekt bzw. um mehrere Objekte in einem Bild und weist ihnen eine Annotation zu. Diese Problemstellung wird zusammen als Objekterkennung bezeichnet (Aggarwal, 2018; Dey, 2020; Kumar & Renuka, 2023).

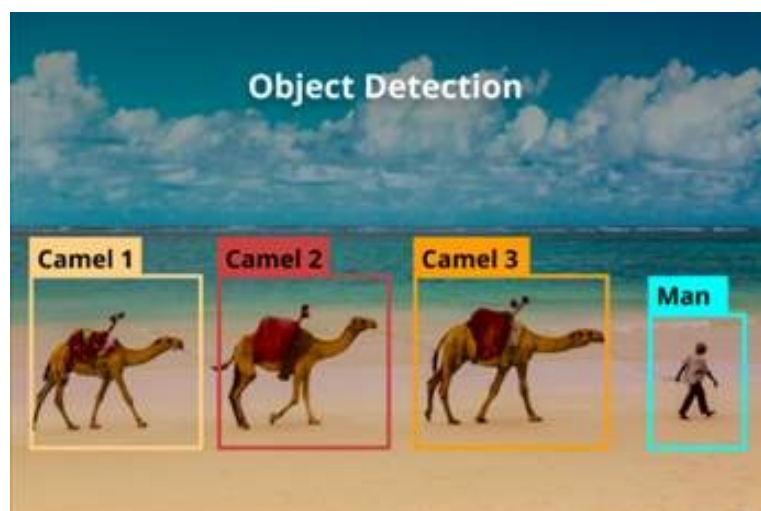
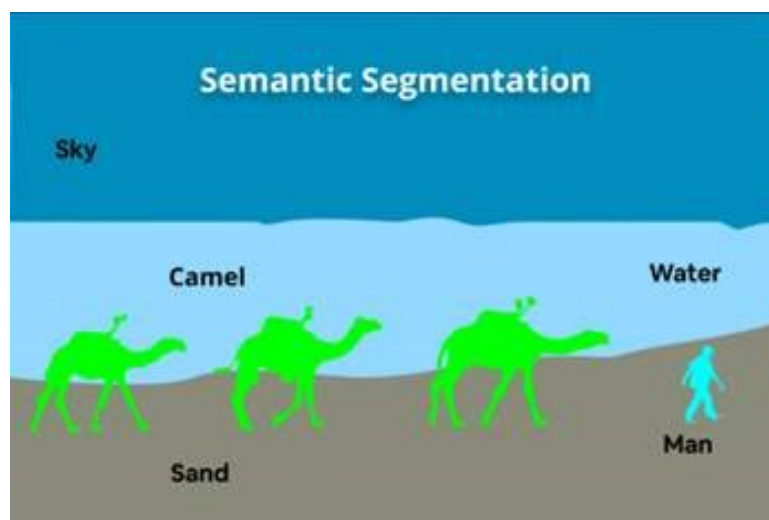


Abb. 16: Objekterkennung
(DeepLobe, 2021)

5.3.3 Semantische Segmentierung und Instanzsegmentierung

Die Bildsegmentierung ist eine Technik, die ein Bild in verschiedene Kategorien bzw. Regionen unterteilt. Dabei wird das Bild in der Regel auf Basis der Pixel unterteilt, wobei jede Region Pixel mit ähnlichen Attributen enthält und jedes dieser Pixel einer dieser Kategorien zugeordnet wird (Aggarwal, 2018; Dey, 2020). Bildsegmentierung ist ein Vorverarbeitungsschritt von Computer Vision-Systemen. Anschließend können Bildmerkmale bzw. Features für die Bildklassifizierung und Bilderkennung extrahiert werden (Dey, 2020; Kumar & Renuka, 2023).

Die **semantische Segmentierung** hat das Ziel, die Darstellung eines Bildes in einzelne Segmente zu vereinfachen, damit die Bildanalyse leichter durchgeführt werden kann und mehr Aussagekraft erhält. Bei einer gelungenen Segmentierung, die sowohl qualitativ hochwertig als auch zuverlässig ist, werden alle nachfolgenden Schritte der Analyse vereinfacht, was zu einer insgesamt erfolgreicherem Bildanalyse führt. Ein Bild fehlerfrei in die richtigen Segmente zu unterteilen, ist jedoch häufig ein schwieriges Unterfangen (Aggarwal, 2018; Dey, 2020). Zudem kann die pixelgenaue Annotation großer Datensätze sowohl schwierig als auch arbeitsintensiv sein. Die semantische Bildsegmentierung hat in den letzten Jahren, besonders mit dem Aufkommen von Deep Learning, enorme Fortschritte erzielt (Hassaballah & Awad, 2020).



*Abb. 17: Semantische Segmentierung
(DeepLobe, 2021)*

Bei der **Instanzsegmentierung** hingegen werden mehrere Objekte derselben Klasse als verschiedene, individuelle Instanzen behandelt. Beispielsweise würde jedes Kamel in Abbildung 18 der Klasse „Kamel“ zugeordnet werden, aber jeweils eine andere Farbe erhalten, da es sich um verschiedene Instanzen derselben Klasse handelt (DeepLobe, 2021).

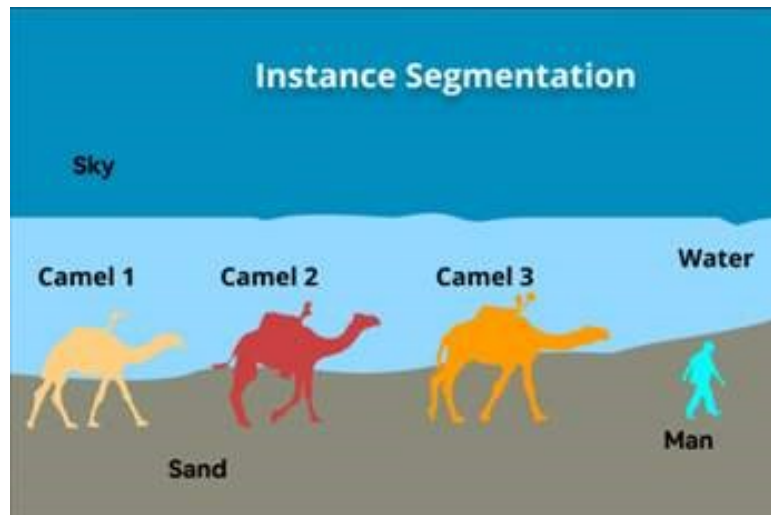


Abb. 18: Instanzsegmentierung
(DeepLobe, 2021)

5.4 Digitale Bilddarstellung in der Computer Vision

Convolutional Neural Networks sind für Aufgaben konzipiert, die mit gitterstrukturierten Eingaben arbeiten, also Daten, die in mehreren Reihen vorliegen und welche starke räumliche Abhängigkeiten in lokalen Regionen des Gitters aufweisen (Aggarwal, 2018). Ein digital gespeichertes Bild besteht aus einer solchen Gitterstruktur und wird von einem Computer als ein zweidimensionales Feld (bzw. eine 2D-Matrix) wahrgenommen, wobei jede Rasterzelle dieses Feldes Intensitätswerte zur Darstellung des Bildes enthält. Ein einfaches Bild ist ein Schwarz-Weiß-Bild, bei dem 0 für Weiß und 1 für Schwarz steht, was als Binärbild bezeichnet wird (Dadhich, 2018; Ramo, 2019). Es gibt auch andere Beispiele für gitterstrukturierte Daten: Eindimensionale Daten können Signale und Sequenzen sein, einschließlich der Sprache. Zweidimensionale Daten umfassen, wie oben genannt, Bilder oder Audiodateien. Dreidimensionale gitterstrukturierte Daten sind u. a. Videos oder volumetrische Bilder. Diese verschiedenen Datentypen weisen jeweils unterschiedliche Arten von Beziehungen zwischen benachbarten Elementen auf (Aggarwal, 2018; LeCun et al., 2015).

Neuronale Netze sagen weitgehend binäre Klassen voraus, bei denen die Antwort entweder „Ja“ oder „Nein“ bzw. „richtig“ oder „falsch“ ist (Ramo, 2019). Eine Erweiterung dieser Klassifikation ist die Unterteilung von Schwarz und Weiß in eine breiter gefasste Grauskala (engl.: Grayscale), ein Kanal, der die Intensität der Pixel in einem Wertebereich von 0 bis 255 repräsentiert. In einer dreidimensionalen Ansicht eines Bildes sind die X- und Y-Achsen die Pixelpositionen und die Z-Achse repräsentiert den Intensitätswert. Bilder können verschiedene Formate haben. Für jedes Format haben die Werte in den Rasterzellen einen anderen Wertebereich (Dadhich, 2018).

Bilder weisen räumliche Abhängigkeiten auf, da benachbarte räumliche Bereiche meist ähnliche Farbwerte der einzelnen Pixel besitzen. Eine zusätzliche Dimension erfasst die verschiedenen Farben, wodurch eine dreidimensionale Eingabemenge entsteht. Ein Bild wird wie ein 3D-Objekt behandelt. In einem CNN sind Merkmale aufgrund der räumlichen Abstände zueinander abhängig (Aggarwal, 2018; Bhattacharyya et al., 2020).

5.4.1 Graustufenbilder

Ein Bild besteht also aus Pixeln, die entsprechend dem darin gespeicherten Wert visualisiert werden. Jeder in einem Pixel gespeicherte Wert ist mit einer festen Darstellung verknüpft. Beispielsweise kann ein Pixelwert von 10 den Grauintensitätswert 10 oder den Blauintensitätswert 10 darstellen (Dadhich, 2018). Die Pixelwerte eines Schwarz-Weiß-Bildes stellen die Intensität des Lichts dar. Die Zahl 0 bedeutet Weiß, und je näher die Zahl 255 heranrückt, desto dunkler wird das Pixel. Als Beispiel: Ein Bild mit den Maßen 4×7 sowie 28×28 muss für die Verarbeitung in einen eindimensionalen Vektor umgewandelt werden. Dies bedeutet, dass ein Bild mit den Maßen 28×28 in ein Bild mit 784×1 und ein Bild mit den Maßen 4×7 in ein Bild mit 28×1 umgewandelt wird. Der eindimensionale Vektor unterscheidet sich nicht von einem binären Vektor. Jedes dieser Pixel ist nur ein Merkmal für eine Computer Vision-Anwendung (Ramo, 2019).

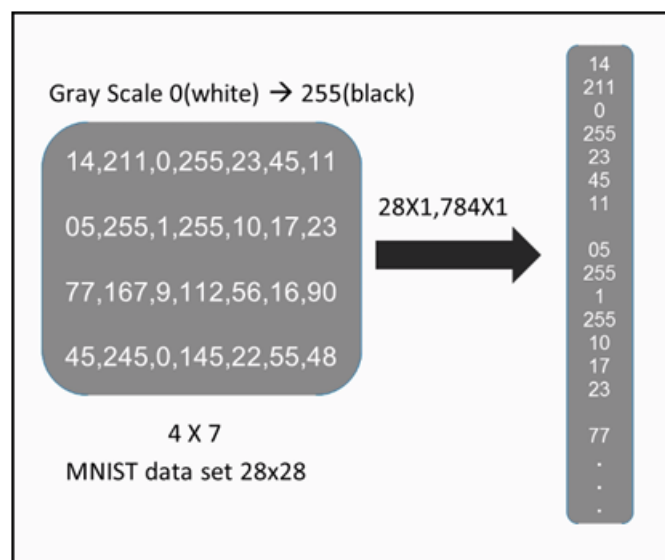


Abb. 19: Average-Pooling bei Grauwertbildern
(Ramo, 2019)

5.4.2 RGB-Bilder

Bei RGB-Farbbildern, beispielsweise einem JPEG, mit einer Größe von 260×194 Pixeln sieht der Computer diese als eine dreidimensionale Matrix von Zahlen. Genauer gesagt, wird die Matrix als $260 \times 194 \times 3$ wahrgenommen. Die dritte Dimension repräsentiert die drei Farbkanäle Rot, Grün und Blau. Das bedeutet, dass jede der drei Dimensionen die Intensität der roten, grünen und blauen Farbe darstellt (Ramo, 2019).

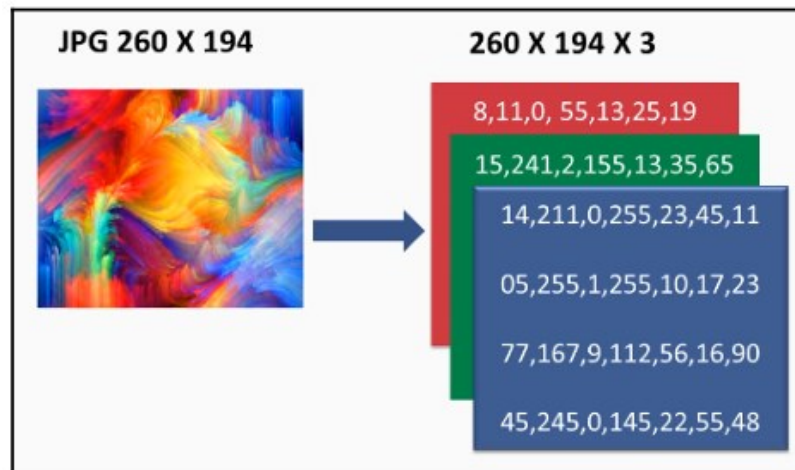


Abb. 20: Average-Pooling bei RGB-Bildern
(Ramo, 2019)

Als Beispiel: Bei der Farbe Rot bedeutet 0 Schwarz und 255 ein sattes Rot. Dasselbe gilt für Grün und Blau. Auch in diesem Fall muss die 3D-Matrix in einen eindimensionalen Vektor umgewandelt werden (Ramo, 2019).

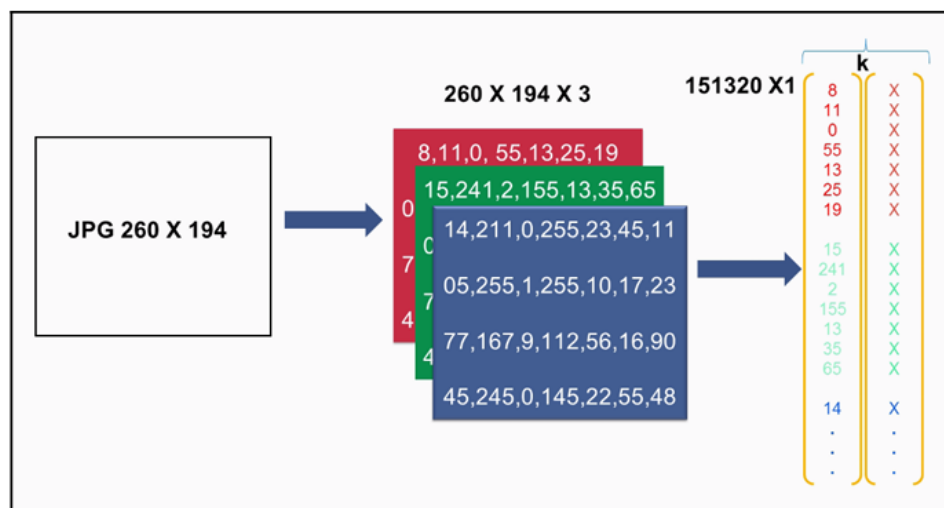


Abb. 21: Beispiel für Average-Pooling
(Ramo, 2019)

5.5 Architektur eines Convolutional Neural Networks

Convolutional Neural Networks (CNN) sind schichtenartig aufgebaut, wobei jede Schicht als räumliche Gitterstruktur angeordnet ist. Jede Schicht in einem CNN ist eine 3D-Gitterstruktur, die Höhe, Breite und Tiefe umfasst. Die Tiefe einer Schicht bezieht sich dabei auf die Anzahl der Kanäle in dieser Schicht, wie etwa die Anzahl der Farbkanäle (Rot, Grün, Blau) im Eingabebild oder die Anzahl der Feature Maps in den Zwischenschichten. Diese Tiefe sollte jedoch nicht mit der Tiefe des gesamten Netzwerks verwechselt werden (Aggarwal, 2018).

Räumliche Beziehungen werden von einer Schicht zur nächsten übertragen, wobei jeder Merkmalswert (engl.: Feature Value) auf einer kleinen, lokalen Region der vorangegangenen Schicht basiert. Es ist wichtig, dass diese räumlichen Beziehungen zwischen den Rasterzellen beibehalten werden, da die sogenannte „Convolution operation“ (dt.: Faltung) und die Transformation in die nächste Schicht von diesen Beziehungen abhängen (Aggarwal, 2018). CNN funktionieren ähnlich wie traditionelle

Feedforward-Netzwerke. Der Unterschied besteht jedoch darin, dass die Operationen in den Schichten räumlich organisiert sind und die Verbindungen zwischen den Schichten spärlich sind (Aggarwal, 2018). Convolutional Neural Networks nutzen räumlich-lokale Korrelationen, indem sie ein lokales Verbindungsmuster zwischen den Neuronen benachbarter Schichten bilden (Bhardwaj et al., 2018). Ein CNN setzt sich meist aus mehreren Schichten zusammen. Im Folgenden werden die Bestandteile primär unter Verwendung der englischen Bezeichnungen erläutert, da sie in der Fachliteratur üblich sind:

- Input Layer
- Convolutional Layer
- Activation (dt.: Aktivierung)
- Pooling Layer
- Fully-Connected Layer

5.5.1 Input Layer

Der Input Layer (dt.: Eingabeschicht) ist der Startpunkt eines Convolutional Neural Networks. Hier werden die Eingabedaten, meist Bilder, als Matrix aus Pixeln dargestellt (Hassaballah & Awad, 2020; Poonkuntran et al., 2022).

5.5.2 Convolutional Layer

Der Convolutional Layer (dt.: Faltungsschicht) ist das Herzstück eines CNN und verantwortlich für die Extraktion von Merkmalen aus den Eingabedaten. Diese Schicht führt die „Convolution operation“ aus, bei der Eingabedaten mit Gewichten (Filter bzw. Kernel) multipliziert werden, um lokale Muster wie Kanten oder Formen zu erkennen. Die dabei generierten Werte bilden eine zweidimensionale Feature Map, welche die Positionen der erkannten Merkmale im Bild beschreibt. Jeder Filter erzeugt eine eigene Feature Map (Géron, 2022; LeCun et al., 2015; Poonkuntran et al., 2022).

Funktionsweise:

- **Input:** Bilder in der Dimension „Breite × Höhe × Tiefe“ (z. B. Graustufenbild $32 \times 32 \times 1$ oder Farbbild $512 \times 512 \times 3$) (Dadhich, 2018).
- **Filter** (auch **Kernel** genannt): Sind kleine Matrizen aus Gewichten (z. B. 2×2 Pixel), die während des Trainings über die Bilder geschoben werden (Kumar & Renuka, 2023; Planche & Andres, 2019). Die Gewichtungen im Filter werden dabei angepasst, um lokale Muster oder wichtige Merkmale wie Linien, Kanten oder Formen in den Bilddaten zu erkennen (Géron, 2022).
- **Receptive Field** (dt.: Rezeptives Feld): Bereich des Bildes, auf den der Filter angewendet wird. Dabei wird der Filter über das Bild verschoben und für jeden Bereich eine Multiplikation mit den Werten des Inputs durchgeführt. Die Ergebnisse werden aufsummiert, um einen Output-Pixel zu erzeugen. Dieser Prozess wird als Konvolution bzw. Faltung bezeichnet (Dadhich, 2018; DataScientest, 2023; Géron, 2022).
- **Stride** (dt.: Schrittgröße): Der Filter wird in bestimmten Abständen über das Bild geschoben.
- **Output:** Werte, die durch die Multiplikation des Filters mit der Eingabematrix entstehen, bilden die Feature Map (Poonkuntran et al., 2022).

Der Filter gleitet schrittweise (Stride) über das Bild und identifiziert dabei Merkmale, wodurch die sogenannte **Translationsinvarianz** gewährleistet wird. Das bedeutet, dass Merkmale unabhängig von

ihrer genauen Position im Bild erkannt werden können. Die wiederholte Anwendung des Filters auf die rezeptiven Felder des Bildes ermöglicht die Erkennung von Mustern im gesamten Bild. Bilddaten weisen diese besondere Eigenschaft auf, im Gegensatz zu anderen rasterstrukturierten Datenarten (Aggarwal, 2018; Poonkuntran et al., 2022).

Nach der Faltung wird eine Aktivierungsfunktion (meist ReLU) angewendet, um Nichtlinearität in das System einzuführen. Dies ermöglicht dem Netzwerk, komplexe Beziehungsstrukturen zu erlernen und relevante Merkmale hervorzuheben. Die Tiefe des Outputs wird durch die Anzahl der verwendeten Filter bestimmt (Hassaballah & Awad, 2020; Poonkuntran et al., 2022). Der Output, also die Zahlenwerte, die durch die Multiplikation des Filters mit der Eingabematrix entstehen, wird in einem CNN zweidimensional angeordnet und als **Feature Map** bezeichnet. Diese Zahlenwerte können als Aktivierungen der Neuronen in der entsprechenden Feature Map interpretiert werden. Die Feature Map gibt an, wo sich die erkannten Merkmale im Bild befinden. Jeder Filter erzeugt dabei eine eigene Feature Map. Mit zunehmender Tiefe des Netzwerks werden die Merkmale in den Feature Maps immer komplexer und abstrakter (DataScientest, 2023).

Parameter der Feature Map:

- Input
- Filter
- Stride
- Padding (mehr Informationen in Kapitel 5.5.3)

Nach jedem Batch während des Trainings wird mithilfe einer Verlustfunktion der Verlust berechnet, indem die Modellvorhersagen mit den tatsächlichen Ground Truth-Daten (Annotationen) abgeglichen werden. Insgesamt ermöglicht der Convolutional Layer die automatische hierarchische Extraktion von Merkmalen, was besonders für Aufgaben wie die Objekterkennung von zentraler Bedeutung ist (Hassaballah & Awad, 2020).

5.5.3 Pooling Layer

Der Pooling Layer, auch Subsampling Layer genannt, wird meist nach jedem Convolution Layer eingesetzt. Er reduziert die Datenmenge, verdichtet die Merkmale und verringert die Auflösung, wodurch Rechenleistung, Speicherbedarf und Anzahl der lernbaren Parameter gesenkt werden, ohne die Leistungsfähigkeit zu beeinträchtigen (Poonkuntran et al., 2022; Wang et al., 2019). Diese Schicht hilft zudem, Overfitting (dt.: Überanpassung) zu vermeiden, indem sie nur die wichtigsten Merkmale der Eingabedaten beibehält (Hassaballah & Awad, 2020). Parameter wie Filteranzahl, Filtergröße, Stride und Padding beeinflussen das Pooling. Der Stride gibt an, wie viele Pixel der Filter bei der Anwendung verschoben wird. Bei einem Stride von 1 verschiebt sich der Filter um ein Pixel (Dadhich, 2018). Eine größere Schrittweite führt zu spärlicheren Feature Maps (Planche & Andres, 2019). Padding (dt.: Auffüllung) füllt den Rand des Bildes mit Nullwerten oder anderen Werten auf, sodass der Filter den gesamten Bildbereich abdecken kann (Aggarwal, 2018; Dadhich, 2018).

Die Pooling-Operation reduziert die räumliche Dimension der Feature Maps und kann mit Methoden wie Max-Pooling oder Average-Pooling durchgeführt werden. Max-Pooling, bei dem der Maximalwert eines rezeptiven Feldes berechnet wird, ist weit verbreiteter und erzielt bessere Ergebnisse. Bei einer

Pooling-Operation von 2×2 Pixel mit Stride 2 und ohne Padding wird die Feature Map um die Hälfte reduziert (Poonkuntran et al., 2022; Ramo, 2019). Pooling trägt zur Verbesserung der Berechnungsgeschwindigkeit, Effizienz und Genauigkeit des Netzwerks bei und hilft, Overfitting zu reduzieren (Ramo, 2019).

Pooling-Methoden:

- **Average Pooling:** Für jedes rezeptive Feld wird der Durchschnittswert der Feature Maps berechnet. Die Average-Pooling Funktion ist in Abbildung 22 dargestellt.
- **Max-Pooling:** Max-Pooling ist verbreiteter, da sie in der Regel bessere Ergebnisse liefert. Hier wird für jedes rezeptive Feld im Output der Maximalwert der Feature Maps berechnet. Diese Funktion ist in Abbildung 23 dargestellt (Poonkuntran et al., 2022; Ramo, 2019).

In den unteren beiden Beispielen beträgt die Filtergröße 2×2 . Es wird eine Schrittweite von 2 angewendet, und es wird kein Padding verwendet (Ramo, 2019).

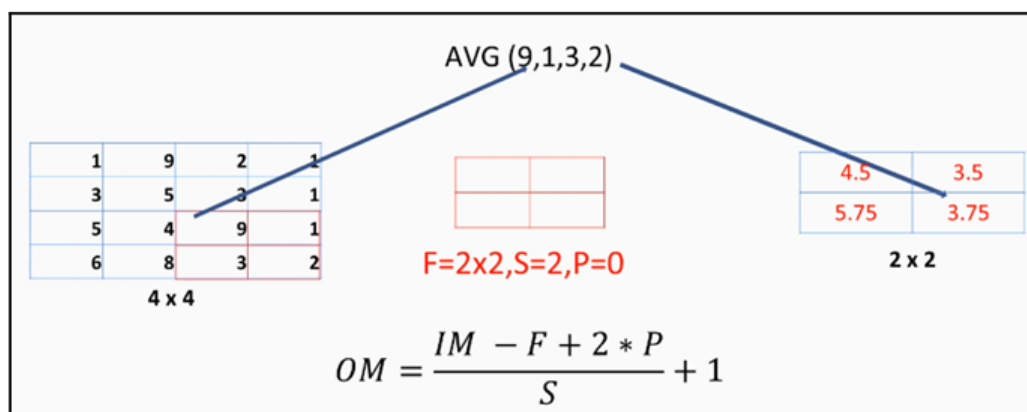


Abb. 22: Average-Pooling
(Ramo, 2019)

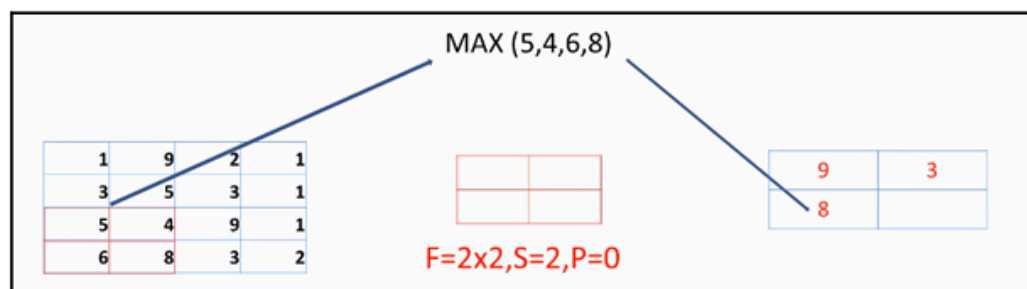


Abb. 23: MAX-Pooling
(Ramo, 2019)

5.5.4 Fully-Connected Layer

Der Fully-Connected Layer dient in einem Convolutional Neural Network zur finalen Klassifikation des Bildinhalts. In der Objekterkennung und Segmentierung wird er heute seltener verwendet, da effektivere Alternativen existieren (Géron, 2022; Planche & Andres, 2019). Bei Bildsegmentierungsaufgaben ersetzt oft ein weiterer Convolutional Layer den Fully-Connected Layer, oder spezialisierte Architekturen wie Fully Convolutional Networks (FCN) kommen zum Einsatz. In der Objekterkennung wird stattdessen häufig ein Global Average Pooling (GAP) Layer genutzt (Planche & Andres, 2019). Der Fully-Connected Layer folgt den Convolutional- und Pooling-Layern, verarbeitet die extrahierten

Merkmale und verbindet sich mit den vorherigen Feature Maps (Pu et al., 2019). Dazu werden die Feature Maps geglättet (engl.: Flattening), summiert und mit lernbaren Gewichtungen multipliziert. Dadurch entstehen dimensionslose, formlose Neuronen, die nur einzelne Werte repräsentieren (Poonkuntran et al., 2022; Sampath et al., 2021).

CNN enthalten oft mehrere Fully-Connected Layer, die für Klassifikations- oder Regressionsaufgaben genutzt werden. Der letzte Fully-Connected Layer besitzt dabei so viele Neuronen wie Klassenzugehörigkeiten. Das Neuron mit dem höchsten Wert gibt die vorhergesagte Klasse an. Enthält beispielsweise das dritte Neuron den höchsten Wert, wird das Bild der dritten Klasse zugewiesen (Pu et al., 2019).

5.6 Der Lernprozess eines Künstlichen Neuronalen Netzes (KNN)

Nach der Erstellung eines Trainings- und Testmodells kann das Netzwerk mit geeigneten Hyperparametern optimiert werden (Poonkuntran et al., 2022). Diese Parameter müssen manuell konfiguriert und iterativ angepasst werden, um bessere Ergebnisse zu erzielen. Dies geschieht während des Trainings. Während des Trainingsprozesses lernt das Modell anhand bekannter Daten, neue Eingaben zu klassifizieren. Beim Training steht ein Satz von Trainingsdaten zur Verfügung, und die Beziehungen zwischen den Trainingseingaben und den gewünschten, erwarteten Outputs (oder Musterklassen) müssen in das neuronale Netz eingespeist werden. Das Netz wird anhand dieser bekannten Daten trainiert und lernt, neue Daten zu erkennen und zu klassifizieren (Vasuki & Govindaraju, 2017).

5.6.1 Forward Propagation, Backpropagation und Verlustfunktion

Der Trainingsprozess lässt sich in drei Phasen gliedern:

- **Forward Propagation** (dt.: Vorwärtsschritt): Die Inputdaten werden durch das Netzwerk bis zum Output-Layer weitergeleitet. Dabei werden die Aktivierungen und der endgültige Output berechnet. In der Forward Propagation werden also Signale durch das gesamte neuronale Netz durchgeschleust (Grunau, 2022; Kobold AI, 2024; Wlodarczak, 2019).
- **Backpropagation** (dt.: Rückwärtsschritt): Die Modellparameter, insbesondere die Gewichte des Netzwerks, werden durch Berechnung der Fehlergradienten (engl.: Gradient Descent) schrittweise angepasst, um die Differenz zwischen den vorhergesagten und den tatsächlichen Werten zu minimieren. Nach jeder Iteration wird der Verlust (engl.: Loss) berechnet und zurück durch das Netzwerk propagiert. Dabei werden die Gewichte so aktualisiert, dass der Fehler in der nächsten Iteration verringert wird (Kalita, 2022; Planche & Andres, 2019; Wlodarczak, 2019). Durch diesen Prozess werden die optimalen Gewichtungen ermittelt, indem die Modellvorhersagen mit den erwarteten Werten verglichen und die Abweichungen minimiert werden (Sewak et al., 2018).
- **Loss Function** (dt.: Verlustfunktion): Sie misst die Abweichung der Vorhersage von der erwarteten Zielvariable, indem die Modellparameter iterativ angepasst werden, und dient als Grundlage für die Optimierung des Netzwerks bzw. Modells (La La, 2019; Ng & Soo, 2018).

Während des Trainings werden die Gewichte und Bias-Werte zunächst zufällig gesetzt und anschließend durch Backpropagation schrittweise optimiert. Ziel ist es, durch Optimierungen der

Gewichtungen die Fehlerquote zu minimieren und die Modellgenauigkeit zu erhöhen (Bhattacharyya et al., 2020; Kobold AI, 2024).

5.6.2 Aktivierungsfunktionen

Neuronale Netzwerke müssen komplexe Muster erkennen und verarbeiten. Um Nichtlinearität in Convolutional Layer einzuführen, wird nach jedem dieser Layer eine Aktivierungsfunktion angewendet (Kumar & Renuka, 2023). Mit zunehmender Tiefe eines Modells können Probleme wie der verschwindende Gradient auftreten. Häufig wird die Rectified Linear Unit (ReLU) als Aktivierungsfunktion genutzt, aber auch Sigmoid und Tanh sind gebräuchlich (Wlodarczak, 2019). Die Aktivierungsfunktion entscheidet in jedem Neuron, ob ein eingehendes Signal weitergeleitet wird. Sie ist essenziell für das Training, da sie den Fehlerausgleich während der Backpropagation ermöglicht. Insbesondere durch das Gradientenverfahren werden die Gewichte optimiert und Fehler reduziert (Bhardwaj et al., 2018).

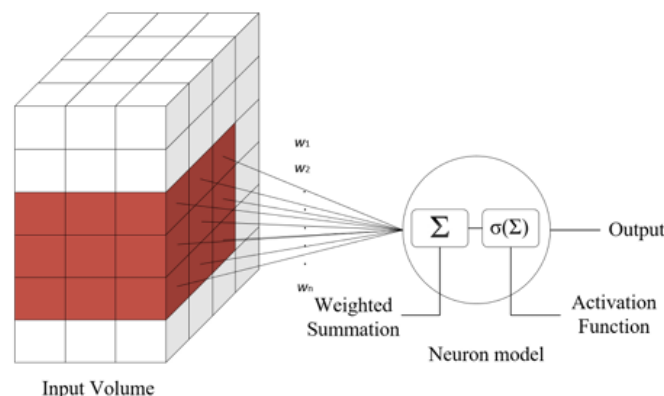


Abb. 24: Darstellung einer Aktivierungsfunktion
(Hassaballah & Awad, 2020)

5.6.3 Batch, Iteration und Epoche

Da neuronale Netzwerke oft mit großen Datenmengen trainiert werden, erfolgt das Training in **Batches**. Der Datensatz wird dabei in gleich große Teilmengen unterteilt, die jeweils einen Batch darstellen. Eine **Iteration** ist abgeschlossen, wenn die Parameter mit einem Batch aktualisiert wurden (Grunau, 2022). Eine **Epoche** bezeichnet einen vollständigen Durchlauf des gesamten Trainingsdatensatzes, wobei Forward und Backward Propagation mehrfach durchlaufen werden, um den Verlust zu minimieren. Dieser Prozess erfordert eine hohe Rechenleistung (Hassaballah & Awad, 2020; Kalita, 2022).

5.6.4 Gradientenverfahren

Die Verlustfunktion eines neuronalen Netzwerks kann als Kurve mit einem Gradienten dargestellt werden. Ihr Ziel ist es, das Minimum dieser Funktion zu finden, da dieses den geringsten Fehler repräsentiert. Dies geschieht durch schrittweise Anpassung der Gewichte, wobei der Gradient angibt, in welche Richtung sich das Modell bewegen muss, um das Minimum zu erreichen (Bhattacharyya et al., 2020; La La, 2019).

Ist die Steigung negativ, werden Werte zu den Gewichten addiert, ist sie positiv, werden Werte subtrahiert. Die Schrittgröße wird durch die **Lernrate** (engl.: Learning Rate) bestimmt. Ist die Lernrate zu hoch gewählt, kann das Minimum übersprungen werden, während eine zu niedrige das Training

verlangsamt (La La, 2019; Planche & Andres, 2019). Abbildung 25 zeigt ein Diagramm einer Verlustfunktion mit einer einfachen Kurve. In der Praxis ist die Verlustfunktion komplex und besitzt viele lokale Minima. Es muss das globale Minimum gefunden werden, der niedrigste Tiefpunkt. Da nahe gelegene lokale Minima irreführend sein können, beginnt das Training oft an einem zufälligen Punkt entlang der Lernkurve und nutzt Methoden wie das stochastische Gradientenverfahren, um die Optimierung effizienter zu gestalten. (La La, 2019; LeCun et al., 2015).

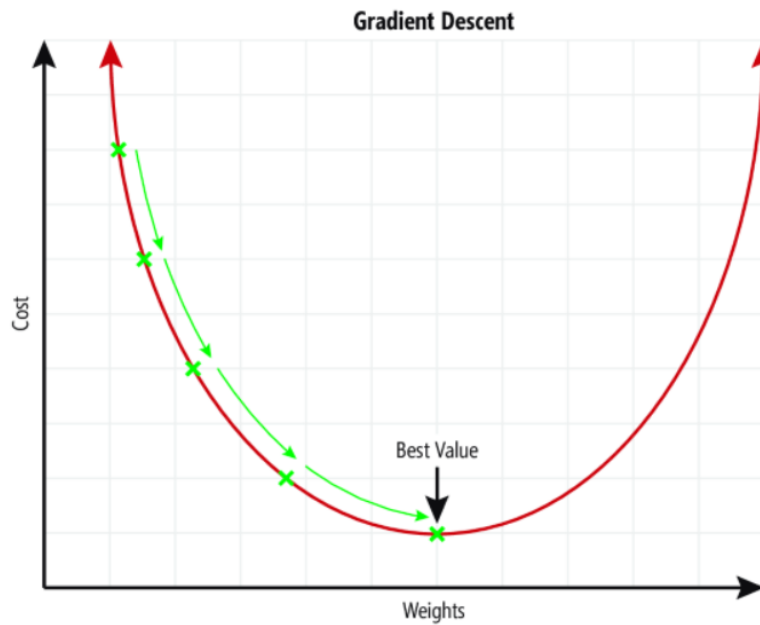


Abb. 25: Verlustfunktion mit Kurve
(La La, 2019)

Folgende Parameter beeinflussen das Training eines neuronalen Netzwerks:

- Netzwerkarchitektur bzw. Anzahl der Schichten (besonders die Zwischenschichten)
- Aktivierungsfunktion
- Verlustfunktion
- Filtergröße und -anzahl
- Stride und Padding
- Lernrate
- Batchgröße
- Anzahl der Epochen

5.7 Zu beachtende Herausforderungen

Beim Training von CNN müssen bestimmte Prinzipien beachtet werden, um ein effizientes und fehlerfreies Training sicherzustellen. Die **Anzahl an Trainingsdaten**, insbesondere repräsentativer Natur, ist von entscheidender Bedeutung. Zudem ermöglichen **Validierungsdaten** die frühzeitige Erkennung potenzieller Probleme und helfen, die Generalisierungsfähigkeit des Modells zu bewerten (Planche & Andres, 2019). Dazu werden vor dem Training 20% der Daten zurückgehalten. Eine Aufteilung in 80% Trainings- und 20% Validierungsdaten stellt eine bewährte Balance dar. Sie gewährleistet, dass die Validierungsdaten nicht in den Trainingsprozess einfließen, wodurch eine

objektivere Bewertung der Modellleistung möglich wird (Géron, 2022). Eine ausreichende und repräsentative Datenmenge ist entscheidend. Die Größe der Eingabe- und Ausgabeschichten variiert je nach Anwendung. Bei einer Klassifizierung entspricht die Eingabegröße der Anzahl der Pixelwerte und die Ausgabegröße der Anzahl der vorhergesagten Klassen. Die Zwischenschichten erfordern eine sorgfältige Anpassung der **Hyperparameter**, was zahlreiche Tests erfordert (Planche & Andres, 2019).

Ein zu einfaches Netzwerk mit wenigen oder kleinen Schichten kann zu Underfitting (dt.: Unteranpassung) führen, da die Modellkapazität nicht ausreicht, um die zugrundeliegenden Muster in den Daten zu erfassen, was zu einer schlechten Genauigkeit führt. Ein zu komplexes Modell mit wenig Trainingsdaten birgt hingegen das Risiko von Overfitting (dt.: Überanpassung). (ebd.)

5.7.1 Overfitting

Overfitting (dt.: Überanpassung) tritt auf, wenn ein Modell zu stark an die Trainingsdaten angepasst wird und somit schlecht auf neue Daten generalisiert. Besonders tiefe neuronale Netzwerke mit vielen Parametern sind anfällig dafür (Bhardwaj et al., 2018). Overfitting beschreibt die Tendenz von Modellen, sich zu sehr an die Trainingsdaten anzupassen, einschließlich spezifischem Rauschen und Details, was zu einer schlechten Generalisierung auf neue, nicht annotierte Testdaten führt. In diesem Fall wird eine große Lücke zwischen der Trainings- und Validierungsgenauigkeit sichtbar, die bei komplexen Modellen und kleinen Datensätzen besonders ausgeprägt ist (Aggarwal, 2018; Planche & Andres, 2019).

Dropout ist eine bewährte Methode zur Vermeidung von Overfitting. Dabei werden zufällig Neuronen deaktiviert, sodass das Modell weniger abhängig von einzelnen Neuronen wird und robustere Merkmalsrepräsentationen (engl.: Feature Representation) lernt. Dies fördert eine bessere Generalisierung des Modells (Bhardwaj et al., 2018).

5.7.2 Underfitting

Underfitting (dt.: Unteranpassung) tritt auf, wenn ein Modell zu simpel ist, um die zugrunde liegenden Muster in den Daten zu erfassen. Dadurch kann das Modell sowohl auf Trainings- als auch auf Validierungsdaten eine unzureichende Leistung zeigen (GeeksforGeeks, 2024). Zur Lösung kann eine komplexere Architektur mit verbesserter Merkmalsrepräsentation gewählt werden. Ein hoher Validierungsverlust im Vergleich zum Trainingsverlust weist auf Underfitting hin (Brownlee, 2019).

5.7.3 Verschwindende und explodierende Gradienten

Tiefere Netzwerke sind anfällig für zwei Hauptprobleme: verschwindende und explodierende Gradienten. Diese treten während der Backpropagation auf, wenn die Gradienten entweder zu klein oder zu groß werden (Bhardwaj et al., 2018; Ramo, 2019).

- **Verschwindende Gradienten** (engl.: Vanishing Gradient): Der verschwindende Gradient, bereits in Kapitel 5.2 erwähnt, stellt eine häufige Herausforderung für Convolutional Neural Networks dar. Besonders in tiefen Netzen mit vielen Schichten können kleine Gradientenwerte das Lernen verlangsamen oder sogar vollständig stoppen. Dies tritt häufig auf, wenn Aktivierungsfunktionen mit Ableitungen kleiner als 1, wie etwa die Sigmoid-Funktion, verwendet werden. In solchen Fällen multiplizieren sich die Gradienten bei der Backpropagation durch jede Schicht und schrumpfen mit zunehmender Tiefe des Netzes immer weiter gegen Null (Bhardwaj et al., 2018; Planche & Andres, 2019).

- **Explodierende Gradienten** (engl.: Exploding Gradient): Zu große Gradientenwerte führen zu instabilen Gewichtsaktualisierungen oder sogar NaN-Werten. Dies kann auftreten, wenn Aktivierungsfunktionen unbeschränkt sind oder die Lernrate zu hoch ist (Bhardwaj et al., 2018; Planche & Andres, 2019). Ein bewährtes Mittel gegen explodierende Gradienten ist Gradient Clipping, das extreme Gradientenwerte auf einen Maximalwert begrenzt (Bhattacharyya et al., 2020).

5.8 Forschung und verwandte Arbeiten

Die Einsatzbereiche reichen von der einfachen Erkennung und Klassifizierung von Photovoltaikanlagen bis hin zur Identifizierung von Anomalien durch den Einsatz von Infrarotbildern. Weitere Aspekte, wie die Berücksichtigung von Montagemöglichkeiten (z. B. Sonnentracker oder fest installierte Anlagen) und die Art der Anlagen (auf Dächern oder Freiflächen), werden ebenfalls in den Klassifizierungsprozess einbezogen.

Vasku (2019) klassifizierte großräumige PV-Anlagen in Solarparks mithilfe von Sentinel-1- und Sentinel-2-Satellitenbildern mit einer geometrischen Auflösung von mindestens 10m. Dabei kam die cloudbasierte Plattform Google Earth Engine (GEE) zum Einsatz. Für die Objekterkennung wurden keine neuronalen Netze verwendet, sondern traditionelle Architekturen wie SVM (Support Vector Machines), Naive Bayes, CART (Classification and Regression Trees) und Random Forest. Die Ergebnisse wurden mit Bewertungsmethoden wie dem Kappa-Koeffizienten überprüft. Random Forest erzielte dabei die besten Ergebnisse, was auch visuell bestätigt wurde. Im Vergleich zu CART wies Random Forest eine geringere Anzahl falsch positiver Klassifikationen auf.

Ge et al. (2022) entwickelten eine hierarchische Methode zur Erkennung und Klassifizierung von großen Photovoltaikanlagen. Diese Methode besteht aus zwei Teilen: Zunächst wird das Deep-Learning-Objekterkennungsmodell EfficientNet-B5 verwendet, eine erweiterte Variante der EfficientNet-Modellfamilie, um PV-Anlagen auf Satellitenbildern mit einer Auflösung von 16m zu lokalisieren. Anschließend wird das semantische Segmentierungsmodell U²-Net eingesetzt, um die Anlagen mit einer präziseren räumlichen Auflösung von 2m zu extrahieren. Die entwickelte Methode wies in einem Vergleich mit vier anderen klassischen Modellen die höchste Genauigkeit auf.

Perry und Campos (2023) verfolgten einen ähnlichen Ansatz zur Erkennung und Klassifizierung von PV-Anlagen auf Satellitenbildern, wobei auch die Montage und Installation berücksichtigt wurden. Sie nutzten das Modell Faster R-CNN mit ResNet-50 als Backbone. Es wurden verschiedene Typen von Anlagen unterschieden, darunter fest installierte Aufdachanlagen, Parkdeck-Anlagen, einachsige Freiflächenanlagen mit Sonnentrackern und fest installierte Freiflächenanlagen. Die Satellitenbilder hatten eine Auflösung von 0,6m. Zur Beurteilung der Modellleistung wurden die Metriken AP (Average Precision) und mAP (Mean Average Precision) verwendet. Das Modell erzielte die besten Ergebnisse bei der Erkennung von fest installierten Parkdeck- und Freiflächenanlagen, hatte jedoch Schwierigkeiten bei der Erkennung von Aufdachanlagen. Durch die zusätzliche Berücksichtigung der Montagekonfiguration wurde die Aufgabe komplexer und das Modell fehleranfälliger. Wiederholt wurden Schiffscontainer und Dachfenster fälschlicherweise als PV-Anlagen klassifiziert.

Petrik und Hamme (2020) konzentrierten sich auf die Lokalisierung von Solaranlagen auf Gebäudedächern mithilfe von Orthofotos und OSM-Kartendaten. Sie betrachteten verschiedene

Algorithmen, darunter einen selbst mit der **PyTorch**-Bibliothek entwickelten. Der Algorithmus basierte auf CNN und Feature-Vektoren, die von Malof et al. im Jahr 2015 eingeführt wurden. Die Klassifizierung erfolgte auf Basis der Bodenauflösung der verwendeten Orthofotos. Die Autoren verwendeten zum Training sowohl einen eigenen Datensatz (mit einer Auflösung von 0,1m) als auch einen bestehenden Datensatz (mit einer Auflösung von 0,3m). Der eigene Datensatz erzielte eine Genauigkeit von 21% bei einer Trefferquote von 45,7%, während der fertige Datensatz eine Genauigkeit von 72,2% bei einer Trefferquote von 68,9% ermöglichte. Verbesserungsideen beinhalteten das Schwärzen der Dächer, was die Erkennungsgenauigkeit steigerte, sowie das Trainieren des neuronalen Netzes mit mehr Epochen. Unterschiede in der Auflösung der Datensätze führten jedoch zu ungenauen Ergebnissen bei der Klassifizierung der einzelnen Pixel.

Díaz et al. (2020) nutzten UAV-Thermalbilder zur Identifizierung von Anomalien auf PV-Anlagen, um ein gezieltes Monitoring zur Verlängerung ihrer Lebensdauer zu ermöglichen. Die Bilder wurden mit einer DJI Zenmuse XT-Kamera aufgenommen. Der Datensatz bestand aus elf Flügen mit einer Flughöhe zwischen 34m und 56m. Zur Erkennung wurden zwei Verfahren eingesetzt: Die erste Methode beinhaltete Bildkorrektur, Segmentierung und Klassifizierung der Segmente mithilfe von SVM. Die zweite Methode basierte auf einem Postprocessing-Schritt und Deep Learning, genauer gesagt auf Mask R-CNN. Diese Methode reduzierte falsch-positive Ergebnisse um mehr als 60% im Vergleich zur ersten Methode und ermöglichte eine genauere Erkennung von PV-Anlagen, insbesondere bei komplexen Hintergründen.

Es existieren sicherlich noch weitere Arbeiten zur Lokalisierung von PV-Anlagen basierend auf Satellitenbildern oder Orthofotos. Eine weiterführende Recherche und eine detaillierte Zusammenfassung würden jedoch den Rahmen dieser Arbeit sprengen.

5.8.1 Ableitende Erkenntnisse aus den vorgestellten Forschungen

Zusammenfassend lässt sich feststellen, dass die Identifikation von Photovoltaikanlagen anhand von Orthofotos bereits in mehreren Studien untersucht wurde, wobei unterschiedliche Ansätze und Algorithmen zum Einsatz kamen. In den betrachteten Publikationen basieren die Modelle, die neuronale Netze nutzen, auf CNNs. Nicht eingeschlossen sind klassische Machine-Learning-Modelle, wie sie beispielsweise Vasku (2019) verwendet. Diese werden auch als „flache Modelle“ bezeichnet (Jiang et al., 2019). Die eingesetzten Modelle sind meist vortrainierte Netzwerke, die mithilfe von Deep Learning- oder Machine Learning-Bibliotheken für Python aufgerufen und an den eigenen Datensatz angepasst werden können. Eine Ausnahme bilden Petrik und Hamme, die einen eigenen Algorithmus entwickelt haben.

Die folgende Tabelle fasst die fünf ausgewählten Kriterien (X-Achse) und die untersuchten Studien (Y-Achse) zusammen.

Studie	Modellart & Modell	Datengrundlage & Auflösung	Samplegröße	Klassen	Metriken
Díaz et al. (2020)	Flache ML-Modelle, Neuronales Netz: SVM, Mask R-CNN	Orthomosaik 34 – 56m	18 244 Annotationen	2 Klassen: PVA und „Nicht“-PVA	Recall, Precision, F1
Ge et al. (2022)	Neuronales Netz: EfficientNet-B5, U ² -Net	Satellitenbilder 16m	22 025 Annotationen	2 Klassen: PVA und „Nicht“-PVA	Accuracy, Recall, Precision, F1, IoU
Perry & Campos (2023)	Neuronales Netz: Faster R-CNN	Satellitenbilder 0,6m	862 Bilder	4 Klassen: Versch. PVA-Installationsarten	AP, mAP
Petrik & Hamme (2020)	Neuronales Netz: Feature-Vektoren	Orthofotos 0,1 – 0,3m	341 379 Bilder/Ann.	1 Klasse: PVA	Genauigkeit, Trefferquote
Vasku (2019)	Flache ML-Modelle: Random Forest, SVM, CART, Naive Bayes	Satellitenbilder 10 – 60m	89 Annotationen	2 Klassen: PVA und „Nicht“-PVA	Overall Accuracy, Producer's/Consumer's Accuracy, Kappa

Tab. 1: Vergleich verwandter Forschungen

Die verwendeten Datenquellen unterscheiden sich erheblich. Während in vielen Studien aufgrund der leichten Verfügbarkeit auf Satellitendaten zurückgegriffen wird, finden auch Orthofotos, die aus Flugzeug- oder Drohnenaufnahmen stammen, zunehmend Verwendung. Letztere sind jedoch in manchen Regionen kostenpflichtig. Die Bodenauflösung variiert stark und reicht von 0,1m bis 60m. Eine höhere Auflösung (niedrigerer Wert) kann sich positiv auf die Erkennungs- und Klassifizierungsaufgaben auswirken. Die Datenbasis dieser Arbeit bildet ein Orthomosaik.

Die Größe des Datensatzes spielt eine entscheidende Rolle für die Modellleistung. Eine höhere Anzahl an Annotationen ermöglicht dem Modell, bessere Zusammenhänge zwischen Eingabedaten und Vorhersagen zu erlernen. In den betrachteten Studien wurden meist entweder nur PV-Anlagen annotiert (eine Klasse) oder zwischen PV-Anlagen und Nicht-PV-Anlagen unterschieden (zwei Klassen). Einzige Ausnahme bildet Perry und Campos (2023), die vier verschiedene Installationsarten von PV-Anlagen klassifizierten – jedoch keine weiteren Objekte, die aufgrund ähnlicher spektraler Eigenschaften zu Verwechslungen führen könnten. Keines der Werke thematisiert explizit die visuelle Ähnlichkeit zwischen Photovoltaikanlagen und Solarthermieranlagen. Bereits Petrik und Hamme (2020) stellten fest, dass diese Problematik kaum berücksichtigt wird, führten jedoch selbst keine entsprechende Untersuchung durch. Daher wird in dieser Arbeit eine feinere Klassifikation angestrebt, um Photovoltaikanlagen von Solarthermieranlagen zu unterscheiden, da beide ähnliche spektrale Eigenschaften aufweisen.

Die Evaluierung bzw. die Modellbewertung erfolgt anhand spezifischer Metriken, die je nach Architektur (flaches oder tiefes Netzwerk) und Klassifizierungsmethode (objekt- oder pixelbasiert) variieren können. Wie die Tabelle zeigt, werden in den betrachteten Studien mindestens zwei verschiedene Bewertungskriterien verwendet – ein essenzieller Aspekt, da jede Metrik unterschiedliche Facetten der Modellleistung erfasst. Die ausschließliche Nutzung einer einzelnen Metrik könnte zu irreführenden Schlussfolgerungen führen.

Schlussfolgernd liegt der Schwerpunkt dieser Masterarbeit auf folgenden Aspekten:

- **Modellwahl:** Für die praktische Umsetzung werden vortrainierte Modelle auf Basis von Convolutional Neural Networks (CNNs) verwendet. Die in Betracht kommenden Modelle werden in Kapitel 6.2 vorgestellt. Voraussetzung ist, dass sie von der Deep Learning-Bibliothek `arcgis.learn` unterstützt werden (siehe Kapitel 6.1).
- **Datengrundlage:** Als Grundlage dient ein Orthomosaik.
- **Samplegröße:** Die Trainingsdaten umfassen etwa 10 000 Bilder inklusive Annotationen. Die Größe des Untersuchungsgebiets ist zweitrangig, da die Anzahl an PV-Anlagen im Vordergrund steht, die je nach Gebiet unterschiedlich ausfallen kann.
- **Klassifikation:** Die Annotationen werden in zwei Klassen unterteilt (Photovoltaikanlagen und Solarthermieranlagen).
- **Metriken:** Die Auswahl geeigneter Bewertungsmethoden wird in den folgenden Kapiteln diskutiert.

Im nächsten Kapitel werden die Modelle vorgestellt, die in die engere Auswahl gelangt sind. Zwei davon werden schließlich für die praktische Umsetzung verwendet.

6 Praktische Herangehensweise

In diesem Kapitel wird die Methodik der Masterarbeit dargelegt. Ziel der Arbeit ist es, Photovoltaikanlagen (PVA) auf Orthofotos zu identifizieren und zu klassifizieren, indem zwei Convolutional Neural Networks (CNN) eingesetzt werden. Ein Vergleich beider Modelle soll zeigen, welches die beste Performance erzielt. Dabei werden sowohl objektbasierte als auch pixelbasierte Ansätze berücksichtigt. Die CNN werden zunächst auf bisher unbekannten Daten trainiert. Vor der Inferenz – also der Anwendung des trainierten Modells zur Erkennung und Klassifikation von Objekten – müssen die Daten entsprechend aufbereitet werden. Dieser Schritt wird auch als Datenvorverarbeitung (engl.: Preprocessing) bezeichnet.

6.1 Begründung der Softwarewahl

Für die Umsetzung wird ArcGIS verwendet, insbesondere in der Kombination aus ArcGIS Pro, dem Desktop-GIS, `ArcPy`, der Python-Bibliothek von ArcGIS sowie der ArcGIS API for Python mit ihrem Deep Learning-Modul `arcgis.learn`. Diese Wahl basiert auf mehreren Faktoren:

1. **Vielseitigkeit und Integrationsmöglichkeiten:** Die ArcGIS API for Python bietet eine breite Palette an GIS-Funktionen, darunter für Visualisierung, räumliche Analysen, Datenmanagement und die Administration von WebGIS-Systemen. Dies ermöglicht eine nahtlose Integration verschiedener Tools innerhalb einer Plattform. Sie ist u. a. für die Arbeit mit WebGIS (ArcGIS Enterprise und ArcGIS Online) konzipiert, bietet jedoch auch leistungsfähige Funktionen für lokale Datenanalysen. Die API kann sowohl in interaktiven Umgebungen wie *Jupyter Notebook*, in klassischen Entwicklungsumgebungen (IDEs) als auch direkt in ArcGIS Pro für die Automatisierung von GIS-Prozessen eingesetzt werden (Esri Developer, 2024). Im Vergleich dazu ist `ArcPy` stärker auf die Automatisierung von GIS-Tools und Funktionen innerhalb von ArcGIS Pro ausgerichtet. Die Verknüpfung zwischen ArcGIS Pro, `ArcPy` und der ArcGIS API ermöglicht eine nahtlose Arbeitsumgebung.
2. **Unterstützung für Deep Learning:** Mit der Bibliothek `arcgis.learn` wird die Anwendung von Deep Learning-Methoden direkt innerhalb der GIS-Umgebung ermöglicht. Dies reduziert den Aufwand für den Datenaustausch zwischen verschiedenen Softwaretools erheblich und erlaubt ein durchgängiges Arbeiten.
3. **Erweiterbarkeit durch externe Python-Bibliotheken:** Ein großer Vorteil ist, dass die ArcGIS API for Python es ermöglicht, externe Python-Bibliotheken wie `TensorFlow`, `PyTorch` und `Scikit-Learn` zu integrieren. Dadurch kann die Funktionalität erweitert werden, falls die nativen Möglichkeiten von `arcgis.learn` nicht ausreichen. Sie erfordern jedoch eine separate GIS-Datenvorverarbeitung und -visualisierung.
4. **Vertrautheit und Effizienz:** Ein weiteres Kriterium ist eine umfassende Erfahrung mit der ArcGIS-Plattform. Sie ermöglicht eine effiziente Umsetzung dieser Arbeit, da keine zusätzliche Einarbeitungszeit in neue Software erforderlich ist.

Neben ArcGIS gibt es auch andere Softwarelösungen, die ebenfalls für die Umsetzung geeignet wären. Zum einen wäre QGIS eine leistungsstarke Open Source-Alternative mit einer aktiven Community. In Kombination mit Plugins wie „Semi-Automatic Classification Plugin“ oder „Deep Learning Frameworks“ könnten ähnliche Aufgaben bearbeitet werden. Eine weitere GIS-Software auf Open Source-Basis ist GRASS GIS, welche über Funktionen für Geodatenanalyse verfügt. Zudem bietet Google Earth Engine

(GEE) die Möglichkeit zur Verarbeitung großer räumlicher Datensätze, den Zugriff auf viele geodatenbezogene Ressourcen sowie die Nutzung leistungsstarker Cloud Computing-Kapazitäten. Allerdings ist GEE für Nutzung integrierter Algorithmen konzipiert, und weniger für benutzerdefinierte Deep Learning-Modelle und deren individuelle Anpassungen.

Ein typischer Deep Learning-Workflow umfasst folgende Schritte:

- Annotation von Trainingsdaten
- Datenvorverarbeitung und -aufbereitung für das Training
- Modellwahl, Training des Modells und Hyperparameter-Tuning
- Inferenz, d. h. das Erkennen von Objekten und Klassifizieren von Pixeln
- Modellverbesserung und -validierung
- Abschlussklassifikation

ArcGIS Pro installiert die ArcGIS API for Python standardmäßig mit. Dadurch entfällt eine separate Installation. Auf die Bibliothek und deren Werkzeuge kann direkt zugegriffen werden, indem Jupyter Notebook aus dem Installationsordner von ArcGIS Pro gestartet wird. In einem Verzeichnis der Wahl können Notebooks kreiert oder bestehende bearbeitet werden. Alternativ bietet ArcGIS Pro die Möglichkeit, Jupyter Notebook direkt in die Benutzeroberfläche zu integrieren, über das Modul *ArcGIS Notebook*. In dieser Arbeit wird diese integrierte Variante verwendet (Esri, 2023h).

6.2 Datengrundlage

Für die Durchführung der Modelltrainings und -tests wird eine geeignete Datengrundlage benötigt, die aktuelle, hochaufgelöste und georeferenzierte Informationen über das Untersuchungsgebiet bereitstellt. Das Untersuchungsgebiet und die verwendeten Orthofotos, die als Basis für die Erstellung der Trainings- und Validierungsdatensätze dienen, werden in diesem Kapitel näher beschrieben.

6.2.1 Untersuchungsgebiet

Das Untersuchungsgebiet muss auf einer aktuellen Datengrundlage basieren. Für den Trainingsprozess ist es außerdem essenziell, dass das Gebiet keine homogene Struktur aufweist. Das bedeutet, dass die Landschaft auf dem Gebiet eine abwechslungsreiche Struktur enthalten sollte, darunter Waldflächen, Felder, Siedlungen unterschiedlicher Größe (sowohl kleinere als auch größere Gemeinden) sowie Wasserflächen. Größere Gebiete können für das Training und Testen genutzt werden, um den Arbeitsprozess zu erleichtern. Darüber hinaus ist eine geographische Nähe zu meinem Wohnort von Vorteil, insbesondere für eventuell notwendige visuelle Überprüfungen. Unter Berücksichtigung dieser Anforderungen eignet sich der Flugblock „Nordburgenland“ am besten. Dieser wurde zum Zeitpunkt des Beginns dieser Masterarbeit neu befliegen, sodass die Orthofotos aus dem Jahr 2022 stammen. Die Daten waren für Bezieher ab Juli 2023 verfügbar (BEV Datenkatalog, 2023).

Der Flugblock „Nordburgenland“ umfasst das Leithagebirge, die Vorläufer der Alpen (den Wienerwald) sowie größere Städte und Stadtgemeinden wie Wiener Neustadt, Baden und Mödling im niederösterreichischen Industrieviertel oder Eisenstadt und Neusiedl am See im Norden Burgenlands. Für die praktische Umsetzung wird nicht das gesamte Untersuchungsgebiet genutzt. Stattdessen wird ein innerhalb des Flugblocks liegendes Trainingsgebiet ausgewählt, das die definierten Kriterien erfüllt.

Die Modellvorhersagen werden anschließend auf einem separaten Testgebiet überprüft, das ebenfalls ein Ausschnitt dieses Flugblocks darstellt.

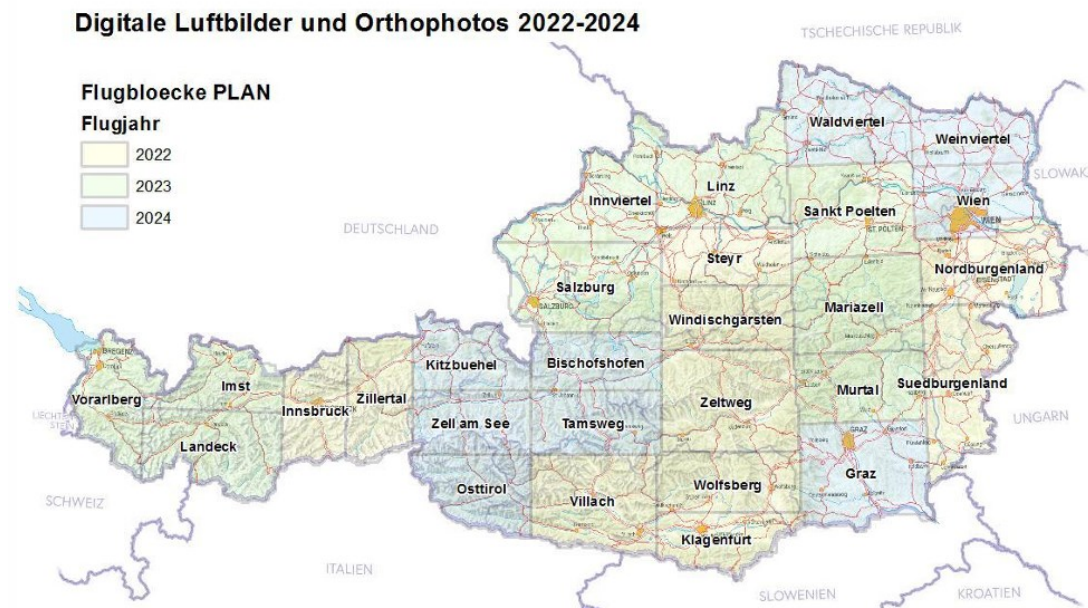


Abb. 26: Blöcke für die Luftbildaufnahme
(Photogrammetrische Datenerfassung (W2021, Vorlesungsskript, Universität Wien))

6.2.2 Orthofotos

Ein Orthofoto ist eine entzerrte und maßstabsgetreue Abbildung unserer Erdoberfläche. Orthofotos werden in Österreich aus Luftbildern digitaler Kameras erstellt und verfügen über eine geometrische Bodenauflösung von 20cm. Produzent der Daten ist das Bundesamt für Eich- und Vermessungswesen (BEV). Neben Orthofotos in Echtfarbe (RGB) stehen auch Varianten mit vier Kanälen (RGBl) sowie Falschfarbenbilder im infrarotnahen Farbbereich (CIR) zur Verfügung (Bundesamt für Eich- und Vermessungswesen (BEV), 2024).

„Als Grundlage für die Berechnung der Orthophotos dienen Geländehöhendaten, welche aus dem DGM von Österreich bereitgestellt werden. Die erforderlichen Parameter für die Orientierung der Luftbilder werden über luft sichtbare Passpunkte und GPS-Positionierung der Aufnahmepunkte berechnet. [...] Mit Hilfe der Elemente der äußeren Orientierung und eines Höhenrasters erfolgt die Berechnung des Orthophotos durch eine neue Zuordnung der Bildelemente. Benachbarte Orthophotos eines gemeinsamen Flugprojekts werden seit dem Produktionsjahr 2001 durch Mosaikieren zu einem einheitlichen Bild zusammengefügt.“ (BEV Datenkatalog, 2023).

Für diese Masterarbeit wird ein Orthofoto verwendet, das in Form von Rasterdaten vorliegt. Konkret handelt es sich dabei um Mosaik im Format Cloud Optimized GeoTIFF (COG), die flugblockweise zusammengestellt und komprimiert wurden. Laut Aussage des BEV ist die Kompression verlustbehaftet. Dabei wurde die Anzahl der Bits reduziert, und es kam ein YCbCr JPEG-Verfahren mit einer Qualität von 75% zum Einsatz. Dadurch beträgt die Pixeltiefe nun 8 Bit (BEV, persönliche Mitteilung, 24. Juni 2024). Nach Angaben des BEV enthält das genutzte Cloud Optimized GeoTIFF drei Kanäle (Rot, Grün, Blau). Zusätzlich existiert ein separates Mosaik mit einem Kanal für den nahen Infrarotbereich. Der Flugblock „Nordburgenland“ verwendet das Koordinatensystem *Austria GK East*

Objekterkennungsmodell, das unüberwachte Vortrainingsstrategien mit Region Priors – also vorherige Informationen über Regionen – kombiniert, wird ausgeschlossen, da es in einem unüberwachten Verfahren ohne Annotationen vortrainiert wird und somit nicht den Anforderungen dieser Arbeit entspricht (Bar et al., 2022).

Kriterium	EfficientDet ¹	FasterRCNN ²	MaskRCNN ³	RetinaNet ⁴	SSD ⁵	YOLOv3 ⁶
Architektur	Einstufig	Zweistufig	Zweistufig	Einstufig	Einstufig	Einstufig
Erkennungsstrategie	Bounding Box	Bounding Box	Bounding Box & Maske	Bounding Box	Bounding Box	Bounding Box
Detailgrad der Vorhersagen	Hoch	Hoch	Hoch (pixelgenau)	Hoch	Mittel	Mittel
Geschwindigkeit	Schnell	Langsam	Langsam	Mittel	Schnell	Sehr schnell
Effizienz bei geringer Rechenleistung	Sehr hoch	Mittel	Niedrig	Mittel bis hoch	Hoch	Hoch
Vorteile	Sehr effizient und skalierbar, anpassbar an verschiedene Aufgaben	Hohe Genauigkeit	Präzise Segmentierung	Hohe Präzision durch Focal Loss für schwer erkennbare Objekte	Schnelle Echtzeit-Erkennung	Effizient und schnell
Nachteile	Komplexere Architektur, erfordert Feinabstimmung	Langsam, hohe Rechenleistung	Hohe Rechenleistung, langsamer	Langsamer als YOLOv3, aber höhere Genauigkeit		Geringere Genauigkeit
Eignung für Masterarbeit	Sehr effizient bei geringer Rechenleistung	Vergleich für Genauigkeit, aber langsamer	Für detaillierte Segmentierung, hohe Anforderungen	Gute Genauigkeit bei schwer erkennbaren Objekten		Gut für schnelle Erkennung auf beschränkten Ressourcen

Tab. 2: Vergleich unter zur Auswahl stehender, vortrainierter Modelle

(¹Tan et al. (2020); ²Ren (2015); ³He et al. (2017); ⁴Lin (2017); ⁵Liu et al. (2016); ⁶Redmon (2018))

Da kein Modell perfekt ist, weisen auch die ausgewählten Modelle sowohl Vor- als auch Nachteile auf. Modelle wie RetinaNet, SingleShotDetector (SSD), YOLOv3 sowie EfficientDet wurden nicht in die Analyse einbezogen. Obwohl EfficientDet ein vielversprechendes einstufiges Modell mit guter Skalierbarkeit ist, fiel die Wahl auf zweistufige Modelle mit höherer Präzision, da diese für die Fragestellung der Arbeit – die möglichst genaue Identifizierung von Photovoltaikanlagen – relevanter sind. Die ausgewählten Modelle werden in den folgenden Kapiteln detailliert hinsichtlich ihrer Merkmale, Architektur sowie Vor- und Nachteile vorgestellt.

Die Wahl fiel auf Faster R-CNN und Mask R-CNN, da beide Modelle zur Familie der zweistufigen Detektoren gehören und für ihre hohe Erkennungsgenauigkeit bekannt sind. Faster R-CNN stellt eine Weiterentwicklung früherer Objekterkennungsmodelle dar und integriert erstmals ein Region Proposal Network (RPN), um Regionenvorschläge direkt während des Trainings zu generieren. Mask R-CNN erweitert dieses Konzept um eine zusätzliche Segmentierungskomponente, wodurch nicht nur Objekte lokalisiert, sondern auch deren exakte Silhouetten erkannt werden können. Dieses Modell deckt somit

zwei zentrale Aufgabenbereiche der Computer Vision ab: Objekterkennung und Instanzsegmentierung. Diese beiden Modelle ermöglichen es, die Forschungsfrage gezielt zu untersuchen, indem analysiert wird, ob die zusätzliche Segmentierungsebene von Mask R-CNN einen Mehrwert für die Erkennung von Photovoltaikanlagen bietet oder ob Faster R-CNN als reines Objekterkennungsmodell bereits hinreichend präzise arbeitet. Zudem bieten sie eine gute Vergleichsbasis, da sie auf einer ähnlichen Architektur fußen, sich jedoch in ihrer Funktionsweise und ihrem Rechenaufwand unterscheiden. Während Faster R-CNN effizienter arbeitet, jedoch nur Bounding Boxes liefert, könnte Mask R-CNN aufgrund seiner zusätzlichen Segmentierungsebene eine genauere Identifizierung ermöglichen.

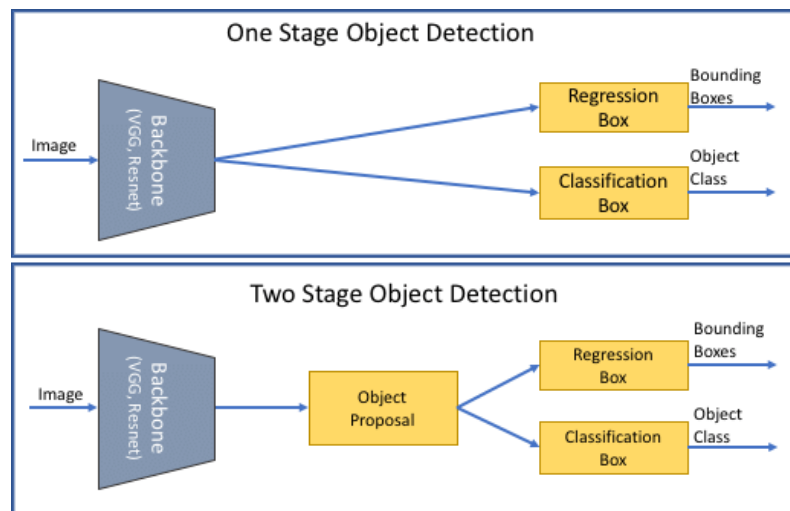


Abb. 28: Vergleich von ein- und zweistufigen Netzwerken in der Objekterkennung
(Corcoll, 2021)

6.3.1 Faster R-CNN

Das Modell Faster R-CNN stellt eine bedeutende Verbesserung gegenüber früheren Objekterkennungsmodellen wie R-CNN und Fast R-CNN dar („Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“ von Ren, 2015). Vor Faster R-CNN verwendeten diese Modelle sogenannte Selective-Search-Algorithmen, um Regionenvorschläge zu generieren und potenzielle Objekte im Bild zu lokalisieren. Diese Algorithmen sind jedoch langsam und zeitaufwendig, da sie als separate Schritte außerhalb des neuronalen Netzes durchgeführt werden. Faster R-CNN integriert das **Region Proposal Network (RPN)** direkt in das neuronale Netz, wodurch Regionenvorschläge und Klassifikation in einem gemeinsamen Netzwerk verarbeitet werden (Ren, 2015).

6.3.1.1 Architektur von Faster R-CNN

- **Convolutional Layer:** Diese Schichten extrahieren Merkmale (Features) aus dem Bild.
- **Region Proposal Network (RPN):** Eine zentrale Neuerung des Modells. Es generiert Regionenvorschläge für die extrahierten Merkmale direkt während des Trainings, wodurch die Effizienz und Geschwindigkeit der Objekterkennung erheblich verbessert wird.
- **RoI Pooling:** Das Region-of-Interest Pooling transformiert die vorgeschlagenen Regionen in eine feste Größe, sodass sie für die Klassifikation genutzt werden können.
- **Fully Connected Layer:** Diese Schicht führt die endgültige Objekterkennung und -klassifizierung durch (Ren, 2015).

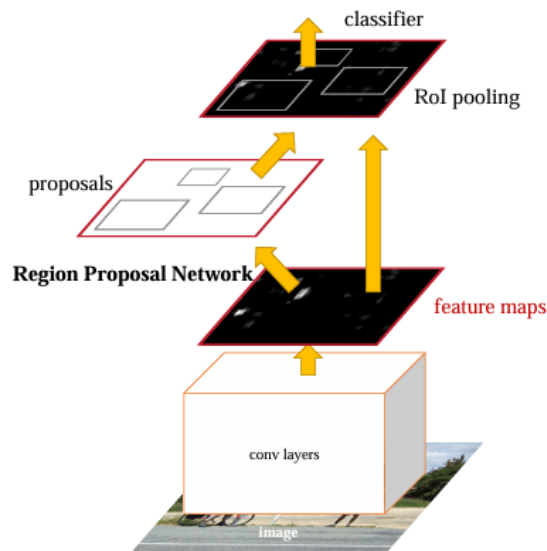


Abb. 29: Architektur von Faster R-CNN
(Ren, 2015)

6.3.1.2 Vor- und Nachteile von Faster R-CNN

Faster R-CNN bietet eine hohe Genauigkeit und segmentiert Objekte detailliert. Durch das RPN entfallen die langsamen Selective-Search-Algorithmen, wodurch Regionenvorschläge und Objekterkennung in einem einzigen Netzwerk stattfinden. Zudem verbessern sogenannte Anchor Boxes die Erkennung unterschiedlich großer Objekte und Formen. Allerdings ist Faster R-CNN im Vergleich zu Echtzeit-Detektoren wie YOLOv3 oder SSD nach wie vor relativ langsam, insbesondere bei hochauflösenden Bildern und großen Datensätzen. Zudem erfordert das Modell eine hohe Rechenleistung, da sowohl RPN als auch die Klassifikation separat durchgeführt werden (zweistufiges Verfahren). Dadurch ist Faster R-CNN komplexer und schwieriger zu implementieren und zu optimieren. Besonders auf Systemen mit begrenzter Hardware oder mobilen Endgeräten kann die Erkennungszeit problematisch sein (Ren, 2015).

6.3.2 Mask R-CNN

Mask R-CNN stellt eine Erweiterung des Faster R-CNN-Modells dar („Mask R-CNN“ von He et al., 2017). Neben der Objekterkennung ermöglicht Mask R-CNN auch eine Instanzsegmentierung. Das Modell identifiziert nicht nur Objekte im Bild und markiert deren Position mit Bounding Boxes, sondern erstellt zusätzlich für jedes erkannte Objekt eine pixelgenaue Maske, was dieses Modell vielseitig macht. Die Segmentierung erfolgt zusätzlich zur Objekterkennung und Klassifizierung in einer separaten Vorhersageebene. Eine wesentliche Verbesserung ist die Einführung von **RoIAlign (Region-of-Interest Align)**, das die bisher verwendete Methode RoI Pooling (Region-of-Interest Pooling) ersetzt. Während RoI Pooling durch Abrundung der Bounding Boxes zu Ungenauigkeiten führte, eliminiert RoIAlign dieses Problem durch präzisere Berechnungen und verbessert so die Segmentierungsgenauigkeit erheblich (He et al., 2017).

6.3.2.1 Architektur von Mask R-CNN

- **Backbone-Netzwerk:** Mask R-CNN verwendet dieselben **Backbones** wie Faster R-CNN, typischerweise ResNet oder ResNeXt in Kombination mit einem Feature Pyramid Network (FPN). Diese Backbones extrahieren Merkmale aus den Bildern auf verschiedenen Ebenen.

- **Region Proposal Network (RPN):** Wie Faster R-CNN nutzt Mask R-CNN ein RPN zur Generierung von Regionenvorschlägen.
- **RoIAlign:** Eine präzisere Alternative zu RoI Pooling. Neben Klassifikation und Bounding Box-Vorhersage führt Mask R-CNN eine dritte Verzweigung für die Masken-Vorhersage ein. Dies ermöglicht eine pixelgenaue Bestimmung von Silhouetten für jedes erkannte Objekt, während Ansätze wie Faster R-CNN nur Bounding Boxes zur Lokalisierung nutzt (He et al., 2017).

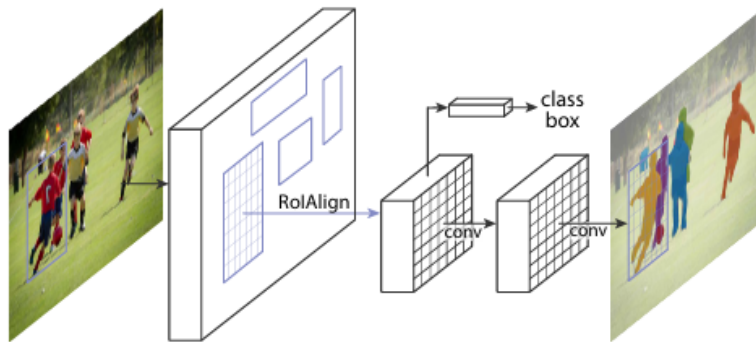


Abb. 30: Architektur von Mask R-CNN
(He et al., 2017)

6.3.2.2 Vor- und Nachteile von Mask R-CNN

Der größte Vorteil von Mask R-CNN ist die Fähigkeit, für jedes Objekt eine präzise Maske zu generieren. Die Einführung von RoIAlign verbessert zudem die genaue Lokalisierung von Objekten und Masken erheblich. Das Modell kann gleichzeitig Objekterkennung, Instanzsegmentierung und sogar Schlüsselpunkt-Erkennung durchführen, was es für viele Anwendungen besonders vielseitig macht. Allerdings ist Mask R-CNN rechenintensiver als einfachere Objekterkennungsmodelle wie SSD oder YOLOv3, insbesondere aufgrund der zusätzlichen Masken-Vorhersage und parallelen Verarbeitungsschritte. Da es auf einem zweistufigen Verfahren basiert (Region Proposal gefolgt von Klassifikation und Segmentierung), ist es langsamer als einstufige Modelle. Zudem kann Mask R-CNN Schwierigkeiten bei der Segmentierung sehr kleiner Objekte haben, da deren Merkmale auf den Feature Maps weniger deutlich ausgeprägt sind. Insgesamt erfordert Mask R-CNN eine komplexe Implementierung und eine aufwendigere Optimierung (He et al., 2017).

6.4 Grundvoraussetzungen für die praktische Umsetzung

Das folgende Gerät mit den angegebenen Gerätespezifikationen wird für die Deep Learning-Aufgaben verwendet:

Gerät	Acer Aspire 7 (A715-74G)
RAM	16 GB
Prozessor (CPU), x64-basiert	Intel® Core™ i7-9750H CPU @ 2.60GHz
Grafikkarte (GPU)	NVIDIA® GeForce® GTX 1650 mit 4GB dedizierter VRAM
Speicher (SSD)	475 GB (nominell sind 512 GB angegeben), davon 213 GB frei

Tab. 3: Gerät und Spezifikationen

Ein 64-Bit-Betriebssystem sowie bestimmte Versionen der verwendeten Software sind erforderlich, um eine ordnungsgemäße Funktionalität zu gewährleisten. In dieser Arbeit wird die Version 2.3.0 der

ArcGIS API for Python genutzt, die bereits in ArcGIS Pro 3.2 enthalten ist. Für die Nutzung bestimmter Werkzeuge in ArcGIS Pro müssen zudem die Erweiterungen *Image Analyst* und *Spatial Analyst* lizenziert sein.

Name	Licensed	Expires
Image Analyst	Yes	31.01.2025
Indoors	No	N/A
LocateXT	Yes	31.01.2025
Location Referencing	Yes	31.01.2025
Maritime	Yes	31.01.2025
Network Analyst	Yes	31.01.2025
Production Mapping	Yes	31.01.2025
Publisher	Yes	31.01.2025
Reality	No	N/A

Name	Licensed	Expires
Spatial Analyst	Yes	31.01.2025
StreetMap Premium Asia Pacific	No	N/A
StreetMap Premium Europe	No	N/A
StreetMap Premium Japan	No	N/A
StreetMap Premium Latin America	No	N/A
StreetMap Premium Middle East and Africa	No	N/A
StreetMap Premium North America	No	N/A
Workflow Manager	Yes	31.01.2025

Abb. 31: Erweiterungen Image Analyst und Spatial Analyst
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Für Python muss mindestens Version 3.9 und höchstens Version 3.11 installiert sein, um die Kompatibilität mit der eben genannten API zu gewährleisten. Zusammenfassend sind softwareseitig folgende Voraussetzungen und Softwareversionen für eine kompatible Nutzung wichtig:

Software	Version
Windows 10 Home (64-Bit-Version)	22H2
ArcGIS Pro	3.2 mit Patch 2 (3.2.2)
Deep Learning Libraries for ArcGIS Pro 3.2	3.2.49743
ArcGIS API for Python (enthalten in ArcGIS Pro 3.2)	2.3.0
Python	3.9

Tab. 4: Genutzte Software und Versionen

Um die Grafikkarte (GPU) samt CUDA (Compute Unified Device Architecture) bei den Berechnungen nutzen zu können, müssen bestimmte Voraussetzungen erfüllt sein. Dazu gehört insbesondere eine CUDA-fähige Grafikkarte. Mein Gerät, eine NVIDIA® GeForce® GTX 1650, erfüllt diese Anforderung aufgrund der CUDA Compute Capability-Version (NVIDIA Developer, 2024). Die Mindestversionen des GPU-Treibers sowie der erforderlichen **PyTorch**-Bibliothek sind der folgenden Tabelle zu entnehmen (GitHub, 2024). **PyTorch** ist in diesem Zusammenhang wichtig, weil es ermöglicht, Berechnungen sowohl auf der GPU als auch auf der CPU auszuführen.

Software	Erforderliche Mindestversion	Installierte Version
CUDA Compute Capability	5.0	7.5
CUDA	11.1.0	11.1.1
GPU-Treiber	527.41	555.85
PyTorch (Deep Learning Libraries for ArcGIS Pro 3.2)	1.8.2	1.8.2

Tab. 5: Software, die Mindestversionen erfordern sowie installierte Versionen

Mit dem Befehl `nvidia-smi` kann in der Eingabeaufforderung geprüft werden, welche GPU-Treiberversion installiert ist. In diesem Fall handelt es sich um Version 555.85, welche für die Aufgaben qualifiziert ist.

```

Eingabeaufforderung
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\...>nvidia-smi
Wed Oct 30 20:57:35 2024

+-----+
| NVIDIA-SMI 555.85                  Driver Version: 555.85          CUDA Version: 12.5   |
+-----+-----+-----+-----+-----+-----+
| GPU   Name                               Driver-Model  Bus-Id         Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf              Pwr:Usage/Cap     Memory Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+
|  0  NVIDIA GeForce GTX 1650      WDDM          00000000:01:00:0  On  |        N/A           |
| N/A   42C    P8              2W /   50W         479MiB /   4096MiB |        5%    Default |
+-----+-----+-----+-----+-----+-----+

+-----+
| Processes: |
| GPU   GI   CI        PID   Type   Process name                      GPU Memory |
| ID     ID   ID             |              | Usage     |
+-----+-----+-----+-----+-----+
|  0   N/A  N/A       9264    C+G    ... Files\ArcGIS\Pro\bin\ArcGISPro.exe  N/A       |
+-----+

C:\Users\...

```

Abb. 32: NVIDIA-Befehl zur Abfrage von GPU-Informationen
(Eigene Darstellung, erstellt mit Windows CMD)

6.5 Klassifikation von Trainingsdaten

Für die Klassifizierung wurden insgesamt zwei Trainingsdatensätze erstellt: Einer, in dem nur Photovoltaikanlagen berücksichtigt werden, und der andere, in dem sowohl Photovoltaik- als auch Solarthermieranlagen als Klassen festgelegt und annotiert wurden.

Wie bereits in Kapitel 5.8.1 erwähnt, wird in der Literatur kaum darauf eingegangen, dass Photovoltaik- und Solarthermieranlagen eine optische Ähnlichkeit aufweisen. Aus der Nähe ist der Unterschied zwar mit bloßem Auge erkennbar, jedoch stellt sich die Frage, ob ein Deep-Learning-Algorithmus dies auf dem 0,2m-Orthomosaik korrekt erkennen kann. Aus diesem Grund wurde der zusätzliche Trainingsdatensatz mit weiteren Annotationen für die Objektart „Solarthermieranlage“ (Klasse 2) erstellt.

Objekt	Klasse/Annotation
Datensatz 1	
Photovoltaikanlage (kurz PVA)	Class_1
Datensatz 2	
Photovoltaikanlage (kurz PVA)	Class_1
Solarthermieranlage (kurz STA)	Class_2

Tab. 6: Für das Training festgelegte Klassen

Vor dem eigentlichen Trainingsprozess werden die Daten in einen Trainings- und Validierungsdatensatz aufgeteilt. Dabei werden 80% der Daten für das Modelltraining verwendet, während die verbleibenden 20% für die Validierung genutzt werden. Der Validierungsdatensatz dient nach jeder Epoche als Kontrollinstanz, um die Genauigkeit sowie Reliabilität des Modells zu überprüfen.

Zudem wird ein Testdatensatz erstellt, dessen Annotationen manuell erhoben werden. Diese Daten werden nicht für das Training eingesetzt, sondern vollständig vom Modelltraining (einschließlich des Validierungsdatensatzes) ferngehalten. Das Testgebiet wurde im Gegensatz zum Trainingsgebiet einer visuellen Überprüfung unterzogen, bei der vor Ort sichergestellt wurde, dass Photovoltaikanlagen korrekt klassifiziert wurden. So können diese Daten mit hoher Sicherheit als Kontrollmaßstab dienen. Die Leistung der Modelltrainings kann durch den Vergleich der Überlappung zwischen den Modellvorhersagen (Detektion bzw. Inferenz) und den manuell erhobenen Annotationen (Ground Truth-Daten des Testgebiets) überprüft werden. Eine Konfusionsmatrix kann dabei helfen, den aktuellen Status zu ermitteln.

Die Arbeit in ArcGIS Pro erfolgt ausschließlich in englischer Sprache, weshalb die Werkzeugnamen und Parameter ebenfalls auf Englisch angegeben werden. Da es sich meist um GIS-spezifische Fachbegriffe aus dem englischen Sprachraum handelt, ist dies ohnehin vorzuziehen.

Der Workflowdiagramm (siehe Abbildung 33) bietet einen groben grafischen Überblick darüber, wie die praktische Umsetzung erfolgt ist. Zuerst wurde im GIS ein neues Projekt mit einer eingebetteten Karte angelegt. Die notwendigen Grunddaten werden hinzugefügt, darunter sowohl das RGB- und NIR-Orthomosaik des BEV als auch eine öffentlich zugängliche Grundkarte des kartographischen Produkts *basemap.at* im Format eines WMTS (Web Map Tile Service) (*basemap.at*, 2024). *Basemap.at* wird standardmäßig im Web Mercator Auxiliary Sphere (EPSG:3857) bereitgestellt. Da die Orthomosaik in EPSG:31256 vorliegen, wird die Projektion der Karte in ArcGIS Pro im Bereich *Contents* manuell auf dieses Koordinatensystem geändert, sofern das noch nicht geschehen ist.

In Abbildung 34 ist das Gesamttrainingsgebiet abgebildet. Zunächst wurden der einfachen Handhabung und aufgrund von Trainingsprobedurchläufen drei kleinere Rasterdatensätze aus dem ursprünglichen Orthomosaik des Blocks "Nordburgenland" ausgeschnitten und nach den größten Gemeinden benannt: Traiskirchen, Guntramsdorf und Mödling. Dazu wurde das Data Management-Werkzeug **Clip Raster** verwendet, um das entstandene Raster vom Input, dem Orthomosaik, mittels einer angelegten Polygon-Feature-Class zu trennen. Die Feature-Classes wurden so erstellt, dass unterschiedliche Landschaftstypen auf den Bilddaten sichtbar sind. Am wichtigsten sind jedoch ohnehin die Siedlungsgebiete. Die Wahl fiel auf südlich von Wien gelegene Gemeinden. Die Trainingsgebiete 1 („Traiskirchen“) und 2 („Mödling“) umfassen viele Gewerbeflächen mit Photovoltaikanlagen auf den Dächern, sodass in diesen Gegenden viele Annotationen erstellt werden konnten (Stadtgemeinde Traiskirchen, 2024). Die rechte Hälfte des Trainingsgebiets „Mödling“ umfasst auch das Industriezentrum Niederösterreich, welches aufgrund einer kürzlichen Investition Photovoltaikanlagen aufweist (Niederösterreichische Landesregierung, 2020).

Für den Export der Trainingsdaten wurden die drei einzelnen Gebiete schließlich mit dem Werkzeug **Mosaic To New Raster** zusammengefügt. Die resultierende Kombination der drei Raster ergibt ein neues Bild im TIFF-Format. Dieser Prozess erfolgte jedoch erst später. Die Gesamtfläche des Trainingsgebietes beträgt etwa 35,06km² (3506ha).

WORKFLOW-DIAGRAMM

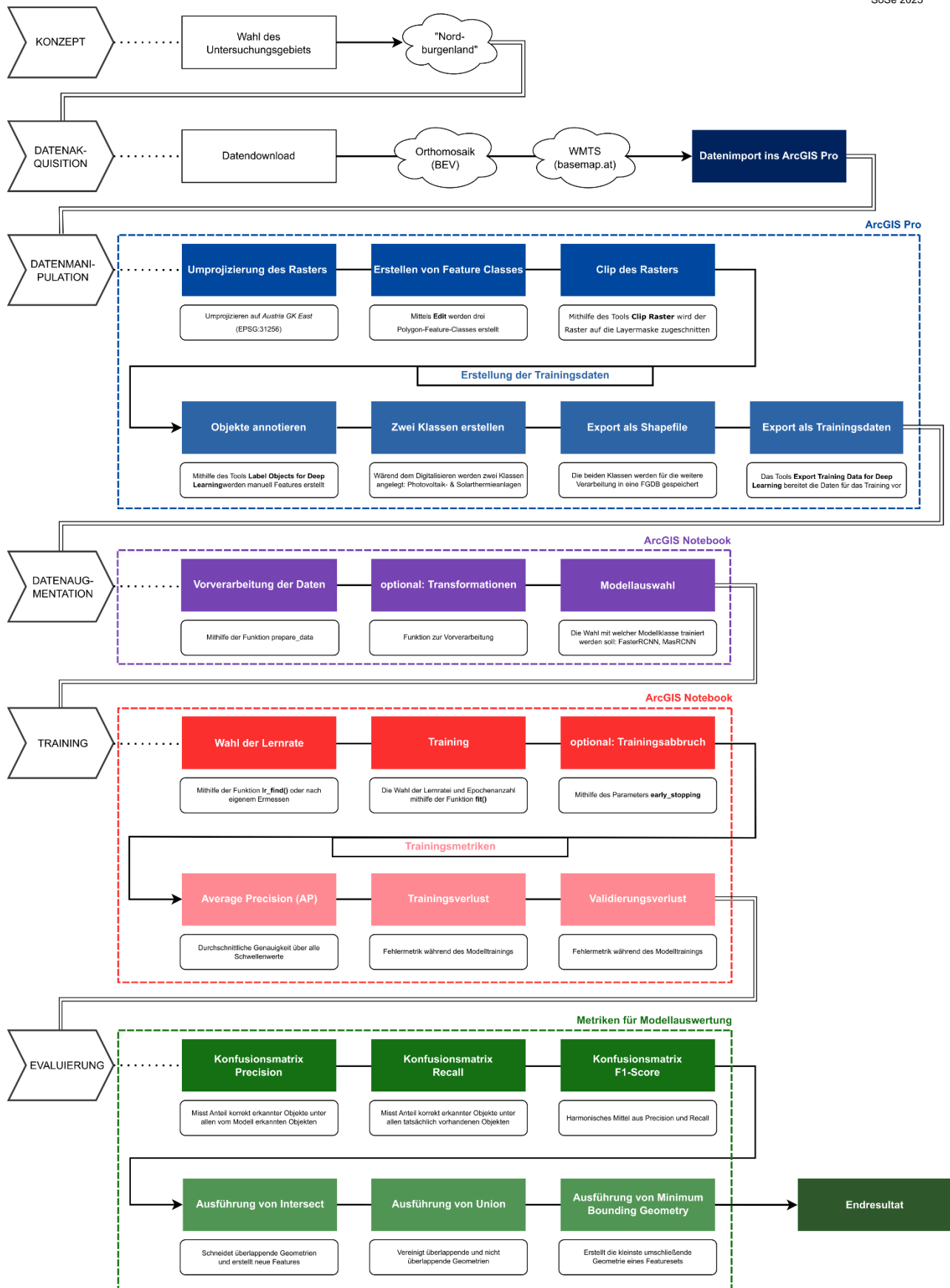




Abb. 34: Gesamttrainingsgebiet, eigene Darstellung, erstellt mit ArcGIS Pro 3.2
(Bundesamt für Eich- und Vermessungswesen (BEV), 2023)

ArcGIS Pro bietet mehrere Werkzeuge für Deep Learning-Workflows an. Mit dem Werkzeug **Label Objects for Deep Learning** können Objekte in den Daten annotiert werden, um diese als Trainingsdaten für Deep-Learning-Modelle zu verwenden. Es ermöglicht, Objekte auf Bildern bzw. Rasterdaten zu markieren und zu klassifizieren, die als Grundlage für ein Modell dienen (Esri, 2023d). Für die Klassifizierung wurde in Datensatz 1 eine Klasse mit dem Namen Class_1 für Photovoltaikanlagen (PVA) erstellt. In Datensatz 2 wurden zwei Klassen angelegt: Class_1 (für PVA) sowie eine weitere Klasse mit

dem Namen Class_2, die Solarthermieranlagen (STA) repräsentiert. Entsprechend dieser Klassen wurden die beiden Objekttypen als Polygone manuell digitalisiert.

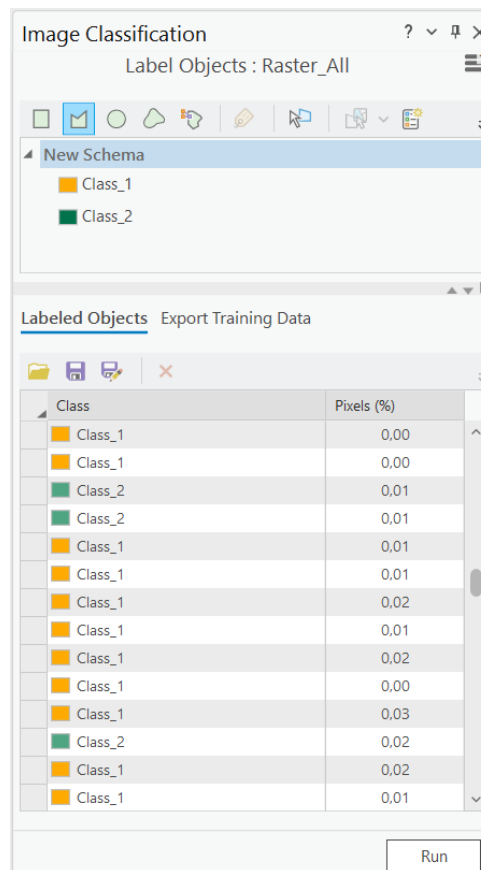


Abb. 35: Werkzeug für die Klassifikation
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Klasse	Klassenname	Klassenwert
Photovoltaikanlage	Class_1	1
Solarthermieranlage	Class_2	2

Tab. 7: Klassennamen und -werte

Da es hierzu keine Referenzdaten gibt, basieren die Trainingssätze auf visuellen Schätzungen von Photovoltaik- und Solarthermieranlagen, die mithilfe des RGB- und NIR-Orthomosaiks getroffen wurden. Zusätzlich wurde Google Maps Street View als Hilfsmittel herangezogen, um schwer identifizierbare Objekte korrekt zu klassifizieren.

Die Identifikation von Objekten auf Flachdächern stellt bei der Klassifikation häufig eine Herausforderung dar. Hilfreich kann es sein, wenn bereits viele Annotationen vorliegen, um anhand von Form, Anordnung der Solarmodule und Farbe einzuschätzen, ob es sich um eine PV-Anlage handeln könnte. Die Farbzusammensetzung von Photovoltaikanlagen variiert, was die Klassifikation zusätzlich erschwert. Sie reicht von hellen, polykristallinen Modulen bis hin zu dunklen, monokristallinen Modulen, die optisch Solarthermieranlagen ähneln können. Während Photovoltaik Sonnenlicht mithilfe von Solarzellen direkt in elektrische Energie umwandelt, nutzt Solarthermie die Sonnenenergie, um Wasser oder eine Flüssigkeit zu erhitzen und in thermische Energie (Wärme) umzuwandeln, die zur Warmwasserversorgung oder Heizungsunterstützung dient. Photovoltaik- und Solarthermieranlagen

werden zusammenfassend als Solaranlagen bezeichnet. In Abbildung 36 ist der Unterschied zwischen Photovoltaikanlagen (unten) und Solarthermieranlagen (oben) zu erkennen.



Abb. 36: Unterschied zwischen Photovoltaik- und Solarthermieranlagen
(Ofenseite, 2024)

Beim Digitalisieren wurde zwar präzise, aber nicht pixelgenau gearbeitet, da eine solche Präzision mit einem hohen Zeitaufwand verbunden wäre. Das bedeutet, dass an die Objekte näher herangezoomt wurde, um sie bestmöglich vom Hintergrund abzugrenzen. Außerdem unterscheiden sich die digitalisierten Formen und Größen je nach Anordnung und Menge der Module (unabhängig davon, ob es sich um PVA oder STA handelt).

Nachdem alle Photovoltaikanlagen sowie Solarthermieranlagen im gesamten Trainingsgebiet klassifiziert wurden, wurden die Daten abschließend in einer File Geodatabase für den Export gespeichert. In der folgenden Tabelle ist dargestellt, wie viele Annotationen und Bilder nach dem Trainingsdatenexport insgesamt angelegt wurden. Es ist jedoch wichtig zu erwähnen, dass die Anzahl der Class_1-Daten im späteren Verlauf noch angepasst wurde (mehr dazu in Kapitel 6.5.1):

Objekt	Klasse/Annotation	Anzahl an Annotationen	Anzahl an Bildern
Datensatz 1			
Photovoltaikanlage	Class_1	6206	6206
Datensatz 2			
Datensatz 2a			
Photovoltaikanlage	Class_1	6206	6206
Datensatz 2b			
Solarthermieranlage	Class_2	5644	5644
		11 850	11 850

Tab. 8: Anzahl an Annotationen und Bildern

6.5.1 Export der erzeugten Trainingsdaten

Mit dem Werkzeug **Export Training Data for Deep Learning** können die annotierten Daten so aufbereitet werden, dass sie für das Training von Deep-Learning-Modellen geeignet sind. Dieses Werkzeug unterstützt verschiedene Frameworks und exportiert die Daten in Form kleiner

Bildausschnitte (sogenannte Chips) des Eingaberasters, zusammen mit den dazugehörigen Annotationen (Esri, 2023b). Der Eingaberaster ist in diesem Fall das zuvor erwähnte Rasterbild, das durch das Zusammenfügen aller Raster der drei Trainingsgebiete entstanden ist.

Mit der Funktion `arcpy.ia.ExportTrainingDataForDeepLearning()` wurde automatisiert der Export der Trainingsdaten mittels ArcPy durchgeführt. Hierbei wurden die wichtigsten Parameter wie Zellgröße, Schrittweite (Stride), Rotation und die minimale Überlappungsrate sorgfältig konfiguriert. Eine Übersicht der relevanten Parameter ist in Tabelle 9 dargestellt (Esri, 2023b):

Parameter	Erläuterung und Wahl
Input Raster <i>in_raster</i>	Der Raster, der für das Training des CNN-Modells verwendet werden soll.
Output Folder <i>out_folder</i>	Der Speicherort, an dem die exportierten Trainingsdaten und Metadaten gespeichert werden sollen.
Input Feature <i>in_class_data</i>	Die Feature-Class, die die Annotationen zu den Objekten enthält, die im Raster erkannt werden sollen.
Image Format <i>image_chip_format</i>	<u>Das Dateiformat der exportierten Daten:</u> Es wurde das TIFF -Format gewählt.
Tile Size X × Tile Size Y <i>tile_size_x × tile_size_y</i>	<u>Die Kachel- bzw. Zellgröße der Bildausschnitte bzw. Chips:</u> Größere Chips können besonders nützlich sein, wenn größere Objekte im Bild erfasst werden sollen, können aber auch die Anzahl der Trainingsbeispiele verringern, was die Variabilität des Modells beeinflussen kann. Die Wahl beläuft sich auf 256 × 256 Pixel .
Stride X × Stride Y <i>stride_x × stride_y</i>	<u>Der Abstand zwischen den Kacheln in Pixel:</u> Dieser Parameter definiert, wie weit sich die Kachel über die Chips hinweg verschieben. Wenn der Stride kleiner als die Tile Size ist, überlappen sich die Kacheln mehr. So kommen mehr Chips zustande, die ähnliche Bereiche abdecken. Es wurden 128 × 128 Pixel gewählt.
Rotation Angle <i>rotation_angle</i>	<u>Gibt den Rotationswinkel in Grad an:</u> Dies steuert, um welchen Winkel die Trainingschips gegen den Uhrzeigersinn gedreht werden, um Variationen für das Training des Modells zu erzeugen. Es wurden 90° sowie 180° definiert, sodass vier sowie zwei Chips aus demselben Bild erzeugt wurden (für 0°, 90°, 180° und 270° sowie 0° und 180°). Diese Methode gehört zur Datenaugmentation.
Minimum Polygon Overlap Ratio <i>min_polygon_overlap_ratio</i>	<u>Prozentsatz für Überlappung von Merkmalen im Bild:</u> Hiermit wird die minimale Überlappungsrate festgelegt, um Merkmale aus den Datenchips zu entfernen, die kleiner als der angegebene Prozentsatz sind. Es wurden 0,25 (für PVA) sowie 0,1 (für STA) definiert.
Metadata Format <i>metadata_format</i>	<u>Speicherformat der exportierten Trainingsdaten:</u> Die Wahl fiel auf PASCAL Visual Object Classes , da dieses Format am gängigsten für Aufgaben in der Objekterkennung genutzt wird. Es speichert Informationen zu den Bounding Box-Bereichen und Annotationen in

XML-Dateien und ist kompatibel mit vielen Frameworks wie z. B. PyTorch und TensorFlow.
--

Tab. 9: Erläuterung der Parameter des Werkzeugs „Export Training Data for Deep Learning“

Das Skript zum Trainingsdatenexport ist im Anhang B zu finden, welches mithilfe des Python-Moduls **ArcPy** geschrieben wurde (Esri, 2023b). Beispiele für die resultierenden Bildchips sind in Abbildung 37 bis Abbildung 40 dargestellt. Die Chips haben eine Größe von 256×256 Pixel. Zur Veranschaulichung wurden rechts neben den Bildausschnitten die entsprechenden Annotationen aus der Feature-Class abgebildet, in der diese gespeichert sind.



Abb. 37: Beispiel eines Gebäudes mit PVA und dazu die Annotationen (v.l.n.r.)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

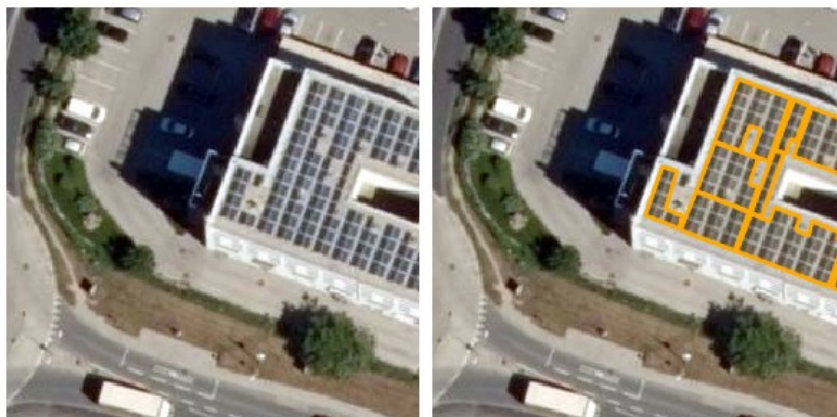


Abb. 38: Beispiel von Aufdach-PVA und dazu die Annotationen (v.l.n.r.)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)



Abb. 39: Beispiel von Gebäuden mit PVA sowie STA und dazu die Annotationen (v.l.n.r.)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)



Abb. 40: Beispiel eines Gebäudes mit STA und dazu die Annotationen (v.l.n.r.)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

6.6 Datenvorverarbeitung

Vor dem eigentlichen Training der Modelle ist eine gezielte Aufbereitung der Daten notwendig, um eine möglichst hohe Modellleistung zu erzielen. Dieses Kapitel beschreibt die durchgeführten Schritte der Datenaugmentationsmethode Rotation zur Erhöhung der Datenvielfalt und zur Ausbalancierung der Klassenverteilung.

6.6.1 Datenaugmentationsmethode Rotation

In diesem Kapitel wird erneut auf die **Rotation** als Datenaugmentationsmethode eingegangen. Diese wurde bereits beim Exportieren der Trainingsdaten in Kapitel 6.5.1 realisiert, indem der Parameter `rotation_angle` mit den Werten 90° für STA und 180° für PVA definiert wurde.

Dazu war es notwendig, die Klassifikationsdaten, die ursprünglich in einem Gesamtdatensatz mit beiden Klassen vorlagen, in zwei separate Datensätze aufzuteilen: Einen mit ausschließlich Photovoltaikanlagen (Datensatz 2a) und einen weiteren mit ausschließlich Solarthermieranlagen (Datensatz 2b). Der Grund für diese Aufteilung liegt darin, dass im Gesamtuntersuchungsgebiet deutlich weniger Solarthermieranlagen vorkommen und entsprechend weniger Annotationen vorhanden sind als Photovoltaikanlagen. Um ein möglichst ausgewogenes Verhältnis zwischen den Klassen zu erreichen, war es erforderlich, die Klassen innerhalb der Funktion `arcpy.ia.ExportTrainingDataForDeepLearning()` separat zu behandeln und zu verarbeiten (siehe Anhang B). Nach dem Export und nach weiteren Anpassungen wurden die beiden Datensätze wieder zusammengeführt.

Der Hintergrund hierzu ist, dass ein Ungleichgewicht in der Anzahl der Trainingsbeispiele zwischen den Klassen dazu führen kann, dass das Modell die besser repräsentierte Klasse priorisiert, was eine Verzerrung (engl.: Bias) verursachen kann. In diesem Fall trainiert das Modell überwiegend die dominante Klasse, wodurch diese einen Vorteil erhält, während die schwächere Klasse unterrepräsentiert bleibt. Um das zu vermeiden, sollte der Unterschied in der Größe der Klassen möglichst gering gehalten werden (Kalita, 2022; Ng & Soo, 2018). Ein ausgewogener Datensatz verbessert außerdem die Generalisierungsfähigkeit des Modells, was bedeutet, dass das Modell in der Lage ist, die in den Trainingsdaten gelernten Merkmale und Muster auf neue, nicht gesehene Daten anzuwenden, um akkurate Vorhersagen zu treffen. Wenn beide Klassen gleichmäßig vertreten sind, steigt die Wahrscheinlichkeit, dass das Modell die spezifischen Merkmale jeder Klasse richtig lernt und

anwendet, was die Generalisierung und letztlich die Gesamtgenauigkeit verbessert (Burkov, 2019; Ng & Soo, 2018). Falls eine solche Ausgewogenheit nicht gewährleistet werden kann, wie in diesem Fall, dann können Strategien zur Datenaugmentation helfen, die Auswirkungen des Ungleichgewichts zu minimieren und die Modellleistung zu optimieren. Diese Aspekte sollten bei der Vorbereitung des Trainingsdatensatzes berücksichtigt werden, um möglichst präzise Ergebnisse zu erzielen (Burkov, 2019).

Nach dem regulären Export beider Klassen stehen unter Berücksichtigung der Rotation sowie der minimalen Überlappungsrate insgesamt 11 850 Bildchips und Annotationen zur Verfügung. Davon entfallen 6206 auf Photovoltaikanlagen (PVA) und 5644 auf Solarthermieranlagen (STA). Zwar konnte keine exakt gleiche Anzahl an STA-Bildchips erreicht werden, doch durch die Angabe eines zusätzlichen Rotationswinkels im Werkzeug **Export Training Data for Deep Learning** ließ sich die Anzahl der Bilddaten und Annotationen beider Klassen angleichen. Zusätzlich wurde die Überlappungsrate angepasst, um das Ungleichgewicht weiter zu verringern. Für PVA wurde ein größerer Wert angegeben, sodass mehr Bilder und Labels entfernt wurden, was bei dem ohnehin größeren PVA-Datensatz weniger ins Gewicht fiel. Im Gegensatz dazu war es bei den STA-Trainingsdaten entscheidend, jeden noch so kleinen Feature-Anteil zu erhalten, da es ohnehin zu wenige Beispiele dieser Klasse gibt.

Klasse	Initiale Anzahl	Rotation	Überlappungsrate	Angepasste Anzahl
Photovoltaikanlage	3224	180 (×2)	25%	6206
Solarthermieranlage	1411	90 (× 4)	10%	5644

Tab. 10: Veranschaulichung der Anzahl an Bildchips und Annotationen beider Klassen



Abb. 41: Rotation einer STA mit 0° (links oben), 90° (rechts oben), 180° (links unten) und 270° (rechts unten)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)



Abb. 42: Rotation einer PVA mit 0° (links) und 180° (rechts)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

6.6.2 Vorverarbeitung der Trainingsdaten

Da ich mit der Funktion `prepare_data` von Esri gearbeitet habe, die den Parameter `transforms` für Transformationen bzw. Datenaugmentationsmethoden (z. B. Skalierung, Spiegelung, Kontrast etc.) bereitstellt, konnte die Datenaugmentation nicht auf spezifische Trainingschips angewendet werden. In ArcGIS Pro und der ArcGIS-API (`arcgis.learn`-Bibliothek) besteht aktuell keine Möglichkeit, gezielt nur bestimmte Datenchips (z. B. nur Solarthermieanlagen) für Transformationen auszuwählen. Alle Transformationen werden global auf den gesamten Trainingsdatensatz angewendet. Daher ist es wichtig, die Daten bereits beim oder nach dem Export der Trainingsdaten mit Methoden vorzubereiten. Der Parameter `transforms` würde in diesem Fall nicht zu einer sinnvollen Datenaugmentation beitragen.

Nach anfänglichen Tests und Überlegungen wurde entschieden, beide Klassen auf die exakte gleiche Anzahl an Trainingschips und Annotationen zu bringen. Konkret bedeutet das, dass die PVA-Daten auf 5466 Bilder und Annotationen reduziert wurden, um eine faire Aufteilung beider Klassen im Training zu gewährleisten. Dieser Schritt wurde bewusst manuell durchgeführt und nicht per Skript, um gezielt Bild- und Annotationspaare zu entfernen, die beispielsweise nur einen minimalen Pixelanteil in den Bildern enthalten oder in einer bestimmten Form oder Anordnung vorkommen, die in den Trainingsdaten bereits ausreichend repräsentiert ist. Das ergibt jeweils 5644 Bildchips und 5644 Label-Dateien pro Klasse (siehe Tabelle 11). Für Datensatz 1, der ohnehin ausschließlich Klasse 1 (Photovoltaikanlagen) als Trainingsdaten enthält, sind keine weiteren Schritte erforderlich. Für Datensatz 2 hingegen müssen die zuvor einzeln exportierten Klassen nach dem manuellen Löschvorgang als Bildchips und Annotationen in einem neuen Verzeichnis zusammengeführt werden. Dies wurde mithilfe eines Skripts realisiert, das die Python-Bibliotheken `os`, `os.path` und `shutil` nutzt (Python Software Foundation, o.J.). Der entsprechende Python-Skript ist im Anhang A zu finden.

Beim Export werden im Gesamtverzeichnis neben den Unterverzeichnissen `images` und `labels` weitere Dateien angelegt, darunter `esri_accumulated_stats.json` und `esri_model_definition.emd`. Die Datei `esri_accumulated_stats.json` enthält ausschließlich die Daten des jeweiligen Exportvorgangs. Die Datei `esri_model_definition.emd` hingegen umfasst Metadaten wie Trainingsparameter, das Koordinatensystem und Informationen zu den Klassen. Die Dateien von Klasse 1 (PV-Anlagen aus dem Datensatz 2a) wurden ebenfalls mithilfe des Python-Skripts

zusammen mit den weiteren angelegten Textdateien map.txt und stats.txt in ein neu erstelltes Verzeichnis verschoben, damit diese im weiteren Verlauf überschrieben werden. Dieser Datensatz wurde auch als „Kombinierter Datensatz“ bezeichnet:

Objekt	Klasse/Annotation	Anzahl an Annotationen	Anzahl an Bildern
Datensatz 1			
Photovoltaikanlage	Class_1	6206	6206
Datensatz 2 = Kombiniertes Datensatz			
Photovoltaikanlage	Class_1	5644	5644
Solarthermieranlage	Class_2	5644	5644
		11 288	11 288

Tab. 11: Anzahl an Annotationen und Bildern nach der Anpassung als Endresultat

Wenn das Training auf dem kombinierten Datensatz durchgeführt wird, erzeugt der Trainingsprozess neue und konsolidierte Metadaten für den gesamten Datensatz. Es wird mit den tatsächlich vorhandenen Daten in den Verzeichnissen images und labels gearbeitet, was bedeutet, dass die ursprünglichen Statistiken aus dem separaten Export keine Konflikte verursachen. Das Modell lernt die Datenmerkmale und Klassenverteilung auf Basis des kombinierten Datensatzes erneut.

Bei den ersten Trainingsversuchen mit diesem Datensatz wurde ein Konflikt festgestellt, der auf die Datei esri_model_definition.emd zurückzuführen ist. Diese Datei spiegelt nur teilweise die Eigenschaften des Datenexports wider, da der Export beider Klassen separat erfolgte. In den jeweiligen esri_model_definition.emd-Dateien fehlt daher die Information über die jeweils andere Klasse. Aus diesem Grund wurde in esri_model_definition.emd, die zuvor in das neue Verzeichnis verschoben wurde, manuell ein weiterer Eintrag für Class_2 im Parameter Classes erstellt. Zudem wurde der korrekte Wert, der Name und die Farbe der Klasse hinzugefügt:

```

34  },
35  "Classes" : [
36    {
37      "Value" : 1,
38      "Name" : "Class_1",
39      "Color" : [
40        255,
41        170,
42        0
43      ]
44    },
45    {
46      "Value" : 2,
47      "Name" : "Class_2",
48      "Color" : [
49        0,
50        168,
51        132
52      ]
53    }

```

*Abb. 43: Nachträgliche Anpassung der EMD-Datei für den kombinierten Datensatz
(Eigene Darstellung, erstellt mit Notepad++ 7.9)*

6.6.3 Erste Arbeitsschritte in ArcGIS Pro

Die Funktion `prepare_data` wurde bereits zuvor erwähnt und dient der Vorbereitung der Datenobjekte der exportierten Trainingsdaten. Bevor sie angewendet werden kann, müssen zunächst die erforderlichen Bibliotheken importiert werden. Dies geschieht in der Notebook-Umgebung in ArcGIS Pro. Neben der Bibliothek `arcgis.learn` und der darin enthaltenen `prepare_data`-Funktion sowie den Funktionen der einzelnen Objekterkennungsmodelle werden auch weitere Bibliotheken wie z. B. `os`, `pathlib` und `time` importiert.

Des Weiteren ist es erforderlich, den Pfad zu den Trainingsdaten zu definieren, die zuvor mit `arcpy.io.ExportTrainingDataForDeepLearning()` erstellt wurden. Sobald dies erledigt ist, wird der Pfad in `prepare_data` übergeben und der Funktion eine Variable zugewiesen. Die Daten werden in Trainings- und Validierungsdaten aufgeteilt – in diesem Fall im Verhältnis 80/20, wobei 80% für das Training und 20% für die Validierung verwendet werden.

Diese Aufteilung hat sich in der Praxis als effektiv erwiesen, da sie einerseits genügend Trainingsdaten für das Modell bereitstellt und andererseits ausreichend Validierungsdaten zur Beurteilung der Modellleistung auf unbekannten Daten gewährleistet. Insbesondere bei kleineren Datensätzen sorgt diese Aufteilung dafür, dass das Modell optimal trainiert wird, ohne dass die Validierung vernachlässigt wird. Eine ausreichende Menge an Trainingsdaten ist entscheidend für die Robustheit des Modells, während repräsentative Validierungsdaten erforderlich sind, um die Generalisierungsfähigkeit sicherzustellen. Die gewählte 80/20-Aufteilung erfüllt beide Anforderungen (Géron, 2022; Goodfellow, 2016).

Zudem werden die Zellgröße der Bildchips, die standardmäßig 224×224 Pixel beträgt, sowie weitere Transformationen bzw. Augmentationen definiert. Letzteres wird durch den bereits erwähnten `transforms`-Parameter festgelegt. Weiters wird die Batchgröße definiert, die unter Berücksichtigung der Kapazität der Grafikkarte, der Modellgröße und der Größe der Daten geschätzt werden muss. Zu Beginn kann eine kleine Batchgröße, wie z. B. 4, gewählt werden, bis mit der Funktion `arcgis.learn.estimate_batch_size()` im späteren Verlauf die empfohlene Batchgröße genauer abgeschätzt werden kann. Bevor diese Funktion ausgeführt werden kann, muss jedoch das Modell definiert werden, da die Funktion die Modelldefinition benötigt. Weitere Details gibt es dazu in Kapitel 6.7.

6.6.4 Datenaugmentationsmethoden in `prepare_data`

Mit dem Parameter `transforms` von `arcgis.learn` können, zusätzlich zu der bereits durchgeführten Transformation mithilfe der Rotation, weitere Augmentationsmethoden angewendet werden, um den Trainingsdaten mehr Variabilität zu verleihen und das Modell robuster gegenüber Abwandlungen in den Daten zu machen. Bevor man die Transformationen an `prepare_data` übergibt, muss jedoch das notwendige Paket importiert werden. Hierzu wird auf andere Bibliotheken zurückgegriffen. Die Wahl fiel hierbei auf die Funktion `get_transforms()` der Open-Source-Bibliothek `fastai`. Sie basiert auf `PyTorch` und bietet einfache sowie effiziente Lösungen für Aufgaben im Bereich des Machine und Deep Learning (mit ihrer API). Unter anderem eignet sich die Bibliothek auch gut für Datenaugmentation (fastai, 2024).

Im nächsten Schritt wird eine Prozesskette (auch Pipeline genannt) für den Trainingsdatensatz erstellt. Dies bedeutet, dass die Trainingsbilder eine Reihe vordefinierter Transformationen durchlaufen, um die

Generalisierungsfähigkeit des Modells zu optimieren. Gleichzeitig wird die Augmentation nicht auf den Validierungsdatensatz angewendet, da dieser unverändert bleiben sollte, da er reale Daten widerspiegelt. Die ausgewählten Transformationsmethoden bzw. Parameter sind `flip_vert` und `max_lighting`. Die erste Methode ermöglicht einen vertikalen Flip, also ein Spiegeln der Bilder, was man nicht mit einer Rotation verwechseln darf. Diese Option ist in diesem Fall sinnvoll, da die Solaranlagen trotz der Spiegelung im Bildmaterial realistisch bleiben. Mit `max_lighting` kann eine beliebige Licht- und Kontraständerung auf den Bildchips angewendet werden. Es wurde der Wert 0.4 als Argument gewählt, da ein zu kleiner oder zu großer Wert die Wahrscheinlichkeit erhöht, dass die Photovoltaik- und Solarthermieanlagen schlecht erlernt werden können (fastai, 2023). Das Modell soll lernen, dieselben Objekte unter leicht unterschiedlichen Lichtverhältnissen zu klassifizieren und zu erkennen.

Da die Augmentation global auf den gesamten Datensatz angewendet wird, werden keine gesonderten Bilder in den bestehenden Verzeichnissen der Trainingsdaten abgelegt. Daher wird anhand eigener Bildbeispiele visuell demonstriert, wie sich die beiden Methoden auf die Daten auswirken:

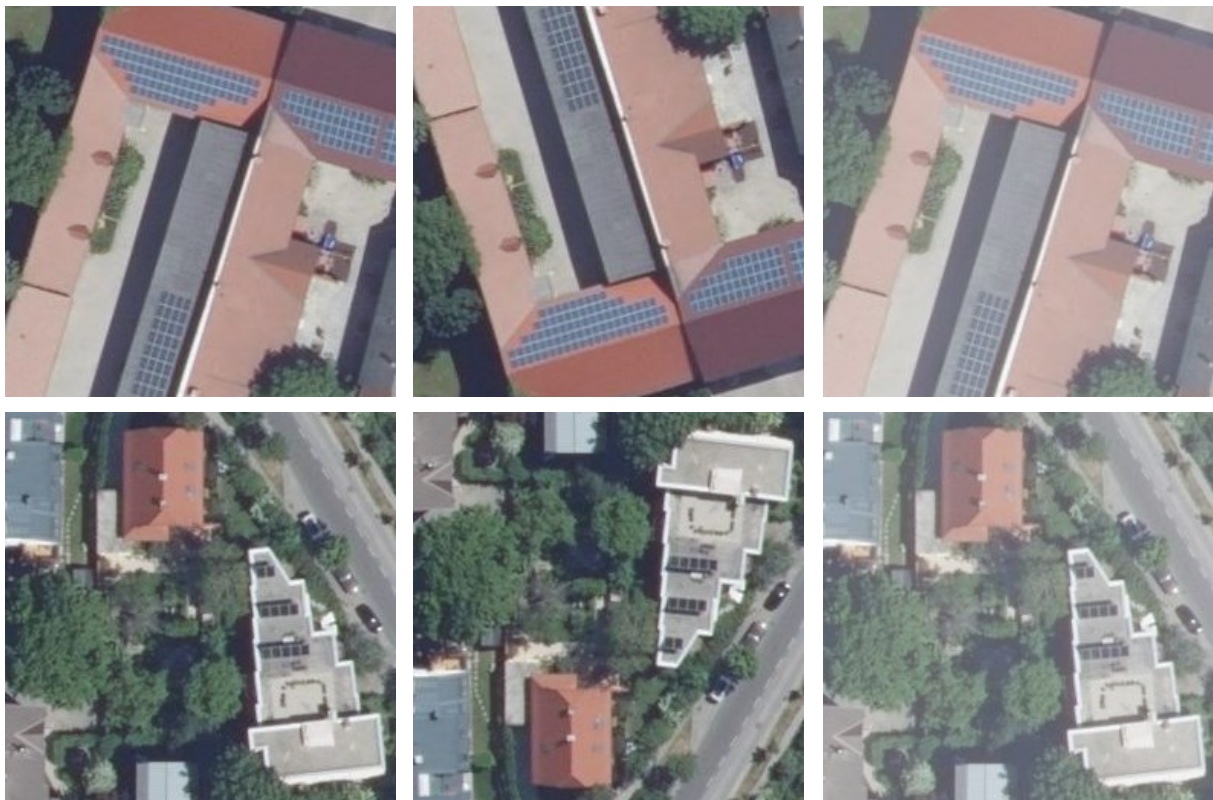


Abb. 44: Originalbild (oben links, unten links), Vertikaler Flip (oben mittig, unten mittig) und Helligkeitsanpassung (oben rechts, unten rechts)

(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2 und IrfanView 4.53)

6.7 Training der Daten

Das Training beider Datensätze (Datensatz 1 und kombinierter Datensatz) wird stets getrennt für die jeweiligen Modelle durchgeführt. Auch innerhalb der beiden Datensätze wird unterschieden, beispielsweise in Bezug darauf, mit wie vielen Epochen trainiert wird und ob weitere Datenaugmentationen mittels `prepare_data` angewendet wurden. Es wurden zwei Jupyter Notebook-Skripte angehängt, die das Vorbereiten und Trainieren von zwei der insgesamt vier Modelle beispielhaft

darstellen (siehe Anhang C und D). Die Skripte wurden u. a. mithilfe der Bibliothek **arcgis.learn** geschrieben (Esri Developer, 2023a). Die folgende Tabelle listet die Formate der verwendeten Metadaten (Bildchips, Annotationen) für die Modelle auf:

Modell	Format der Bildchips	Format der Annotationen
Faster R-CNN	TIFF	XML
Mask R-CNN	TIFF	TIFF

Tab. 12: Übersicht der Datenformate für Trainingsdaten

6.7.1 Faster R-CNN – Datensatz 1

In Kapitel 6.6.3 wurde bereits erwähnt, dass Modellklassen aus dem **arcgis.learn**-Modul in die Arbeitsumgebung importiert werden müssen. In diesem Fall handelt es sich um die Klasse **FasterRCNN**, die ein Faster R-CNN-Modell initialisiert. Es wird eine Variable erstellt, die das definierte Modell speichert. Zur Erinnerung: Faster R-CNN führt als Objekterkennungsmodell sowohl die Lokalisierung (Bounding Boxes) als auch die Klassifizierung von Objekten in Bilddaten durch.

Die vorbereiteten Trainings- und Validierungsdaten mit den Bildern und Annotationen werden an das Modell übergeben. Zudem wird ein Backbone-Netzwerk festgelegt, das die Extraktion von Merkmalen verantwortlich ist. Die Wahl des Backbones bestimmt, wie tief das Modell lernen soll, also wie viele Schichten (engl.: Layer) das Netzwerk besitzt. Je tiefer das Netz, desto genauer lernt das Modell, benötigt jedoch auch deutlich mehr Speicherbedarf (insbesondere auf der GPU). Für die **FasterRCNN**-Variable wurde anfänglich **ResNet-50** ausgewählt. Anschließend wurde die Funktion **arcgis.learn.estimate_batch_size()** ausgeführt, um zu überprüfen, welche Batchgröße für das Modell unter Berücksichtigung der Rechnereigenschaften geeignet ist. Das Ergebnis zeigt, dass 32 Datenpunkte (Bilder) pro Batch verarbeitet werden können, ehe das Modell die Gewichte auf den neusten Stand bringt.

Mithilfe der Funktion **<prepare_data-Variable>.show_batch()** können stichprobenartig die Trainings- und Validierungsdaten sowie deren Aufteilung visualisiert werden. Diese Überprüfung dient dazu, sicherzustellen, dass die Annotationen, z. B. in Bezug auf ihre Richtigkeit, sinnvoll erscheinen. Als Ergebnis wird die Anzahl der Annotationen für Training und Validierung ausgegeben. Die Funktion **<Modelldefinition>.show_results()** zeigt die Modellergebnisse auf einer Auswahl von Daten (Eingabebilder, Bounding Boxes und Klassenzuweisungen) aus dem Validierungsdatensatz an. Sie ermöglicht es, die Leistung des Modells auf diesen Daten zu bewerten, indem die Ground Truth-Annotationen und die Vorhersagen nebeneinander angezeigt werden (siehe Abbildung 45).

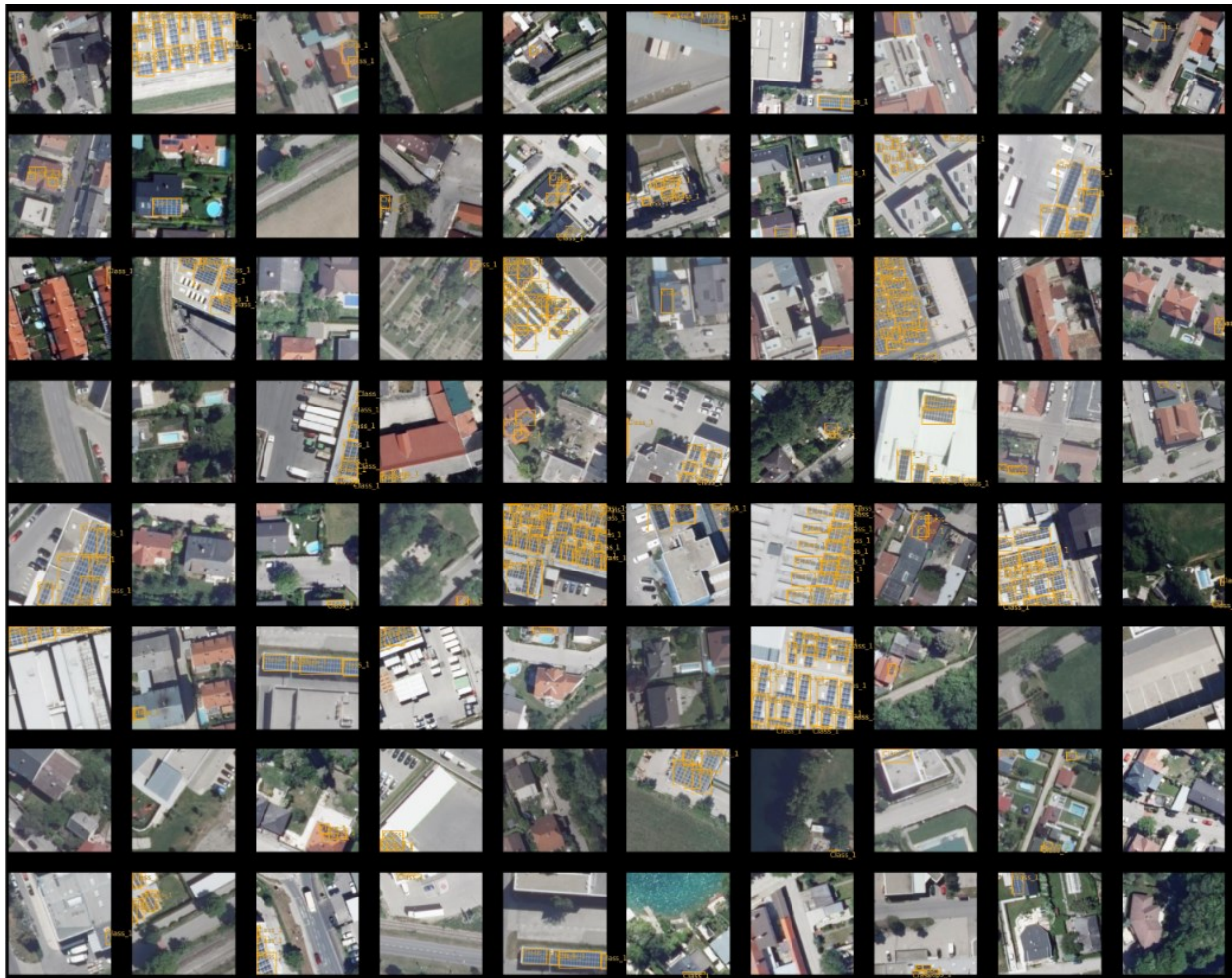
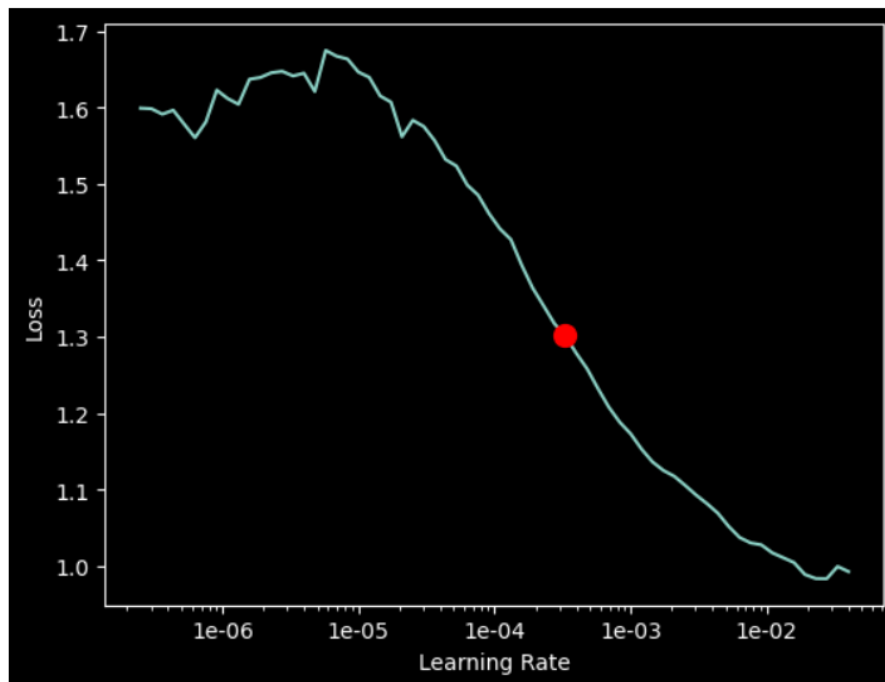


Abb. 45: Stichprobenartige Anzeige der Annotationen auf den Bilddaten in Datensatz 1
(FasterRCNN, 10 Epochen, ohne Transformationen)
 (Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Mit der Funktion `<Modelldefinition>.lr_find()` kann die optimale Lernrate für das Training ermittelt werden, indem ein kurzer Trainingsdurchlauf mit verschiedenen Lernraten durchgeführt wird. Dabei wird der Verlust (engl.: Loss) im Training für jede einzelne Lernrate protokolliert. In der untenstehenden Abbildung ist ein Diagramm bzw. eine Lernratenkurve zu sehen, die den Verlust in Abhängigkeit von der Lernrate darstellt. Es gibt ein empfohlenes Intervall zurück, das als Eingabewert für das Training genutzt werden kann. Die empfohlene Lernrate ist der Punkt im Diagramm, an dem der Verlust stark abnimmt, bevor ein Anstieg erkennbar ist. Sie bestimmt, in welchem Maße das Modell seine Gewichte nach jedem Durchgang (bzw. Trainingseinheit) anpasst (siehe Abbildung 46).

In einem weiteren Schritt wird dieses Modell mit der Methode bzw. der Funktion `<Modelldefinition>.fit()` auf den bereitgestellten Trainingsdaten trainiert. Zuvor können alle für das Modell verfügbaren Metriken als Liste mit `<Modelldefinition>.available_metrics` angezeigt werden. Diese Metriken können zur Bewertung des Modells während der Trainings- und Validierungsphase herangezogen werden. Im Falle von FasterRCNN sind `valid_loss` und `average_precision` verfügbar.



0.0003311311214825911

Abb. 46: Lernratenkurve von FasterRCNN in Datensatz 1 (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Die Parameter, die hier festgelegt werden müssen, sind die Anzahl der Epochen, die Lernrate und das Keyword-Argument `early_stopping=True`. Diese Technik hilft, Overfitting zu verhindern, indem das Training abgebrochen wird, bevor die angegebene Anzahl von Epochen erreicht ist. Dies geschieht, wenn der Validierungsverlust über mehrere aufeinanderfolgende Epochen hinweg nicht mehr abnimmt. Dadurch wird auch Zeit und Ressourcen gespart. In einer Trainingsepoche wird der komplette Datensatz genau einmal durch das Modell verarbeitet. Anfangs eignen sich hier zehn Epochen, bevor die Anzahl bei Bedarf später angepasst werden kann. Bei der Wahl der Lernrate, die die Geschwindigkeit des Lernprozesses steuert, kann das zuvor durch `<Modelldefinition>.lr_find()` ermittelte Lernrate-Intervall verwendet werden. Als Startwerte haben sich jedoch auch 0.01 oder 0.001 als empfehlenswert erwiesen, da sie in der Praxis eine gute Balance zwischen der Stabilität des Lernprozesses und einer ausreichenden Anpassung der Modellparameter bieten (Surmenok, 2017).

Das FasterRCNN-Modell initialisiert die Modellgewichte basierend auf dem gewählten Backbone. Die Trainings- und Validierungsdatsätze werden in Batches geladen, deren Größe zuvor in der Funktion `prepare_data()` festgelegt wurde. In jeder Epoche berechnet das Modell Vorhersagen für die Daten, der entsprechende Verlust wird ermittelt, und die Modellgewichte werden durch Backpropagation optimiert. Um die Leistung des FasterRCNN-Modells auf nicht gesehenen (Validierungs-)Daten zu bewerten, wird nach Beendigung jeder Epoche der Validierungsverlust berechnet.

Das Ziel ist es, die Modellparameter so zu optimieren, dass eine möglichst gute Leistung auf den Trainings- und Validierungsdaten erzielt wird.

6.7.1.1 Faster R-CNN: Training über 10 Epochen ohne Transformationen

Das Ergebnis zeigt neben der Epochenummer sowie der für eine Epoche benötigten Trainingszeit den `train_loss` (Verlust auf dem Trainingsdatensatz), `valid_loss` (Verlust auf dem Validierungsdatensatz) und die `average_precision` (Average Precision, AP) pro Trainingsepoche.

epoch	train_loss	valid_loss	average_precision	time
0	0.545245	0.548575	0.460234	25:39
1	0.477188	0.543680	0.515759	25:36
2	0.486184	0.541264	0.611484	25:37
3	0.472412	0.552094	0.548210	25:31
4	0.456946	0.545391	0.553235	25:31
5	0.445629	0.514889	0.564217	25:31
6	0.438566	0.512592	0.596478	25:27
7	0.432144	0.507440	0.563616	25:37
8	0.416748	0.503577	0.602435	25:34
9	0.422484	0.499887	0.600680	25:32

Abb. 47: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Die Dauer einer Epoche bleibt konstant bei knapp 25 Minuten und 30 Sekunden, was zeigt, dass keine unerwarteten Änderungen in der Verarbeitung der Daten oder den Berechnungen aufgetreten sind. Der Trainingsverlust sinkt von 0.5452 in Epoche 0 auf 0.4225 in Epoche 9. Ein stetig sinkender Trainingsverlust ist ein Zeichen dafür, dass das Modell die Trainingsdaten immer besser anpasst. Auch der Validierungsverlust fällt kontinuierlich von 0.5486 in Epoche 0 auf 0.4999 in der letzten Epoche, was darauf hindeutet, dass das FasterRCNN-Modell nicht nur die Trainingsdaten, sondern auch neue Daten immer besser verarbeitet.

Die Metrik **Average Precision** bewertet die durchschnittliche Genauigkeit des Modells bei der Vorhersage von Objekten. Sie steigt von 0.4602 in der ersten Epoche auf 0.6007 in Epoche 9. Ein steigender Wert deutet auf eine Verbesserung der Modellleistung hin, das Modell verbessert im Laufe des Trainings die Genauigkeit der Objekterkennung.

Zusammenfassend lässt sich eine Verbesserung in allen wichtigen Kennzahlen, wie ein sinkender Verlust und eine steigende Präzision, erkennen, was auf ein erfolgreiches Training hinweist. Die steigende AP zeigt, dass das Modell im Laufe der Trainingsepochen die Photovoltaikanlagen immer besser erkennt. Der Unterschied zwischen Trainingsverlust und Validierungsverlust ist moderat, was bedeutet, dass das Modell noch nicht zu stark an die Trainingsdaten angepasst wurde und eine gute Balance zwischen Training (Anpassung) und Validierung (Generalisierung) besteht. Konkret bedeutet das, dass das Modell bisher nicht überanpasst (Overfitting) wurde. Dies wäre dann der Fall, wenn das Modell die Trainingsdaten positiv feinjustiert (sehr niedriger Trainingsverlust), aber bei unbekannten Daten schlecht generalisiert (hoher Validierungsverlust). Ein Modell sollte nicht nur bei den Trainingsdaten gute Ergebnisse liefern, sondern auch bei neuen, nicht gesehenen Informationen.

Für mehr Einblicke in die Modellleistung bzw. den Lernverlauf kann mithilfe von `<Modelldefinition>.learn.recorder.plot_losses()` ein Verlustdiagramm (engl.: Loss Plot) erstellt werden, das die Verlustkurven für den Trainings- und Validierungsdatensatz über alle Epochen hinweg zeigt. Konkret geht es hier um den Trainings- und Validierungsverlust, die bereits erklärt wurden.

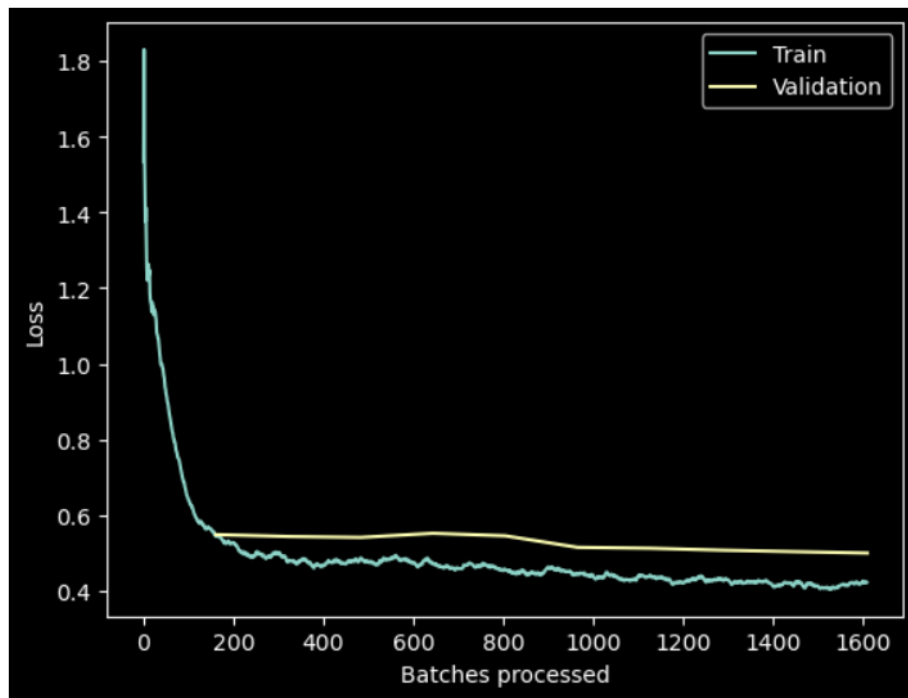


Abb. 48: Verlustdiagramm von FasterRCNN von Datensatz 1I (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Die türkisfarbene Kurve stellt den Verlust während des Trainings dar, während die gelbe Kurve den Validierungsverlust anzeigt.

- Zu Beginn ist ein starker Abfall des Trainingsverlustes zu sehen, die Kurve stabilisiert sich jedoch anschließend.
- Das Modell lernt anfangs sehr schnell, da seine Gewichte noch nicht optimiert wurden.
- Der Verlust auf den Validierungsdaten bleibt höher als der Trainingsverlust. Dies ist aber die Norm und deutet nicht auf Overfitting hin, da die Differenz moderat bleibt.
- Beide Kurven flachen zum Ende hin ab, was darauf hindeutet, dass das FasterRCNN-Modell einen Punkt erreicht hat, an dem weitere Optimierungen der Modellparameter möglicherweise nur minimale bis gar keine signifikanten Auswirkungen auf die Ergebnisse haben.

Die Endwerte des Verlustes deuten auf brauchbare Modellergebnisse hin. Das Modell zeigt keine Anzeichen von Overfitting oder Underfitting und trainiert gut. Potenziell kann das Modell jedoch von Feinanpassungen der Hyperparameter, wie einer anderen Lernrate oder zusätzlichen Epochen, profitieren.

Nach dem Training können abschließend mit der Funktion `<Modelldefinition>.show_results()` die Modellergebnisse sowohl für den Trainings- als auch für den Validierungsdatensatz visualisiert werden. Die Ground Truth-Daten stellen die tatsächlichen Annotationen dar, während die Vorhersagen die vom Modell erkannten Photovoltaikanlagen und deren Klassifikation zeigen (siehe Abbildung 49).

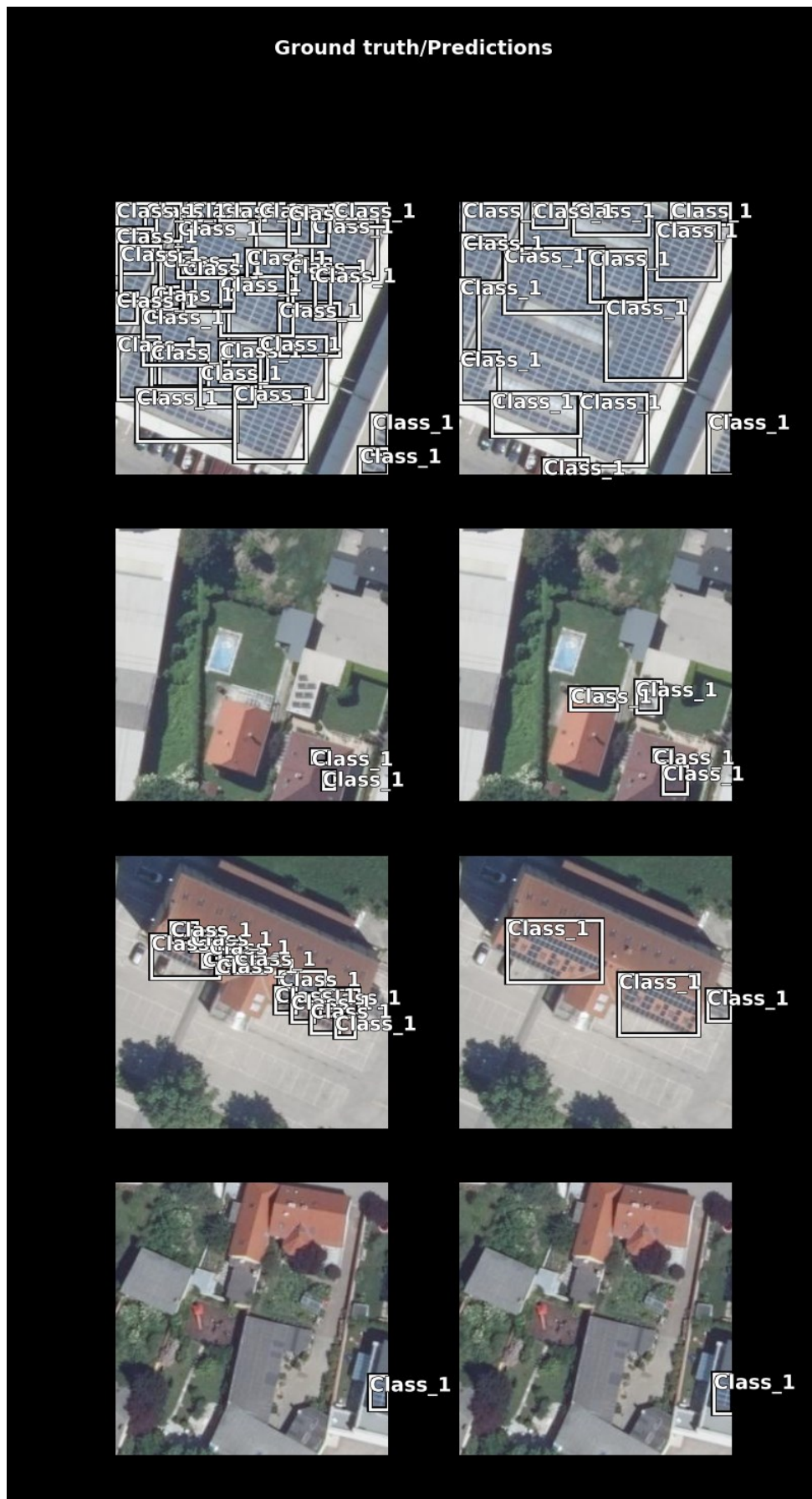


Abb. 49: Funktion `show_results()` für FasterRCNN mit Ground Truth (l.) und Vorhersagen (r.)
(Datensatz 1, 10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Zu guter Letzt muss das Modell mit der Funktion `<Modelldefinition>.save()` an einem Speicherort gesichert werden, sodass es zu einem späteren Zeitpunkt verwendet werden kann, ohne dass der Trainingsprozess erneut durchgeführt werden muss. Dort wird u. a. die EMD-Datei des Modells gespeichert, die im weiteren Verlauf für die Erkennung der Objekte benötigt wird.

6.7.1.2 Hyperparameter-Tuning: Training über 10 Epochen inkl. Transformationen

In einem weiteren Versuch wurden in der `prepare_data()`-Funktion zusätzliche Transformationen definiert. Hier wurde erneut mit zehn Epochen begonnen, bevor die Anzahl erhöht wurde. Pro Datensatz und Modelltyp wurden die Transformationen jeweils einmal angewendet, mit dem Ziel, durch erweiterte Datenvielfalt Verbesserungen in den Modellvorhersagen zu erzielen, gegenüber den Versuchen ohne zusätzliche Augmentationsmethoden.

epoch	train_loss	valid_loss	average_precision	time
0	0.686667	0.553793	0.480640	23:33
1	0.571518	0.553531	0.530103	23:55
2	0.569137	0.552474	0.588876	24:09
3	0.544824	0.532362	0.579928	24:06
4	0.527137	0.558119	0.594353	24:08
5	0.522734	0.515854	0.619496	24:07
6	0.515653	0.520656	0.614655	24:08
7	0.503282	0.517097	0.637580	24:12
8	0.502584	0.506273	0.620025	24:10
9	0.498088	0.506301	0.629529	24:08

Abb. 50: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Die Rechenzeit für eine Trainingsepoche hat sich trotz der erweiterten Datenvielfalt minimal auf knapp 24 Minuten reduziert, ohne größere Abweichungen. Die geringe Differenz zwischen Trainings- und Validierungsverlust deutet darauf hin, dass das Modell weiterhin kein Overfitting aufweist. Die Average Precision zeigt, dass die durchschnittliche Genauigkeit von rund 0.4806 in Epoche 0 auf etwa 0.6295 in Epoche 9 steigt. Mit zunehmender Epochenanzahl lernt und verbessert sich die Leistung des Modells kontinuierlich.

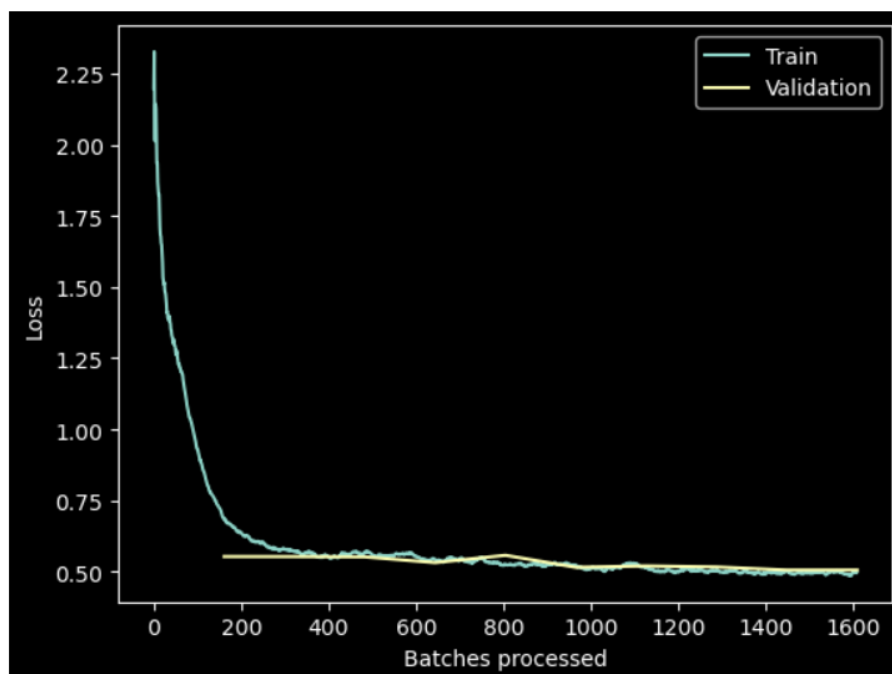


Abb. 51: Verlustdiagramm von FasterRCNN von Datensatz 1 (10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Die gelbe Kurve (Validierungsverlust) und die türkisfarbene Kurve (Trainingsverlust) verlaufen nahezu parallel, was ein Zeichen für eine stabile Generalisierung des Modells ist.
- Das Modell trainiert und lernt stabil, ohne Anzeichen von Overfitting oder Underfitting.
- Weitere Verbesserungen in den Vorhersagen könnten durch zusätzliche Daten oder Feinjustierung der Hyperparameter, wie die Anpassung der Lernrate oder den Einsatz eines stärkeren Backbones, erreicht werden.

6.7.1.3 Hyperparameter-Tuning: Training über 10 Epochen inkl. Transformationen und Lernrate von 0.001

Zusätzlich zu den 10 Epochen und Transformationen wurde in einem weiteren Trainingsversuch eine andere Lernrate definiert als die mittels der `<Modelldefinition>.fit()`-Funktion berechnete.

epoch	train_loss	valid_loss	average_precision	time
0	0.596472	0.586192	0.599179	23:52
1	0.578552	0.602676	0.522225	18:43
2	0.577785	0.571002	0.541890	20:49
3	0.567635	0.560061	0.560027	26:58
4	0.552910	0.557044	0.603333	27:45
5	0.545382	0.561071	0.563968	27:57
6	0.524997	0.526934	0.569796	28:54
7	0.504350	0.525538	0.618731	28:02
8	0.500932	0.518519	0.618952	29:01
9	0.494400	0.519256	0.618821	28:04

Abb. 52: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen, Lernrate)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Die Trainingszeit pro Epoche ist leicht schwankend. Das Maximum beträgt 29 Minuten. Der Trainingsverlust sinkt kontinuierlich von 0.5965 auf 0.4944, während der Validierungsverlust bei 0.5862 startet, leicht schwankt und schließlich bei 0.5193 endet. Der Validierungsverlust bleibt eng am Trainingsverlust, was zeigt, dass das Modell nicht stark überanpasst. Beide Verluste sinken kontinuierlich, was auf eine gut funktionierende Optimierung hindeutet. Die Average Precision beginnt bei 0.5992, fällt nach Epoche 1 auf 0.5222, steigt dann aber wieder an und endet schließlich bei 0.6188. Die leichten Schwankungen bedeuten, dass sich die Modellleistung verbessert, allerdings nicht durchgehend linear.

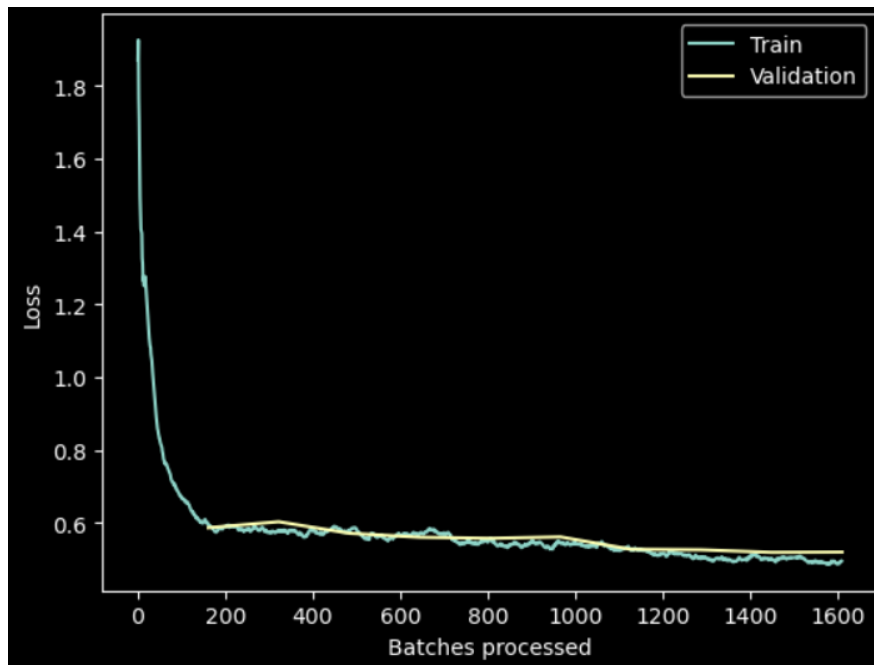


Abb. 53: Verlustdiagramm von FasterRCNN von Datensatz 1 (30 Epochen, ohne Transformationen, Lernrate)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Der Verlust auf den Trainings- (türkisfarbene Kurve) und Validierungsdaten (gelbe Kurve) zeigt auch hier einen starken initialen Abfall in den ersten 200 Batches, was typisch für gut gewählte Lernraten ist.
- Die Kurven verlaufen relativ parallel, ohne dass die Validierungskurve ab einem bestimmten Punkt ansteigt – ein Anzeichen dafür, dass kein Overfitting auftritt.
- Nach etwa 1000 Batches stabilisiert sich der Verlust, was darauf hindeutet, dass das Modell ein Minimum erreicht hat.

Die gewählte Lernrate von 0.001 scheint gut zu funktionieren, da der Verlust stabil bleibt. Eine zu hohe Lernrate könnte bewirken, dass das Modell an einem suboptimalen Punkt stagniert oder gar schlechtere Ergebnisse liefert. Die gewählte Lernrate scheint eine bessere Stabilität zu bieten.

6.7.1.4 Hyperparameter-Tuning für Faster R-CNN: Training über 30 Epochen ohne Transformationen
Um möglicherweise bessere Ergebnisse zu erzielen, wurde zunächst als weitere Anpassungsstrategie getestet, ob eine Erhöhung der Epochenanzahl sich positiv auswirkt. Daher wurden dieses Mal bei der Funktion `<Modelldefinition>.fit()` 30 Trainingsepochen definiert. Nach knapp über zwölf Stunden Rechenzeit ergibt sich folgendes Ergebnis:

epoch	train_loss	valid_loss	average_precision	time
0	0.580311	0.568441	0.357500	25:51
1	0.478139	0.550432	0.511337	25:38
2	0.471161	0.532980	0.527397	25:39
3	0.457596	0.526206	0.551761	25:40
4	0.461387	0.542761	0.546594	25:40
5	0.453673	0.609007	0.468803	26:14
6	0.459598	0.541554	0.602138	28:21
7	0.442680	0.528333	0.569488	25:25
8	0.444925	0.522379	0.555157	25:13
9	0.446927	0.538667	0.563110	25:12
10	0.434340	0.520658	0.559720	25:12
11	0.429882	0.534145	0.617527	25:09
12	0.429593	0.513425	0.570151	25:10
13	0.424701	0.503373	0.602816	25:15
14	0.417771	0.501456	0.592178	25:11
15	0.410488	0.502427	0.598970	25:07
16	0.410832	0.495736	0.592592	25:11
17	0.406821	0.508679	0.578045	25:15
18	0.404332	0.489188	0.637932	25:14
19	0.400331	0.489411	0.612479	25:13
20	0.403546	0.471830	0.615007	25:12
21	0.407720	0.483171	0.633632	25:11
22	0.393414	0.491120	0.588758	25:12
23	0.387292	0.474770	0.646949	25:12
24	0.381119	0.474725	0.635432	25:12
25	0.389338	0.468950	0.634830	25:09
26	0.381216	0.471356	0.646064	25:10
27	0.390351	0.469474	0.640070	25:09
28	0.382560	0.469567	0.637662	25:08
29	0.379442	0.468416	0.638645	25:08

Abb. 54: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (30 Epochen, ohne Transformationen)
 (Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Der Verlust beim Trainingsdatensatz nimmt über die Epochen hinweg beständig ab, was zeigt, dass das Modell während des Trainings die Fehler reduziert und besser an die Trainingsdaten angepasst wird. Ab etwa Epoche 24 stabilisiert sich der train_loss bei etwa 0.3795, was auf eine konsistente Anpassung des Modells hindeutet. Auch der Validierungsverlust deutet auf eine Verbesserung hin und klingt im Verlauf der Epochen allmählich ab. Es gibt kein Overfitting, da weder erhebliche Anstiege noch Schwankungen in den Verlustwerten auftreten. Die Average Precision steigt ebenfalls laufend an und stabilisiert sich gegen Ende bei rund 0.6386 (im Gegensatz zu 0.3575 in Epoche 0). Dies unterstreicht, dass das FasterRCNN-Modell im Laufe des Trainings immer präzisere Vorhersagen trifft.

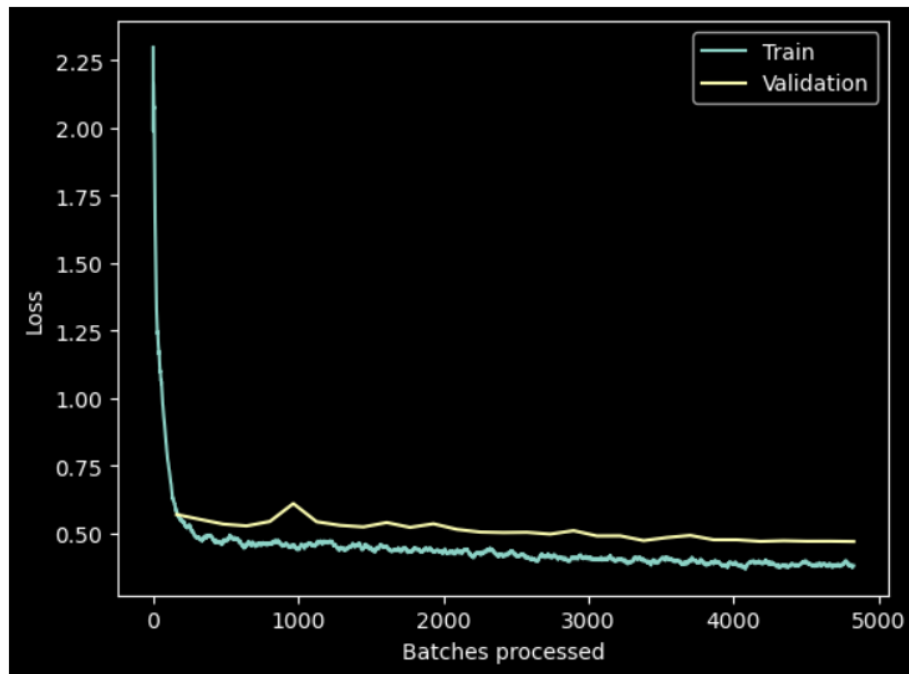


Abb. 55: Verlustdiagramm von FasterRCNN von Datensatz 1 (30 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Der Trainingsverlust (türkisfarbene Kurve) nimmt zu Beginn rasch ab und stabilisiert sich anschließend. Das ist typisch, da das Modell anfangs größere Anpassungen vornimmt, die später feiner und gezielter werden.
- Der Verlust auf dem Validierungsdatensatz (gelbe Kurve) stellt eine ähnliche Entwicklung dar, bleibt aber erwartungsgemäß etwas höher als der Trainingsverlust.
- Insgesamt bleibt die Differenz zwischen Trainings- und Validierungsverlust gering, was darauf hinweist, dass das Modell gut generalisiert.
- Es gibt keine Anzeichen für Overfitting, da der Validierungsverlust nicht signifikant ansteigt.

Da der Verlust auf den Validierungsdaten nicht stark abnimmt und die AP zum Ende hin stagniert, könnten weitere Optimierungen der Hyperparameter sinnvoll sein, um ein potenzielles Underfitting zu vermeiden. Eine mögliche Strategie wäre, die Lernrate zu erhöhen oder z. B. ein leistungsstärkeres Backbone zu verwenden. Jedoch wurde zunächst der Ansatz verfolgt, durch zusätzliche Augmentation die Datenvielfalt zu erhöhen.

Die Methode `<Modelldefinition>.show_results()` zeigt kaum Unterschiede im Vergleich zum ersten Trainingsversuch mit zehn Epochen, besonders hinsichtlich des Vergleichs zwischen Ground Truth-Daten und Modellvorhersagen.

6.7.1.5 Hyperparameter-Tuning für Faster R-CNN: Training über 30 Epochen inkl. Transformationen
Die Funktion `<Modelldefinition>.fit()` wurde mit der zuvor ermittelten Lernrate für 30 Epochen ausgeführt, mit den folgenden Ergebnissen:

epoch	train_loss	valid_loss	average_precision	time
0	0.662046	0.559411	0.451787	25:40
1	0.568556	0.540674	0.564077	25:25
2	0.557751	0.553295	0.583489	25:19
3	0.544827	0.552729	0.576085	25:21
4	0.539233	0.529603	0.565831	25:19
5	0.531412	0.547257	0.595927	25:21
6	0.536618	0.618251	0.629185	25:23
7	0.526079	0.546938	0.563760	25:23
8	0.532452	0.535020	0.631567	25:24
9	0.532031	0.558939	0.533149	25:23
10	0.509903	0.580151	0.511762	25:23

Epoch 10: early stopping

Abb. 56: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (30 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Der Trainingsverlust zeigt einen kontinuierlichen Rückgang, was auf eine gute Anpassung des Modells an die Trainingsdaten deutet. Der Verlust auf den Validierungsdaten hingegen weist nach Epoche 5 keine klare Verbesserung mehr auf, sondern zeigt Schwankungen und bleibt insgesamt auf einem höheren Niveau. Das könnte ein Hinweis auf ein leichtes Overfitting sein, da das Modell beginnt, die Trainingsdaten zu stark zu lernen. Die Average Precision steigt bis Epoche 6 auf einen Höchstwert von 0.6292 an, fällt danach jedoch wieder ab. Diese Verhaltensweise deutet darauf hin, dass dieses Modell ab einem bestimmten Punkt keine weiteren signifikanten Optimierungen erzielt und die Leistung auf dem Validierungsdatensatz sogar abnimmt.

Angesichts dieser Entwicklung wurde das Training frühzeitig bei Epoche 10 angehalten, also nach insgesamt elf Epochen. Der Validierungsverlust zeigt keine weiteren Verbesserungen, da das Modell mit den aktuellen Parametern und der verfügbaren Datenbasis seine Leistungsgrenze erreicht hat.

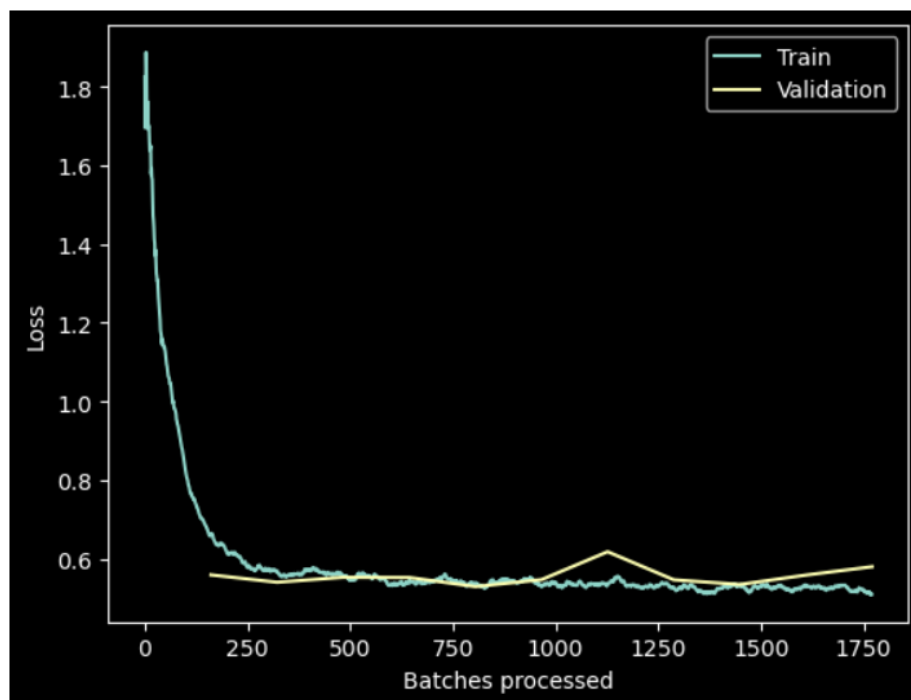


Abb. 57: Verlustdiagramm von FasterRCNN von Datensatz 1 (30 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Die türkisfarbene Kurve (Trainingsverlust) zeigt einen stetigen Rückgang und stabilisiert sich bei einem relativ niedrigen Wert.
- Die gelbe Kurve (Validierungsverlust) verläuft zu Beginn synchron mit dem Trainingsverlust, zeigt aber später Schwankungen. Insbesondere zwischen den Batches 1000 und 1250 ist ein signifikanter Anstieg zu erkennen. Gegen Ende des Trainings steigt der Validierungsverlust erneut an.
- Die Fluktuationen im Validierungsverlust zeigen auf, dass das Modell Schwierigkeiten hat, auf den Validierungsdaten konsistent zu performen. Dies ist ein möglicher Hinweis auf leichtes Overfitting, da die Trainingsdaten besser gelernt werden, während bei der Generalisierung keine Verbesserungen mehr erfolgen.

Trotz der genannten Indikatoren für Overfitting ist es nicht stark ausgeprägt, da die Verluste insgesamt nah beieinanderliegen. Die eingesetzte Augmentation scheint eine bessere Generalisierung des Modells in den ersten Epochen zu unterstützen, wie der Anstieg der AP zu Beginn des Trainings zeigt. Allerdings könnte die verfügbare Datenmenge oder deren Qualität limitiert sein, was die langfristige Modellleistung beeinflusst. Die ermittelte Lernrate von 3.31×10^{-4} bzw. 0.000331 erscheint für die ersten Epochen passend, könnte jedoch für ein längeres Training suboptimal sein. Um die Stabilität des Trainings zu verbessern, wird zunächst ein weiterer Trainingslauf mit einer höheren Lernrate durchgeführt. Das könnte dazu beitragen, die Modellentwicklung auch in späteren Epochen zu unterstützen und Schwankungen im Validierungsverlust zu minimieren.

6.7.1.6 Hyperparameter-Tuning für Faster R-CNN: Training über 30 Epochen inkl. Transformationen und Lernrate von 0.001

epoch	train_loss	valid_loss	average_precision	time
0	0.608608	0.551955	0.571164	21:57
1	0.557211	0.551970	0.549896	25:17
2	0.558707	0.563278	0.595841	26:20
3	0.541944	0.586559	0.574339	27:11
4	0.552016	0.568130	0.551939	34:12
5	0.564859	0.577310	0.546272	32:42
6	0.549361	0.550843	0.551650	25:43
7	0.540192	0.563988	0.557409	26:00
8	0.537418	0.556871	0.558293	25:55
9	0.542185	0.581013	0.567702	26:02
10	0.527303	0.556563	0.658795	26:07
11	0.521070	0.532438	0.586660	25:51
12	0.522805	0.544082	0.623118	26:00
13	0.515047	0.531546	0.596114	25:51
14	0.504921	0.522608	0.634157	25:59
15	0.502701	0.534062	0.645367	26:05
16	0.505724	0.550708	0.609970	26:10
17	0.497927	0.537245	0.636663	26:00
18	0.495239	0.519834	0.607887	25:57
19	0.496726	0.523577	0.642648	26:04
20	0.495112	0.513026	0.614368	26:04
21	0.480471	0.493496	0.655227	26:07
22	0.479180	0.520428	0.623534	26:10
23	0.478737	0.497328	0.639892	26:05
24	0.478809	0.495565	0.653833	26:06
25	0.467835	0.484399	0.636914	26:04
26	0.464995	0.488771	0.654792	26:07
27	0.465137	0.487476	0.654066	26:10
28	0.464133	0.489735	0.654019	26:06
29	0.470504	0.488740	0.651421	26:05

Abb. 58: FasterRCNN-Modellergebnis nach Training mit Datensatz 1 (Lernrate 0.001, 30 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Der Verlust auf den Trainingsdaten bleibt am Ende des Trainingslaufes etwa im Bereich von rund 0.46 bis 0.49, was auf eine stabile Anpassung des Modells an die Trainingsdaten hinweist. Der Validierungsverlust folgt diesem Trend und zeigt ein ähnliches Verhalten, dieser liegt aber leicht höher als der Trainingsverlust. Nach 30 Epochen beträgt dieser 0.4887 und der Trainingsverlust beträgt 0.4705. Die AP steigt kontinuierlich an und erreicht in späteren Epochen einen Höchstwert von etwa 0.6588 (in Epoche 10). Nach einer kurzen Abnahme stabilisiert sich die AP im Bereich von etwa 0.65.

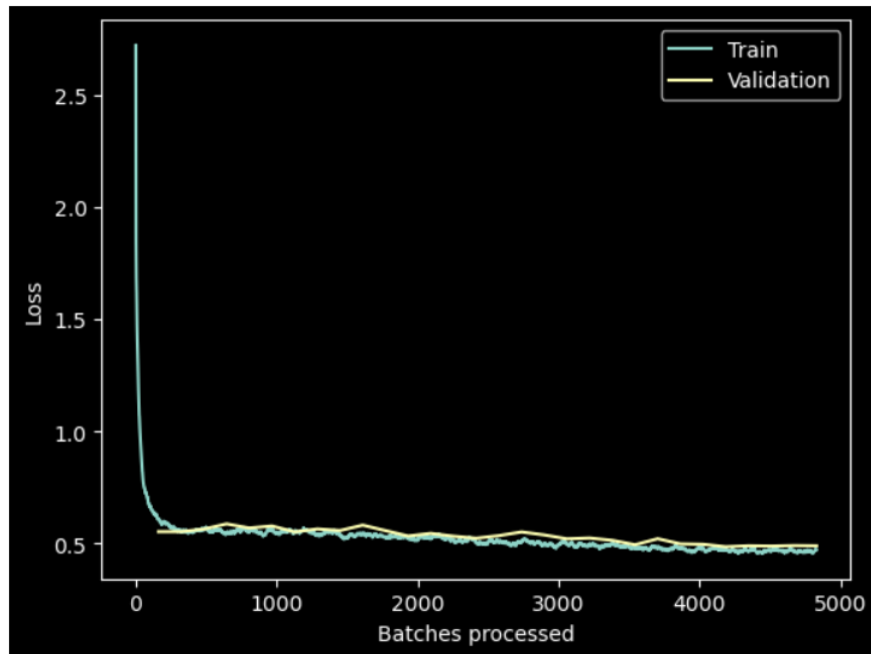


Abb. 59: Verlustdiagramm von FasterRCNN von Datensatz 1 (Lernrate 000.1, 30 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Der Verlust auf den Trainingsdaten (türkisfarbene Kurve) nimmt zunächst stark ab und stabilisiert sich nach den ersten 1000 Batches.
- Die Trainingskurve wird vom Validierungsverlust (gelbe Kurve) sehr eng verfolgt.
- Die geringe Varianz zwischen beiden Kurven weist darauf hin, dass das Modell stabil trainiert wurde und keine plötzlichen Änderungen aufgetreten sind, wie sie beispielsweise durch eine zu hohe Lernrate verursacht werden könnte. Weder Anzeichen für Overfitting noch Underfitting sind vorhanden.

Insgesamt zeigt das trainierte Modell ein stabiles Verhalten. Der Validierungsverlust zeigt keine großen Schwankungen, was möglicherweise auf die angewendeten Augmentationen zurückzuführen ist. Die AP weist eine stetige Verbesserung auf, was auf eine robuste Leistungssteigerung hindeutet. Die Datenqualität und die Trainingsparameter (Hyperparameter) scheinen gut aufeinander abgestimmt zu sein. Zum Beispiel scheint die Wahl der Lernrate von 0.001 dem Datensatz zugutezukommen.

Der Bestand ist erreicht. Da es zu keinem Over- oder Underfitting kommt, wird der aktuelle Trainingszustand als optimal bewertet. Eine weitere Optimierung der Anzahl an Epochen oder anderer Parameter ist nicht notwendig, da das Modell bereits seinen Bestand erreicht hat.

6.7.2 Faster R-CNN – Kombiniertes Datensatz

Der Hauptgrund, warum die zuvor beschriebenen Schritte mit dem kombinierten Datensatz wiederholt werden, liegt darin, zu überprüfen, ob das Modell durch zusätzliche Annotationen anderer Klassen

besser lernt, Photovoltaikanlagen von anderen Objekten zu unterscheiden. In diesem Fall wurden Solarthermieranlagen annotiert, da diese optisch eine starke Ähnlichkeit zu Photovoltaikanlagen aufweisen.

Die Definition des FasterRCNN-Modells, die Wahl des Backbone-Netzes und der Batchgröße bleiben unverändert. Zur Ermittlung der Batchgröße wurde zuvor die Funktion `arcgis.learn.estimate_batch_size()` verwendet. Zur Überprüfung der Datenverteilung und der Gesamtanzahl der Trainings- und Validierungsdaten wurde die Funktion `<prepare_data-Variable>.show_batch()` verwendet, welches per Zufallsprinzip einen Überblick über die Daten liefert.



**Abb. 60: Stichprobenartige Anzeige der Annotationen auf den Bilddaten im kombinierten Datensatz
(FasterRCNN, 10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)**

Mit `<Modelldefinition>.lr_find()` wurde vor dem Training die optimale Lernrate ermittelt. Die dabei erzeugte Lernratenkurve weist ein Intervall um 0.0003 als empfohlene Wahl aus. In der Funktion `<Modelldefinition>.fit()` kam diese Lernrate zum Einsatz. Die Lernrate bestimmt, in welchem Ausmaß das Modell die Gewichte nach jeder Trainingseinheit (Iteration) anpasst.

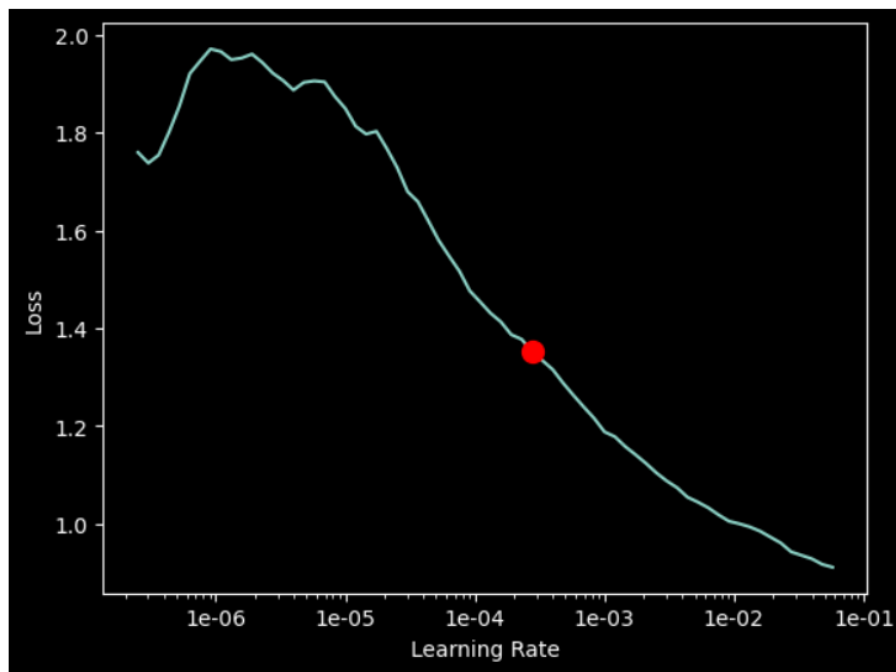


Abb. 61: Lernratenkurve von FasterRCNN im kombinierten Datensatz (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

6.7.2.1 Faster R-CNN: Training über 10 Epochen ohne Transformationen

In `<Modelldefinition>.fit()` wurden erneut die vorher ermittelte Lernrate, die Epochenanzahl, sowie das Keyword-Argument `early_stopping=True` als Parameter definiert. Auch dieses Mal werden nach Abschluss des Trainings die Anzahl an Epochen und der benötigten Zeit je Epoche, der Verlust auf dem Trainingsdatensatz, der Verlust auf dem Validierungsdatensatz sowie die Average Precision für die Vorhersagen angezeigt.

epoch	train_loss	valid_loss	average_precision	time
0	0.386325	0.418240	0.330904	38:26
1	0.358036	0.406338	0.376519	38:21
2	0.358867	0.400539	0.415065	38:22
3	0.346362	0.399107	0.445218	38:23
4	0.341763	0.378326	0.469015	38:24
5	0.307100	0.385934	0.490347	38:23
6	0.310308	0.366121	0.527755	38:26
7	0.307522	0.363784	0.549828	38:24
8	0.303282	0.360924	0.560710	38:28
9	0.296318	0.358649	0.552205	38:28

Abb. 62: FasterRCNN-Modellergebnis nach Training mit kombiniertem Datensatz (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Der Trainingsverlust reduziert sich über die 10 Epochen von etwa 0.3863 auf 0.2963. Das deutet darauf hin, dass das FasterRCNN-Modell gut lernt und die Trainingsdaten effizienter verarbeitet. Auch der Verlust auf den Validierungsdaten zeigt eine ähnliche Entwicklung und verringert sich von 0.4182 auf 0.3587. Er bleibt ab etwa der sechsten Epoche nahezu konstant bei ungefähr 0.36. Diese Entwicklung könnte andeuten, dass das FasterRCNN-Modell eine Sättigung erreicht hat und weitere Epochen nur geringfügige Optimierungen bewirken könnten. Die Average Precision nimmt zu und erreicht 0.5522 (zuvor: 0.3310). Das weist auf eine markante Verbesserung der Leistung des Modells bei der Differenzierung der beiden Klassen (Photovoltaikanlagen und Solarthermieranlagen) hin. Das Ergebnis

erklärt auch, dass das Modell mit lediglich zwei Klassen eine einfachere Aufgabe hat, da Modelle bei einer geringeren Anzahl von Klassen dazu neigen, schneller zu lernen, die Trainingsdaten gut zu repräsentieren und Vorhersagen zu treffen. Auch in den visualisierten Resultaten wird dies deutlich.

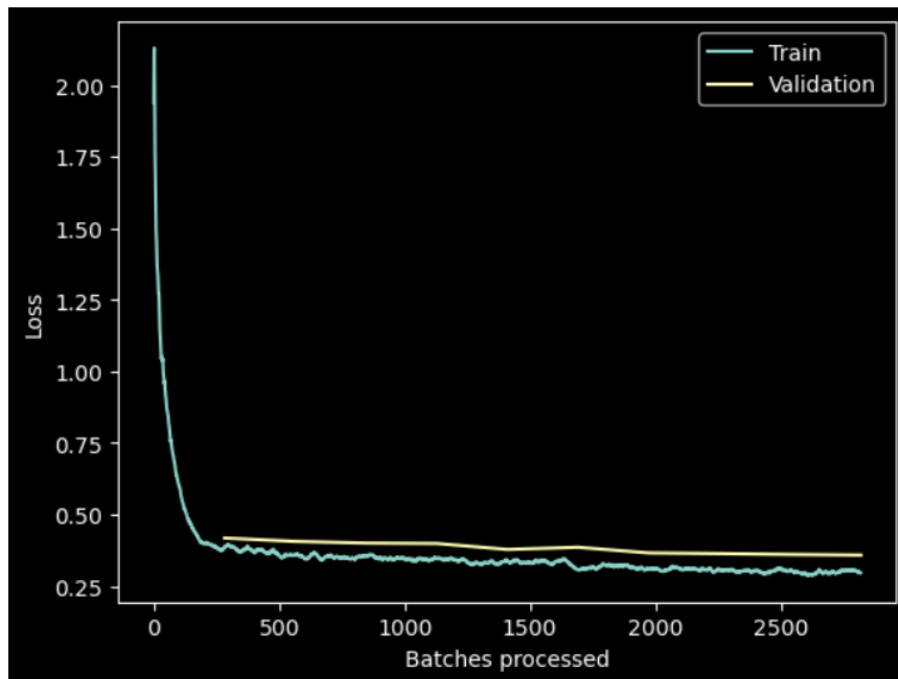


Abb. 63: Verlustdiagramm von *FasterRCNN* vom kombinierten Datensatz (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Vor allem in den ersten Epochen ist eine kleine Lücke zwischen den Kurven des Trainings- und Validierungsverlustes zu erkennen. Ein geringfügiges Ungleichgewicht zwischen dem Trainings- und Validierungsverlust ist besonders nach den ersten Batches charakteristisch, da das Modell zu Beginn intensiver auf die Trainingsdaten optimiert wird.
- Der Trainings- wie auch der Validierungsverlust verbessert sich kontinuierlich, ohne dass Anzeichen von Overfitting zu erkennen sind. Das heißt, der Validierungsverlust nimmt nicht zu.
- Das Modell weist auf den Validierungsdaten konsistente Trainingsergebnisse auf, was auf eine erfolgreiche Generalisierung hindeutet.

Das trainierte Modell zeigt nach nur zehn Epochen stabile und vielversprechende Ergebnisse. Die AP zeigt, dass das Modell beide Klassen gut differenzieren kann. Dennoch ist es möglich, dass weitere Trainingsepochen die Trainingsleistung weiter verbessern könnten und besonders die Average Precision weiter erhöhen. Darüber hinaus könnten weitere Transformationen (Flip sowie Licht- und Kontraständerungen) die Generalisierungsfähigkeit des kombinierten Datensatz verbessern.

Um die Average Precision für jede Klasse separat bewerten zu können, kann die gespeicherte EMD-Datei geöffnet und mit einem Texteditor untersucht werden. Auf diese Weise zeigen sich mögliche Ungleichheiten in der Klassenerkennung. Die bisherige Analyse zeigt eine sehr ausgewogene Verteilung der AP-Werte zwischen den beiden Klassen:

```
68     "average_precision_score": {  
69         "Class_1": 0.5503265241160068,  
70         "Class_2": 0.554084336355611  
71     },
```

Abb. 64: Average Precision-Ergebnis für beide Klassen (FasterRCNN, 10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit Notepad++ 7.9)

Einige Modellergebnisse wurden nach Abschluss des Trainingsprozesses mit der Funktion `<Modelldefinition>.show_results()` dargestellt (siehe Abbildung 65). Mit ihr ist es möglich, die Ground Truth-Daten mit den vom Modell prognostizierten Ergebnissen zu vergleichen.



Abb. 65: Funktion `show_results()` für FasterRCNN mit Ground Truth (l.) und Vorhersagen (r.)
(Kombinierter Datensatz, 10 Epochen, ohne Transformationen)
 (Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

6.7.2.2 Hyperparameter-Tuning für Faster R-CNN: Training über 10 Epochen inkl. Transformationen
Zudem wurden zusätzliche Augmentationen in der `prepare_data()`-Funktion umgesetzt, um die Vorhersagen des Modells zu optimieren. Zunächst wurde die Anzahl der Epochen unverändert auf zehn belassen.

epoch	train_loss	valid_loss	average_precision	time
0	0.463356	0.407888	0.303165	38:52
1	0.412469	0.406233	0.424221	41:13
2	0.407148	0.385843	0.439682	41:19
3	0.397279	0.372877	0.489240	41:27
4	0.391605	0.398267	0.497139	45:00
5	0.373131	0.383611	0.524864	44:59
6	0.376212	0.370041	0.546940	44:58
7	0.364562	0.366036	0.554154	44:59
8	0.349435	0.359534	0.567729	45:02
9	0.362345	0.361912	0.569990	44:58

Abb. 66: FasterRCNN-Modellergebnis nach Training mit kombiniertem Datensatz (10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Bemerkenswert ist, dass die durchschnittliche Trainingszeit auf nahezu 45 Minuten angestiegen ist. Die zusätzlichen Schwankungen zwischen den Epochen sind zwar nicht signifikant, aber aufgrund der zusätzlichen Transformationen nachvollziehbar, da sie den Rechenaufwand erhöhen. Ein positiver Aspekt ist die stetige Abnahme des Validierungsverlusts, was auf eine gute Modellanpassung an die Daten hindeutet. Die AP erhöhte sich von 0.3032 auf 0.5700, was eine deutliche Verbesserung der Vorhersagegenauigkeit zeigt. Es ist möglich, dass die zusätzlichen Augmentationen zu diesem Zeitpunkt eine positive Wirkung auf die Generalisierung des Modells hatten. Das Modell wurde robuster gegenüber Datenmodifikationen – wie die gesteigerte Average Precision zeigt – im Vergleich zum Training ohne Transformationen. Der Validierungsverlust bleibt in den letzten Epochen konstant, was darauf hindeutet, dass ein gutes Gleichgewicht zwischen Overfitting und Generalisierung besteht. Es ist jedoch möglich, dass zehn Epochen nicht ausreichen, um diese Erkenntnis zu stützen. Daher wäre es sinnvoll, die Anzahl der Epochen zu erhöhen, um das Modell weiter zu optimieren.

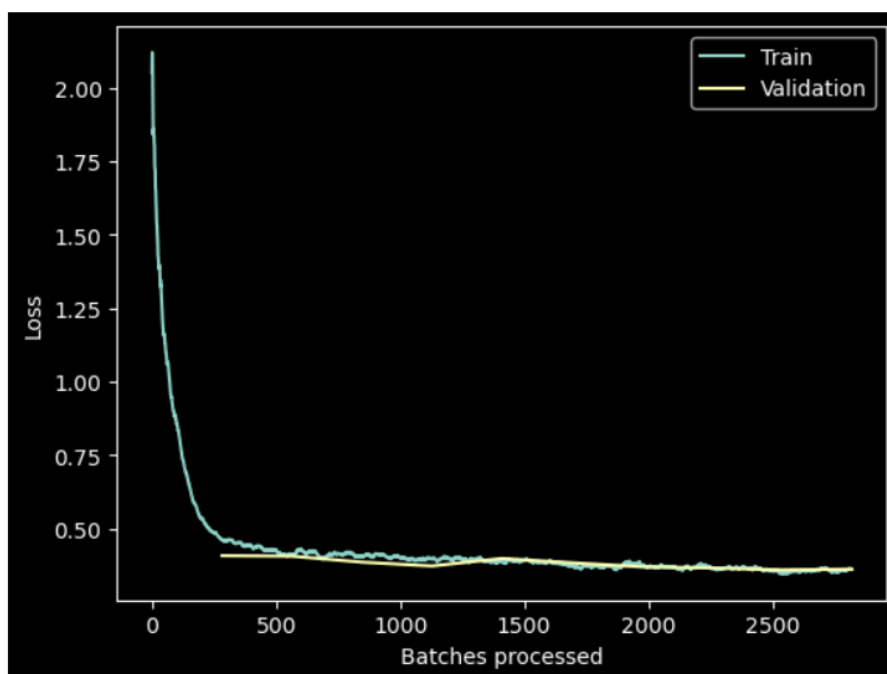


Abb. 67: Verlustdiagramm von FasterRCNN vom kombinierten Datensatz (10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Die Verlustkurven für Training und Validierung liegen eng beieinander, was auf eine gute Generalisierungsfähigkeit des Modells hindeutet.
- Die beiden Verluste nehmen kontinuierlich ab, was darauf Hinweis gibt, dass das Modell weiterhin lernt.
- Im Laufe der Epochen stabilisiert sich der Verlust des Validierungsdatensatzes, was auf ein ausgewogenes Modell hinweist.
- Die geringe Abweichung zwischen den Verlustkurven verdeutlicht die reduzierte Gefahr von Overfitting.

Die Transformationen haben scheinbar dazu beigetragen, das FasterRCNN-Modell stabiler zu machen und die Vielfalt der Daten besser abzubilden, wodurch seine Generalisierungsfähigkeit verbessert wurde. Um das Modell weiter zu verbessern und die Vorhersagegenauigkeit zu optimieren, bevor ein Sättigungspunkt erreicht wird, könnte es sinnvoll sein, bei zukünftigen Trainingsläufen längere Trainingszeiten (mehr Epochen) mit zusätzlichen Augmentationen zu kombinieren.

Die separate Average Precision für jede einzelne Klasse bleibt auch mit den zusätzlichen Transformationen ausgeglichen:

```
68 "average_precision_score": {
69   "Class_1": 0.5783904367245883,
70   "Class_2": 0.5570685081865365
71 },
```

Abb. 68: Average Precision-Ergebnis für beide Klassen (FasterRCNN, 10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit Notepad++ 7.9)

6.7.3 Mask R-CNN – Datensatz 1

Im Gegensatz zu vorher wird dieses Mal die Klasse MaskRCNN in die ArcGIS Pro-Umgebung integriert, um ein Mask R-CNN-Modell zu initialisieren. Zur Erinnerung: Mask R-CNN baut auf Faster R-CNN auf, indem es neben Bounding Boxes auch pixelgenaue Segmentierungsmasken für erkannte Objekte erstellt.

Alle für das Modell verfügbaren Backbone-Netze können mit der Funktion `<Modelldefinition>.backbones()` abgerufen werden. Auch an dieser Stelle kam zunächst **ResNet-50** in der Modelldefinition zum Einsatz. Aufgrund eines „Out of Memory“-Fehlers wurde jedoch eine Liste der verfügbaren Backbones eingesehen, woraufhin **ResNet-34** als Argument bzw. Alternative gewählt wurde. Dieses Netzwerk benötigt weniger Speicher und eignet sich besser für die vorhandene Hardware. Die maximale Batchgröße, die das MaskRCNN-Modell laut der Funktion `arcgis.learn.estimate_batch_size()` verarbeiten kann, beträgt 8 Datenpunkte pro Batch (im Gegensatz zu 32 beim Vorgängermodell). Dies ist auf die Architektur und Komplexität des Modells zurückzuführen: Für jedes identifizierte Objekt wird eine pixelgenaue Maske erstellt, und jede generierte Bounding Box wird zusätzlich pixelweise untersucht. Diese Abläufe führen zu einer höheren Anzahl an Berechnungen und erfordern viel Speicherplatz, wodurch die Batchgröße begrenzt wird, da der Speicherbedarf je Datenpunkt bzw. Bild größer ist.

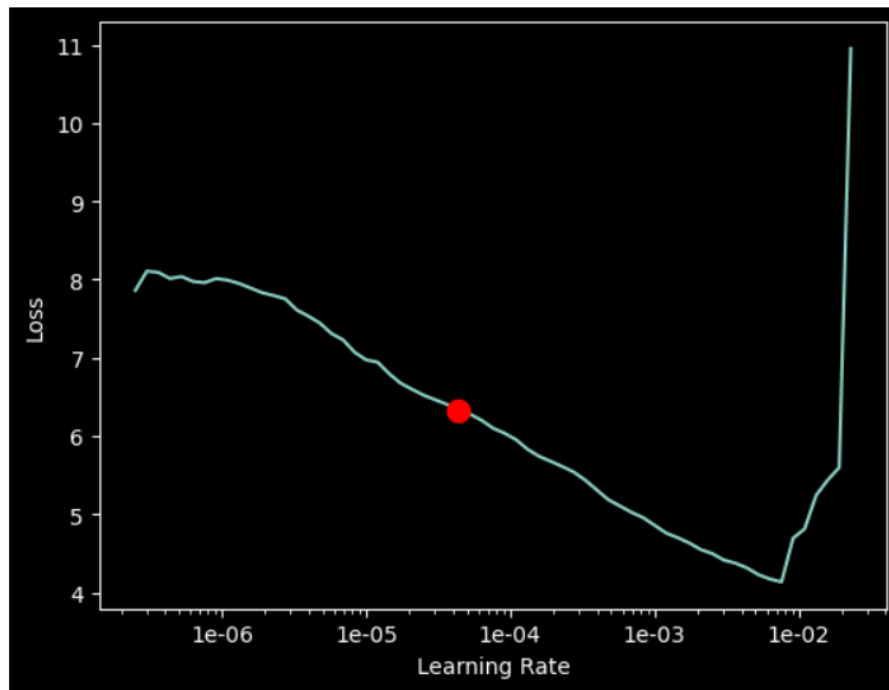
Zur Analyse der Datenverteilung sowie der Gesamtanzahl der Trainings- und Validierungsdaten wurde die Funktion `<prepare_data-Variable>.show_batch()` eingesetzt, um einen Überblick zu ermöglichen (siehe Abbildung 69).



Abb. 69: Stichprobenartige Anzeige der Annotationen auf den Bilddaten in Datensatz 1
(MaskRCNN, 10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Im Anschluss an die Ermittlung der optimalen Lernrate für das Training mithilfe der Funktion `<Modelldefinition>.lr_find()` erfolgt im nächsten Schritt das Trainieren des Modells auf den Trainingsdaten mit `<Modelldefinition>.fit()`. Die Lernrate liegt bei 4.37×10^{-5} bzw. 0.00004365.

Auch für MaskRCNN können die Metriken `train_loss`, `valid_loss` und `average_precision` verwendet werden.



4.365158322401661e-05

Abb. 70: Lernratenkurve von MaskRCNN in Datensatz 1 (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

6.7.3.1 Mask R-CNN: Training über 10 Epochen ohne Transformationen

Zuallererst wurde ein Training über zehn Epochen ohne Transformationen durchgeführt.

epoch	train_loss	valid_loss	average_precision	time
0	2.465095	1.830197	0.000000	1:18:45
1	1.798950	1.387218	0.007107	1:42:29
2	1.605958	1.305072	0.000391	1:20:34
3	1.581034	1.299632	0.007565	1:19:32
4	1.492469	1.258286	0.016469	1:01:42
5	1.561141	1.268867	0.024714	1:23:28
6	1.459822	1.280004	0.022694	55:50
7	1.611877	1.249089	0.058724	50:04
8	1.498023	1.244749	0.047842	1:02:26
9	1.462203	1.244872	0.061928	1:28:42

Abb. 71: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

In der Spalte `time` wird deutlich, dass es während des Trainings zu größeren Schwankungen kam. Der anfängliche Verlust verringerte sich über die Epochen hinweg stetig. Ab der vierten Epoche zeigt der Verlust der Validierungsdaten eine Stabilisierung ohne signifikante Schwankungen. Der Verlust auf dem Trainingsdatensatz belief sich zuletzt auf 1.4622, während der Validierungsverlust 1.2449 war. Die Average Precision (AP) startete bei 0 und hat zum Ende der Epochen ihren Höchstwert von 0.0619 erreicht.

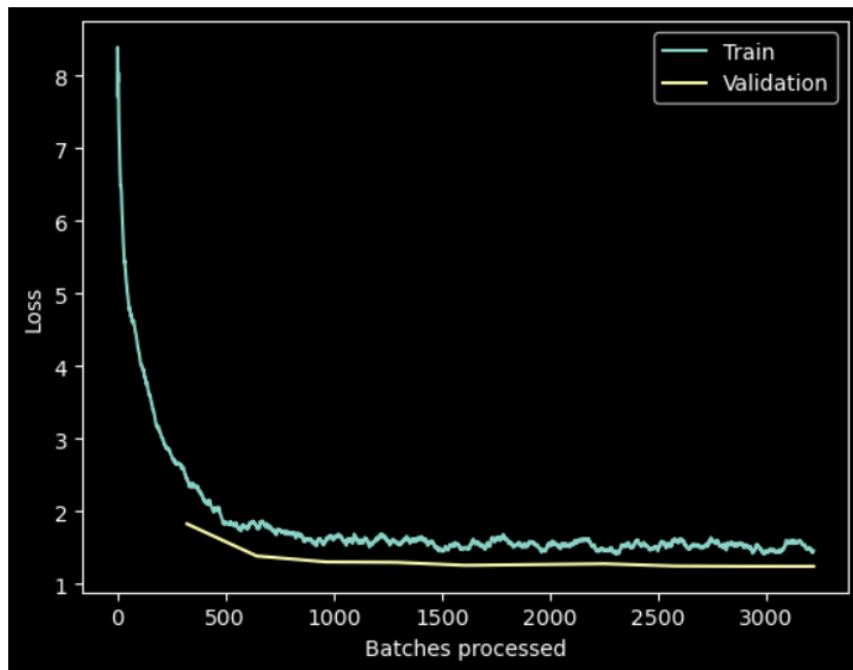


Abb. 72: Verlustdiagramm von MaskRCNN vom Datensatz 1 (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Das Verlustdiagramm deutet auf einen klassischen Verlauf, indem der Trainingsverlust (türkisfarbene Linie) zu Beginn steil abnimmt und sich ab etwa Batch 1000 stabilisiert.
- Auch der Verlust auf den Validierungsdaten (gelbe Linie) verringert sich konstant, liegt jedoch immer leicht unter dem Trainingsverlust, was auf das Fehlen eines Overfittings hindeutet.
- Der Validierungsverlust liegt im akzeptablen Bereich. Die Validierungswerte weisen darauf hin, dass das Modell Fortschritte erzielt hat.
- Die sanfte Abflachung der beiden Kurven weist darauf hin, dass das MaskRCNN-Modell die Trainingsdaten gut erfasst hat. Die Ergebnisse der Validierung ließen sich allerdings noch optimieren.
- Die AP-Werte im Bereich von 0.05 bis 0.06 deuten auf ein leichtes Underfitting hin, da das Modell die Daten noch nicht vollständig ausschöpfen kann.

Nach Abschluss des Trainings wurden die Ergebnisse mit der Funktion `<Modelldefinition>.show_results()` visualisiert (siehe Abbildung 73). In der Darstellung werden die Ground Truth-Daten den Modellvorhersagen gegenübergestellt. Es ist erkennbar, dass die Vorhersagen in einigen Fällen erheblich von den tatsächlichen Annotationen abweichen, was auf eine unzureichende Leistung des Modells hinweist. Insbesondere bei der exakten Form und vollständigen Erfassung der Objekte zeigen sich Schwächen. Größere Objekte werden jedoch tendenziell besser erkannt.



Abb. 73: Funktion `show_results()` für MaskRCNN mit Ground Truth (l.) und Vorhersagen (r.)
(Datensatz 1, 10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

6.7.3.2 Hyperparameter-Tuning für Mask R-CNN: Training über 10 Epochen inkl. Transformationen

Im Anschluss daran wurde ein Training mit zehn Epochen und Transformationen durchgeführt. Die optimalen Lernrate für das Training beträgt laut Funktion `<Modelldefinition>.lr_find()` 4.37×10^{-5} bzw. 0.00004365.

epoch	train_loss	valid_loss	average_precision	time
0	2.273970	1.795817	0.000000	2:19:18
1	1.496199	1.287673	0.000114	1:17:06
2	1.319493	1.210546	0.010119	1:26:36
3	1.293587	1.190125	0.004467	1:24:16
4	1.379870	1.164733	0.020516	1:00:10
5	1.346420	1.157285	0.019905	1:28:21
6	1.304446	1.166306	0.056013	36:39
7	1.346232	1.148097	0.041754	1:45:57
8	1.316486	1.159151	0.058628	1:22:49
9	1.318153	1.159456	0.050773	1:35:31

Abb. 74: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Verglichen mit dem vorherigen Trainingslauf, beginnen die Verluste auf beiden Datensätzen bei niedrigeren Werten. Am Ende des Trainings erreichte der Trainingsverlust einen Wert von 1.3181, während der Validierungsverlust bei 1.1595 lag. Die durchschnittliche Genauigkeit (AP) begann auch in diesem Lauf bei 0 und erreichte zum Ende der Epochen einen Wert von 0.0508, was eine vergleichbare Modellleistung zum ersten Versuch widerspiegelt. Obwohl sich die Verlustwerte mit Transformationen verbessert haben, zeigt die geringere AP, dass die Erkennungsgenauigkeit leicht variieren kann.

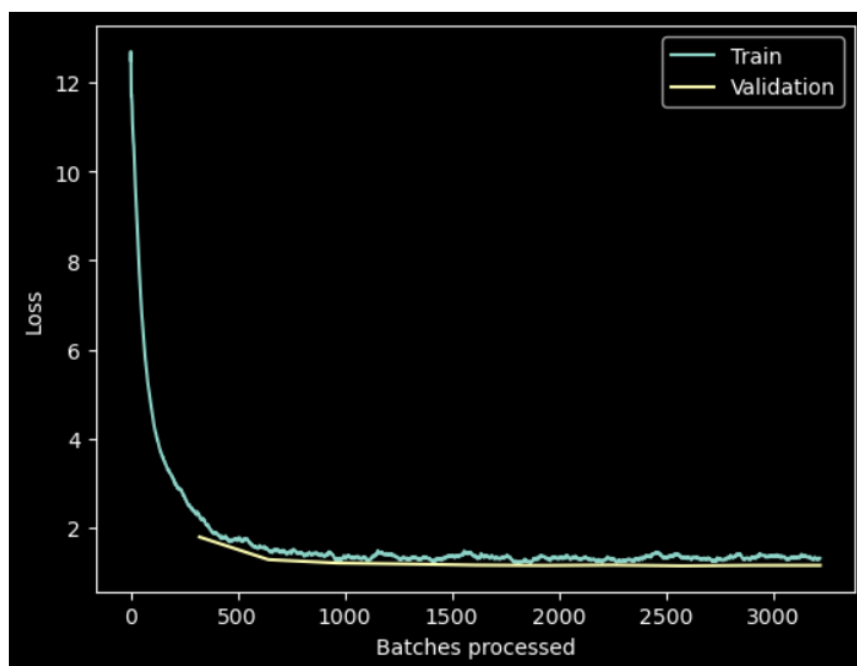


Abb. 75: Verlustdiagramm von MaskRCNN vom Datensatz 1 (10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Der Verlust auf dem Trainingsdatensatz nimmt stark ab und stabilisiert sich nach etwa Batch 1000. Der Verlust auf dem Validierungsdatensatz zeigt leichte Schwankungen, stabilisiert sich jedoch auf einem niedrigeren Niveau als im ersten Versuch. Dies lässt vermuten, dass die Datenaugmentation eine bessere Generalisierung ermöglicht.

- Die Kurven für Trainings- und Validierungsverlust verlaufen hinreichend parallel und konvergieren, was darauf hindeutet, dass kein Overfitting stattfindet.
- Auch in diesem Lauf könnten die geringen Werte der Average Precision auf ein geringfügiges Underfitting hindeuten, weil das Modell die Möglichkeiten der Daten nicht voll nutzt.
- Es ist anzunehmen, dass die vorangegangenen Transformationen einen positiven Effekt gehabt haben, da insbesondere in Bezug auf die Stabilität des Validierungsverlusts Verbesserungen zu erkennen sind.

Offenbar hat die Verwendung von Transformationen zu einem stabileren Validierungsverlust und zu einer besseren Anpassung des MaskRCNN-Modells an die Daten geführt. Dies spielt vor allem dann eine Rolle, wenn das Modell auf Daten angewendet wird, die ihm nicht bekannt waren. Um die Average Precision zu erhöhen, könnten bei künftigen Trainingsläufen weitere Transformationen getestet werden. Darüber hinaus könnte die Lernrate verbessert werden, da sie momentan sehr niedrig ist und eventuell die Fortschritte beim Training einschränkt.

6.7.3.3 Hyperparameter-Tuning für Mask R-CNN: Training über 30 Epochen ohne Transformationen

epoch	train_loss	valid_loss	average_precision	time
0	2.403363	1.943485	0.000463	2:00:59
1	1.765506	1.426231	0.002992	1:40:42
2	1.679627	1.350146	0.000865	1:43:34
3	1.585812	1.302782	0.001470	1:16:13
4	1.598442	1.293771	0.023424	38:15
5	1.540804	1.249179	0.026221	1:00:28
6	1.522107	1.241279	0.004948	1:04:42
7	1.631602	1.303462	0.001471	1:26:51
8	1.485910	1.251137	0.035741	1:13:26
9	1.474906	1.239795	0.029231	1:09:57
10	1.350219	1.251900	0.114201	1:29:49
11	1.437819	1.227068	0.093441	1:06:37
12	1.460096	1.257083	0.106505	1:31:16
13	1.423428	1.203213	0.101995	1:08:39
14	1.495070	1.207899	0.110513	1:29:51
15	1.355799	1.182264	0.159295	1:35:53
16	1.420566	1.215145	0.097804	1:21:04
17	1.349988	1.202709	0.135582	1:23:07
18	1.411986	1.203038	0.188633	1:17:12
19	1.448408	1.218922	0.187760	1:57:57
20	1.403798	1.237905	0.182452	1:09:22
21	1.304836	1.188565	0.184551	1:55:11
Epoch 21: early stopping				

Abb. 76: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (30 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Der Trainingsverlust verringert sich von anfänglich rund 2.4034 in Epoche 0 auf 1.3048 in Epoche 21. Als keine signifikanten Verbesserungen mehr erzielt werden konnten, wurde das Training vorzeitig beendet. Das unterstreicht die effektive Nutzung dieses Parameters. Der Validierungsverlust ist von 1.9435 in Epoche 0 auf 1.1886 in der letzten Epoche gesunken, was eine gute Verbesserung des Modells während des Trainings zeigt. Die Average Precision (AP) startete mit einem sehr geringen Wert von 0.000463 und erreichte in Epoche 18 ihren Höchstwert von 0.1886. Die Entwicklung legt nahe, dass das Modell durch die kleine Lernrate und das ResNet-34-Backbone eine stabile Basis für das Training

erhielt. Allerdings zeigen die Schwankungen der AP eine gewisse Instabilität in der Objekterkennung, die möglicherweise auf begrenzte Datenqualität, eine kleine Batchgröße oder allgemein suboptimale Hyperparameter zurückzuführen ist.

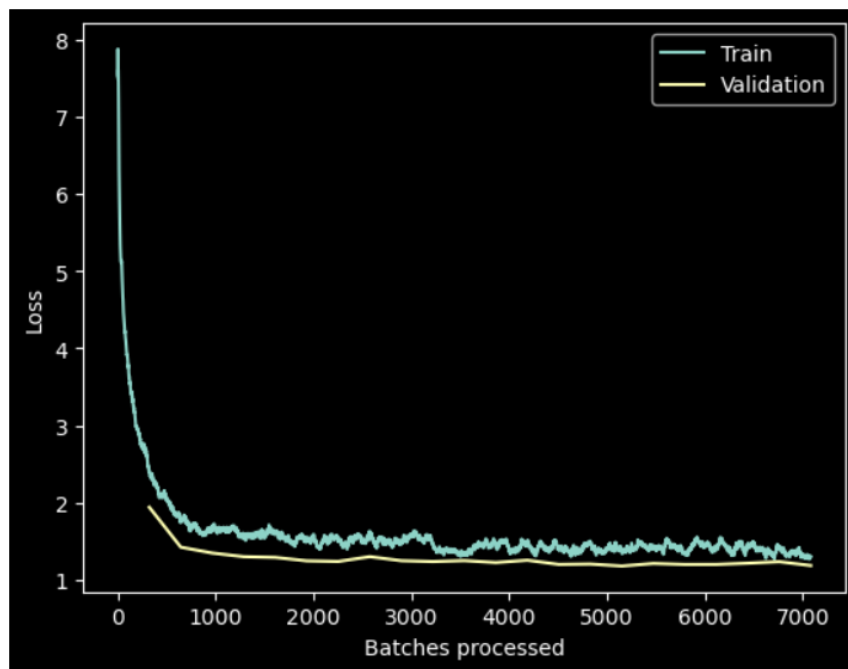


Abb. 77: Verlustdiagramm von MaskRCNN vom Datensatz 1 (30 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Die in Türkis dargestellte Kurve (Trainingsverlust) fällt rasch ab, vor allem in den ersten Epochen, und findet später eine Stabilisierung. Sie bleibt allerdings leicht schwankend.
- Auch der Verlauf der gelben Kurve (Validierungsverlust) ist rückläufig, was auf eine gute Generalisierung des Modells hinweist.
- Der Verlust auf den Validierungsdaten ist konstant etwas niedriger als der Trainingsverlust, was auf eine gute Generalisierung hinweist und darauf, dass weder starkes Overfitting noch Underfitting vorliegt.
- Die Schwankungen in der AP-Metrik könnten darauf hinweisen, dass die Datenqualität oder die Hyperparameter optimiert werden müssen.

Wie die Verlustkurven zeigen, konvergiert das MaskRCNN-Modell gut und zeigt insgesamt eine solide Leistung. Weitere Optimierungen, wie das Implementieren von Transformationen, die Wahl eines besser geeigneten Backbone-Netzwerks oder eine Feinanpassung der Hyperparameter, könnten zu einer weiteren Steigerung der Ergebnisse (insbesondere der Average Precision) und zu einer Reduzierung des leicht schwankenden Trainingsverlusts führen.

6.7.3.4 Hyperparameter-Tuning für Mask R-CNN: Training über 30 Epochen inkl. Transformationen und Lernrate von 0.001

epoch	train_loss	valid_loss	average_precision	time
0	1.546894	1.273198	0.002280	1:19:12
1	1.402005	1.224972	0.000028	40:37
2	1.359689	1.182056	0.002546	24:34
3	1.376855	1.235877	0.015742	33:14
4	1.289404	1.131824	0.066936	1:23:31
5	1.322486	1.195975	0.060073	1:08:32
6	1.323259	1.127724	0.085687	2:20:01
7	1.240180	1.098448	0.151384	1:50:09
8	1.176545	1.088716	0.105197	1:41:06
9	1.228492	1.126912	0.075742	46:18
10	1.183129	1.095843	0.164556	40:40
11	1.192811	1.066255	0.181122	53:15
12	1.198533	1.052088	0.186933	36:00
13	1.162829	1.062034	0.264401	1:31:05
14	1.138955	1.095353	0.261571	1:18:27
15	1.160345	1.037336	0.229677	44:49
16	1.086981	1.045031	0.282301	1:10:53
17	1.176514	1.042913	0.203165	1:08:47
18	1.149632	1.064990	0.310008	1:06:53
19	1.071965	1.062186	0.205400	1:12:07
20	1.111187	1.032020	0.280252	1:01:21
21	1.129746	1.014858	0.272738	1:08:08
22	1.038748	1.039005	0.331572	46:03
23	1.079819	1.044968	0.306064	44:12
24	1.140562	1.060826	0.315586	26:13
25	1.148966	1.049270	0.359726	1:01:38
26	1.174759	1.030798	0.325003	40:25
27	1.124001	1.045293	0.349822	25:55

Epoch 27: early stopping

Abb. 78: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen, Lernrate)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Die Trainingszeit pro Epoche schwankt erheblich, von etwa 25 Minuten bis zu 2 Stunden und 20 Minuten. Dies könnte auf eine unterschiedliche GPU-Auslastung oder Datenladungsprozesse hinweisen. Der Trainingsverlust beginnt bei 1.5469 und sinkt kontinuierlich auf 1.1240. Der Trend des Validierungsverlusts ist derselbe: Er beginnt bei 1.2732 und fällt auf 1.0453. Der Verlust auf den Validierungsdaten zeigt ab Epoche 21 leichte Schwankungen, bleibt jedoch danach relativ konstant. Ein ausgeprägtes Overfitting ist nicht zu beobachten. Die Average Precision beginnt bei einem sehr niedrigen Wert von 0.0023. Ab der vierten Epoche steigt sie jedoch merklich auf 0.0669 und erreicht in der 25. Epoche mit 0.3597 ihren Höhepunkt. In Epoche 27 erfolgt dann ein leichter Rückgang auf 0.3498. Dies könnte darauf hindeuten, dass das Modell sein Leistungslimit erreicht hat.

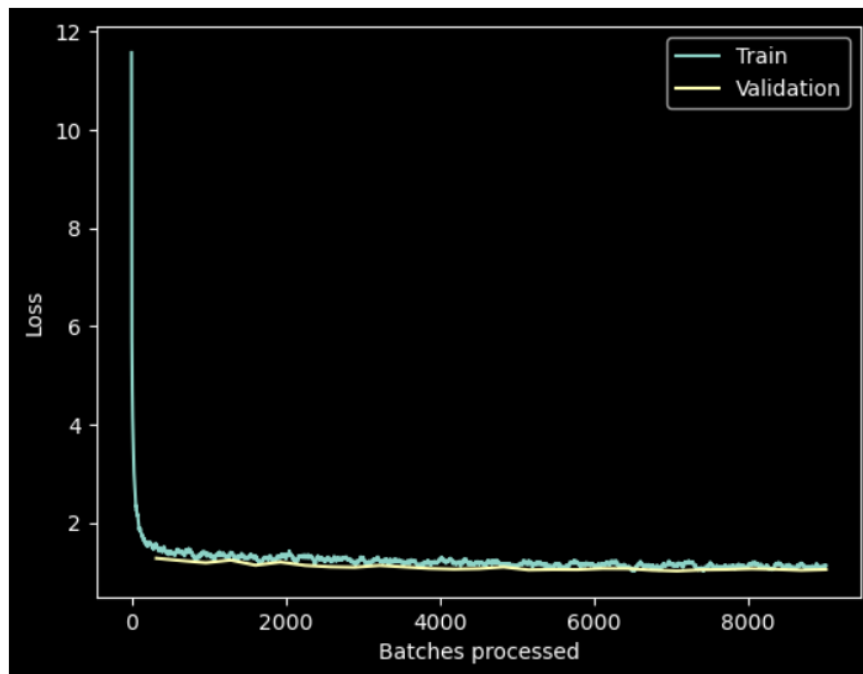


Abb. 79: Verlustdiagramm von MaskRCNN von Datensatz 1 (30 Epochen, ohne Transformationen, Lernrate)
 (Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Der Trainingsverlust (türkisfarbene Kurve) und der Validierungsverlust (gelbe Kurve) zeigen einen starken anfänglichen Abfall. Zu Beginn des Trainings lernt das Modell in sehr hohem Tempo und verringert so den Verlust erheblich.
- Nach den ersten 2000 bis 3000 Batches pendelt sich der Verlust auf einen Wert von etwas über 1 ein. Im Anschluss daran wechselt das Training in eine Lernphase, die sich durch eine verlangsamte Progression und Stabilität auszeichnet. In dieser Phase reduziert sich der Verlust nur noch minimal.
- Da die beiden Kurven sehr nah beieinander verlaufen und keine großen Abweichungen aufweisen, deutet dies darauf hin, dass das Modell gut generalisiert und kein starkes Overfitting vorliegt.

Das Modell konnte eine akzeptable Average Precision von 0.3498 erzielen. Es könnte helfen, noch genauere Anpassungen zu erzielen, wenn die Trainingszeit verlängert und die Lernrate gesenkt wird.

6.7.4 Mask R-CNN – Kombiniertes Datensatz

Der kombinierte Datensatz wird erneut herangezogen, um zu untersuchen, ob das Modell durch die Einbindung zusätzlicher Annotationen anderer Klassen seine Fähigkeit verbessert, Photovoltaikanlagen von anderen Objekten zu unterscheiden. Hierbei wurden speziell Solarthermieanlagen berücksichtigt, da diese optisch eine hohe Ähnlichkeit mit Photovoltaikanlagen aufweisen.

6.7.4.1 Mask R-CNN: Training über 10 Epochen ohne Transformationen

epoch	train_loss	valid_loss	average_precision	time
0	1.888778	1.757827	0.000000	51:59
1	1.200661	1.193736	0.005015	37:54
2	1.102958	1.088593	0.026802	51:17
3	1.076890	1.035077	0.056389	2:15:28
4	0.972189	0.990142	0.082068	26:23
5	0.943230	0.955945	0.089203	1:37:12
6	0.945373	0.938019	0.094137	51:14
7	0.833532	0.937056	0.101552	43:02
8	0.892236	0.927045	0.104473	42:57
9	0.941279	0.923773	0.107672	43:45

Abb. 80: MaskRCNN-Modellergebnis nach Training mit kombinierten Datensatz (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Der Trainings- und Validierungsverlust sinken stetig, jedoch langsam. Der Verlust auf dem Trainingsdatensatz ist etwas geringer als der Validierungsverlust, was darauf hindeutet, dass der Trainingsprozess korrekt war und kein starkes Overfitting aufgetreten ist. Zunächst beträgt die Average Precision 0. Sie erhöht sich bis zum Ende der letzten Epoche auf 0.10767. Aus dem langsamen Anstieg lässt sich ableiten, dass das Modell Probleme hat, präzise Vorhersagen zu machen. Es ist möglich, dass die Trainingsdaten nicht ausreichend Variationen aufweisen oder dass eine weitere Anpassung der Hyperparameter erforderlich ist. Die Wahl einer sehr kleinen Lernrate könnte der Grund für die langsame Verbesserung der Verluste sowie der Average Precision sein.

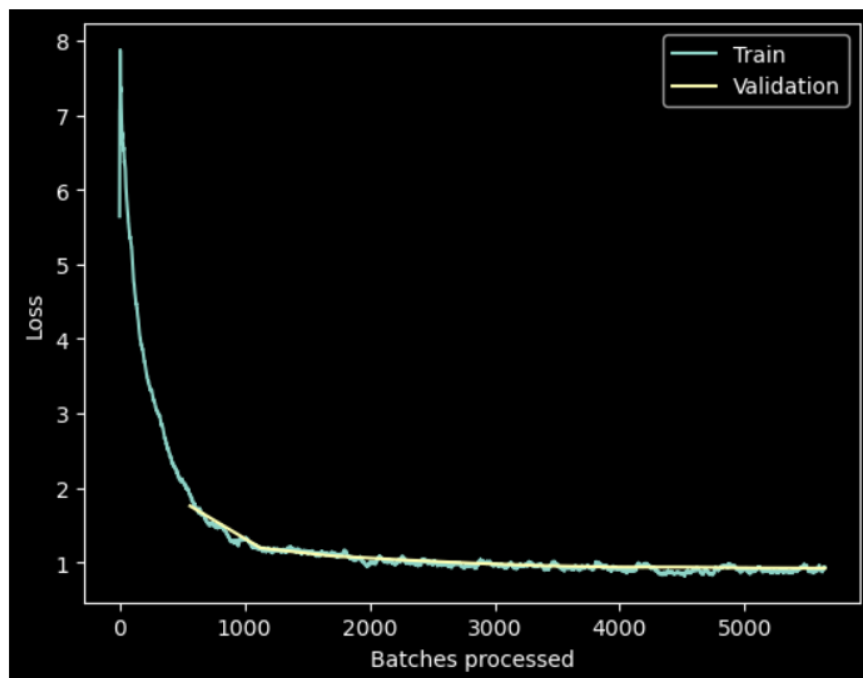


Abb. 81: Verlustdiagramm von MaskRCNN vom kombinierten Datensatz (10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Leichte Schwankungen der Trainingsverlustkurve deuten darauf hin, dass das Modell eventuell empfindlich auf die Gewichtungsinitalisierung, Batchgröße oder die Eingabedaten reagiert.
- Im Vergleich dazu ist die Validierungsverlustkurve glatter und stabiler, was auf eine robuste Generalisierung hindeutet.
- Die kleine Diskrepanz zwischen Trainings- und Validierungsverlust deutet darauf hin, dass kein ausgeprägtes Overfitting vorliegt.

- Der langsame Anstieg der AP-Metrik weist darauf hin, dass das Modell noch nicht vollständig von den Daten profitiert. Dies könnte durch die geringe Datenvariabilität oder eine suboptimale Lernrate bedingt sein.

Einerseits könnte eine moderate Erhöhung der Lernrate den Optimierungsprozess beschleunigen und die Vorhersagegenauigkeit verbessern. Andererseits könnte die Anpassung der Batchgröße die Schwankungen in der Trainingsverlustkurve verringern und somit den Trainingsprozess stabilisieren. Datenaugmentationen wie Flip oder Kontraständerungen könnten die Variationen in den Trainingsdaten erhöhen und die Modellleistung weiter steigern.

Zur getrennten Evaluierung der Average Precision jeder Klasse wird die EMD-Datei mit einem Texteditor geöffnet. Die aktuelle Untersuchung zeigt jedoch eine stark einseitige Verteilung der AP-Werte. Während für Klasse 1 (Photovoltaikanlagen) ein Wert von 0.1937 erreicht wurde, liegt der Wert für Klasse 2 bei 0. Dies ist ein Indiz dafür, dass das Modell Schwierigkeiten hat, Klasse 2 (Solarthermieranlagen) korrekt zu erkennen. Zugleich ist der AP-Wert für Klasse 1 jedoch wenig Erfolg versprechend.

```

64     "average_precision_score": {
65         "Class_1": 0.1937354988399072,
66         "Class_2": 0.0
67     },

```

Abb. 82: Average Precision-Ergebnis für beide Klassen (MaskRCNN, 10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Als Ursachen käme in Betracht, dass die Annotationen der Klasse 2 im Vergleich zu den oft größeren Photovoltaikflächen der Klasse 1 weniger markante Merkmale aufweisen. Darüber hinaus könnte das Modell mit nur zehn Epochen nicht ausreichend Zeit gehabt haben, um die Merkmale beider Klassen hinreichend zu erlernen. Die unzureichenden Modellvorhersagen zeigen sich auch in der Visualisierung mittels `<Modelldefinition>.show_results()` (siehe Abbildung 83). Es gibt praktisch keine zutreffende Erkennung von Objekten, die den Ground Truth-Daten entsprechen. Dies bedeutet, dass das Modell Schwierigkeiten hat, die Klassen (Photovoltaikanlagen und Solarthermieranlagen) korrekt zu identifizieren.

Eine gezielte Datenanreicherung, etwa durch Augmentationen oder das Bereitstellen zusätzlicher Trainingsdaten für Klasse 2, könnte helfen. Eine andere Möglichkeit wäre, die Gewichtung der Klasse 2 (STA) im Training zu verändern, d. h. die Bedeutung dieser Klasse im Lernprozess zu erhöhen. Diese Maßnahme wäre wahrscheinlich effektiver. Allerdings sind solche Modellanpassungen innerhalb von `arcgis.learn` begrenzt. Um den Trainingsprozess oder die Modellarchitektur stärker anzupassen, wären flexiblere Frameworks wie `PyTorch` notwendig.



Abb. 83: Funktion `show_results()` für MaskRCNN mit Ground Truth (l.) und Vorhersagen (r.)
(Kombinierter Datensatz, 10 Epochen, ohne Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

6.7.4.2 Hyperparameter-Tuning für Mask R-CNN: Training über 10 Epochen inkl. Transformationen

epoch	train_loss	valid_loss	average_precision	time
0	1.384841	1.393168	0.000000	2:42:03
1	1.031840	1.110262	0.031987	3:56:11
2	0.917388	1.002064	0.076090	4:33:49
3	0.858873	0.940803	0.066360	55:50
4	0.816339	0.906174	0.100161	1:27:17
5	0.773093	0.876588	0.127427	33:43
6	0.780532	0.847169	0.129823	43:41
7	0.750740	0.841558	0.130769	1:38:28
8	0.749874	0.831419	0.128589	41:45
9	0.747991	0.829099	0.135922	35:06

Abb. 84: MaskRCNN-Modellergebnis nach Training mit kombinierten Datensatz (10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Der Trainingsverlust sinkt von 1.3849 in Epoche 0 auf 0.7480 in Epoche 9. Der Validierungsverlust weist ebenfalls einen raschen und stabilen Rückgang auf, der konstant in der Nähe des Trainingsverlusts liegt, was auf ein besser generalisierendes Modell hindeutet. Im Vergleich zum ersten Versuch zeigt sich bei beiden Verlusten eine Verbesserung. In der neunten Epoche steigt die AP von 0 auf 0.1360. Verglichen mit dem ersten Versuch zeigt sich hier eine bessere Performance.

Die gewählte Lernrate von 5.25×10^{-5} bzw. 0.0000525 ist hoch für ein MaskRCNN-Modell, besonders bei einer Batchgröße von 16. Nichtsdestotrotz hat sich das Training stabilisiert, was darauf hinweist, dass die Lernrate in diesem Fall nicht zu groß war, um Unterschiede zu verursachen. Die etwas schnellere Reduktion der beiden Verluste könnte auf die höhere Lernrate zurückzuführen sein (im Vergleich zum ersten Trainingslauf), was zur schnelleren Optimierung beitrug.

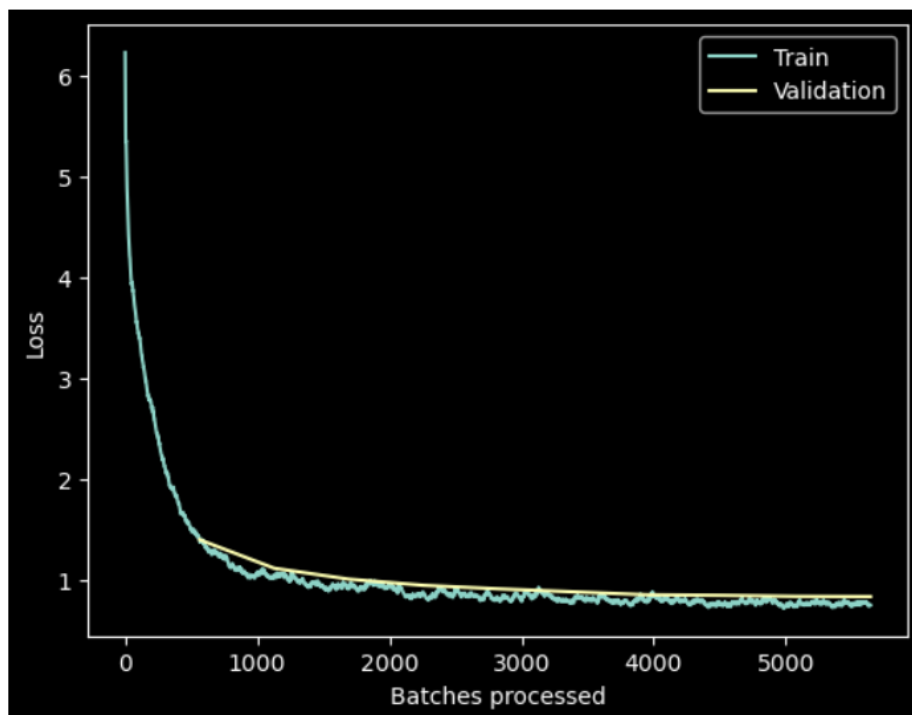


Abb. 85: Verlustdiagramm von MaskRCNN vom kombinierten Datensatz (10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Der Trainingsverlust (türkisfarbene Kurve) zeigt einen gleichmäßigeren Verlauf als im ersten Lauf, was auf ein stabileres Training hindeutet.
- Auch der Verlust auf den Validierungsdaten (gelbe Kurve) weist einen glatten Verlauf und eine gute Anpassungsfähigkeit des Modells auf.
- Mit hoher Wahrscheinlichkeit haben die Transformationen die Modellleistung verbessert.
- Vermutlich hat die höhere Lernrate eine schnellere Reduktion der Verluste bewirkt. Sie könnte in späteren Epochen jedoch verhindern, dass das Modell feinere Anpassungen erlernt, was auch den Average Precision-Wert begrenzen könnte.

Um die Modellfeinabstimmung zu optimieren und den AP-Wert weiter zu erhöhen, könnte es daher ratsam sein, die Lernrate zu reduzieren. Da die Verluste in Epoche 9 weiterhin abnehmen, könnte diese Einschränkung durch die Nutzung von mehr Epochen umgangen werden, was zu einer weiteren Verbesserung der Modellleistung führen würde. Die Effektivität der gewählten Lernrate könnte besser beurteilt werden, um eventuell eine stabilere Leistung zu erreichen. Darüber hinaus hat die Modellleistung durch die Einführung zusätzlicher Transformationen zugenommen, da das Modell robuster gegenüber mehr Variationen in den Daten geworden ist.

Eine Betrachtung der Average Precision beider Klassen in der EMD-Datei offenbart, dass diese – trotz Datenaugmentation – ungleich verteilt ist. In Klasse 1 (Photovoltaikanlagen) wurde ein Wert von 0.2375 ermittelt, wohingegen der Wert für Klasse 2 (Solarthermieranlagen) erneut 0 beträgt. Dies lässt darauf schließen, dass das Modell Schwierigkeiten hat, Klasse 2 korrekt zu identifizieren.

```

64     "average_precision_score": {
65         "Class_1": 0.23753976670201485,
66         "Class_2": 0.0
67     },

```

Abb. 86: Average Precision-Ergebnis für beide Klassen (MaskRCNN, 10 Epochen, mit Transformationen)
(Eigene Darstellung, erstellt mit Notepad++ 7.9)

Trotz globaler Datenaugmentation brachte diese Maßnahme für Klasse 2 keine Verbesserung. Mögliche Ursachen könnten erneut dieselben wie beim Training ohne Transformationen sein.

6.7.4.3 Hyperparameter-Tuning für Mask R-CNN: Training über 10 Epochen inkl. Transformationen und Lernrate von 0.001

epoch	train_loss	valid_loss	average_precision	time
0	0.974720	1.032934	0.083661	1:19:00
1	0.818606	0.880140	0.051296	1:55:25
2	0.713547	0.855348	0.046385	2:11:24
3	0.731615	0.779019	0.089819	2:00:20
4	0.654905	0.750436	0.148386	2:33:54
5	0.701169	0.691008	0.204491	1:10:46
6	0.599096	0.690478	0.190840	3:33:53
7	0.647223	0.649458	0.214506	1:06:02
8	0.599440	0.646831	0.243806	1:19:53
9	0.568297	0.650384	0.244188	1:10:11

Abb. 87: MaskRCNN-Modellergebnis nach Training mit Datensatz 1 (10 Epochen, mit Transformationen, Lernrate)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Die Trainingszeit pro Epoche variiert zwischen ca. 1 Stunde und 3,5 Stunden, was auf unterschiedlich lange Backpropagation-Schritte hindeuten kann. In den 10 Epochen zeigen beide Verlustwerte eine klare Abwärtsbewegung. Der Trainingsverlust startet bei 0.9747 und fällt auf 0.5683, während der Validierungsverlust bei 1.0329 beginnt und auf 0.6504 sinkt. Der Trainingsverlust bleibt dabei durchgehend niedriger als der Validierungsverlust, was normal ist. Am Ende stabilisiert sich der Validierungsverlust, was darauf hinweist, dass das Modell nicht stark überanpasst. Die AP startet sehr niedrig bei 0.0837, steigt aber auf 0.2441. Anfangs ist sie sehr gering, was für Mask R-CNN gängig sein kann, da das Modell in den ersten Epochen oft Probleme mit der Objektsegmentierung hat. Die AP-Werte steigen jedoch mit der Zeit. Dennoch bleibt das Endergebnis bei 0.2442 vergleichsweise niedrig.

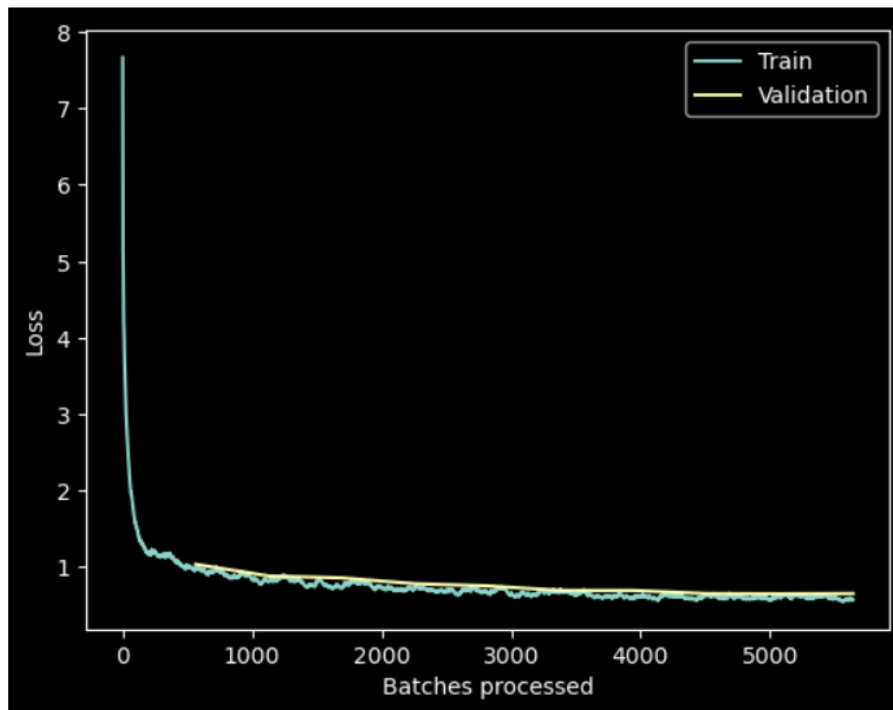


Abb. 88: Verlustdiagramm von MaskRCNN von Datensatz 1 (30 Epochen, ohne Transformationen, Lernrate)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

- Der Verlust sinkt sehr schnell in den ersten 1000 Batches.
- Danach folgt eine langsamere, aber konstante Reduktion des Verlustes. Die Verbesserungen sind nicht mehr so drastisch wie am Anfang, was jedoch ein gutes Zeichen dafür ist, dass das Modell nicht zu früh auf einem Plateau stehen bleibt, sondern weiterlernt.
- Die beiden Kurven (für Training und Validierung) verlaufen sehr nah beieinander, was darauf hindeutet, dass es keine starken Schwankungen oder Instabilitäten im Training gibt und kein starkes Overfitting auftritt.

Das MaskRCNN-Modell zeigt eine stabile Lernkurve, der Verlust reduziert sich konstant, und es tritt kein starkes Overfitting auf. Allerdings bleibt die Average Precision relativ niedrig, was darauf hindeutet, dass das Modell noch weiter trainiert oder optimiert werden könnte. Da sich der Validierungsverlust noch nicht vollständig stabilisiert hat, könnte eine längere Trainingszeit (z. B. 20 Epochen oder mehr) die AP verbessern.

Dieses Mal wurde in der EMD-Datei auch eine Average Precision für Klasse 2 berechnet. Für Klasse 1 (Photovoltaikanlagen) wurde ein Wert von 0.4362 erzielt, während der Wert für Klasse 2 (Solarthermieranlagen) bei 0.0260 liegt. Die Anpassung der Lernrate hatte lediglich einen minimalen positiven Effekt, da dieser Wert weiterhin sehr niedrig ist und das Modell nach wie vor Schwierigkeiten hat, diese Klasse korrekt zu erkennen. Dennoch zeigt dieses Ergebnis, dass das Modell mit weiterem Hyperparameter-Tuning verbessert werden kann.

```

64     "average_precision_score": {
65         "Class_1": 0.436235708003518,
66         "Class_2": 0.025974025974025976
67     },

```

Abb. 89: Average Precision-Ergebnis für beide Klassen (MaskRCNN, 10 Epochen, mit Transformationen, Lernrate)
(Eigene Darstellung, erstellt mit Notepad++ 7.9)

6.7.5 Zusätzliche Erhebung von Informationen zur Grafikkarte

Da die GPU neben den Berechnungen auf Hochtouren läuft, wurde in Erwägung gezogen, mithilfe der Bibliotheken `subprocess`, `time` und `threading` deren Auslastung in einem separaten Thread zu überwachen, in dem die Funktion `monitor_gpu_usage()` ausgeführt wird. Dieser Thread ist als Daemon-Thread konfiguriert, wodurch er automatisch beendet wird, sobald das Hauptprogramm endet. Um präziser zu sein, wird durch den Systembefehl `nvidia-smi` sichergestellt, dass die abgefragten GPU-Informationen (z. B. zu aktiven Prozessen, Auslastung, Speicherverbrauch etc.) in regelmäßigen, frei definierbaren Abständen in eine Logdatei geschrieben werden. Diese Funktion läuft in einer Dauerschleife, bis der Thread beendet wird.

```

22 *****
23 Neuer GPU-Eintrag:
24 Thu Nov 21 10:14:07 2024
25 +-----+
26 | NVIDIA-SMI 555.85                Driver Version: 555.85          CUDA Version: 12.5         |
27 |-----+-----+-----+
28 | GPU   Name                               Driver-Model | Bus-Id          Disp.A | Volatile Uncorr. ECC |
29 | Fan  Temp  Perf              Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M. |
30 |                                           | MIG M.         |
31 |=====+=====+=====+
32 |    0  NVIDIA GeForce GTX 1650        WDDM        | 00000000:01:00:0 Off |            N/A       |
33 | N/A   54C    P0              15W /  50W | 3858MiB / 4096MiB |    100%    Default   |
34 |                                           |                  | N/A         |
35 +-----+-----+-----+
36 |
37 +-----+-----+-----+
38 | Processes:
39 | GPU   GI    CI          PID    Type    Process name                        GPU Memory
40 |   ID   ID                                 Usage
41 |=====+=====+=====+
42 |    0   N/A   N/A         8692    C+G    ... Files\ArcGIS\Pro\bin\ArcGISPro.exe  N/A
43 +-----+-----+-----+

```

Abb. 90: Ermittlung von GPU-Informationen
(Eigene Darstellung, erstellt mit Notepad++ 7.9)

7 Trainingsresultate – Erkennung der Solaranlagen

Unterhalb findet sich eine tabellarische Gegenüberstellung, um die Modellergebnisse übersichtlich darzustellen. Die jeweils farblich gleich gehaltenen Zellen zeigen die Modellpaare, die miteinander verglichen werden. Für den Vergleich raten sich ähnliche Eigenschaften wie die Epochenanzahl und ob Transformationen verwendet wurden oder nicht. Ausschlaggebend ist auch die Average Precision (AP), auf deren Endwert bei der Wahl eher berücksichtigt wird. Auch ein niedriger Validierungsverlust ist ein guter Indikator für eine gute Generalisierungsfähigkeit des Modells.

Modell	Datensatz	Epochen	Transform.	Lernrate	AP	Valid Loss	Train Loss
FRCNN	1	10	Nein	Nein	0.6007	0.4999	0.4225
FRCNN	1	10	Ja	Nein	0.6295	0.5063	0.4981
FRCNN	1	10	Ja	Ja	0.6188	0.5193	0.4944
FRCNN	1	30	Nein	Nein	0.6386	0.4684	0.3794
FRCNN	1	30	Ja	Nein	0.5118	0.5802	0.5099
FRCNN	1	30	Ja	Ja	0.6514	0.4887	0.4705
MRCNN	1	10	Nein	Nein	0.0619	1.2449	1.4622
MRCNN	1	10	Ja	Nein	0.0508	1.1595	1.3182
MRCNN	1	30	Nein	Nein	0.1846	1.1886	1.3048
MRCNN	1	30	Ja	Ja	0.3498	1.0453	1.1240
FRCNN	Kombiniert	10	Nein	Nein	0.5522	0.3586	0.2963
FRCNN	Kombiniert	10	Ja	Nein	0.5700	0.3619	0.3623
FRCNN	Kombiniert	10	Ja	Ja	0.5688	0.3750	0.3631
MRCNN	Kombiniert	10	Nein	Nein	0.1077	0.9238	0.9413
MRCNN	Kombiniert	10	Ja	Nein	0.1360	0.8291	0.7480
MRCNN	Kombiniert	10	Ja	Ja	0.2441	0.6504	0.5683

Tab. 13: Gegenüberstellung der Modellergebnisse

Die Average Precision (AP) misst, wie gut das Modell Vorhersagen macht. Ein höherer AP-Wert bedeutet, dass das Modell präzisere Erkennungen liefert. Der Validierungsverlust (Valid Loss) gibt an, wie gut das Modell auf den Validierungsdaten generalisiert. Ein niedriger Wert deutet auf eine bessere Modellleistung hin. Der Trainingsverlust (Train Loss) sollte im Idealfall niedriger sein als der Validierungsverlust. Ist der Trainingsverlust um einiges kleiner als der Validierungsverlust, könnte Overfitting vorliegen, was heißt, dass das Modell die Trainingsdaten zu gut lernt, sie aber nicht auf neue Daten übertragen kann.

Die Anpassung der Lernrate hat sich bei den Faster R-CNN-Modellen positiv auf das Ergebnis ausgewirkt, weshalb dort mehr Varianten getestet wurden. Beim kombinierten Datensatz von Mask R-CNN wurden die Klassen leider nicht gleichmäßig gut trainiert. Erst die Anpassung der Lernrate auf 0.001 hat einen leichten Anstieg der Average Precision erzielt bei Klasse 2 (Solarthermieanlagen). Trotzdem hat die Lernraten-Anpassung bei Mask R-CNN keinen drastischen bzw. nicht den gleichen Effekt, wie bei den Faster R-CNN-Modellen.

Die neuen Outputs des Trainings bieten eine aktualisierte Verteilung sowie Parameter für beide Klassen. Alle Merkmale (z. B. Klassenrelationen, Datenaugmentation), die in den zusammengeführten Daten enthalten sind, werden direkt in der EMD-Datei des jeweiligen Modells angelernt und spiegeln

somit den kombinierten Gesamtdatensatz wider. Dadurch bleibt die Richtigkeit der Trainingsdaten für beide Klassen erhalten, und das Modell hat somit beide Klassen ohne Inkonsistenzen gelernt und angewendet. Gleichzeitig wurden die Modelleinstellungen und Statistiken vereinheitlicht. Die Erkennung der Features wurde pro Datensatz und Modell mittels **ArcPy** durchgeführt, wobei das Werkzeug **Detect Objects Using Deep Learning** zum Einsatz kam (siehe Anhang E). Folgende Parameter mussten für die Erkennung definiert werden (Esri, 2023a):

Parameter	Erläuterung und Wahl
Input Raster <i>in_raster</i>	Der Raster, der dafür genutzt wird die Objekte zu erkennen/detektieren.
Output Detected Objects <i>out_detected_objects</i>	Die Output-Feature-Class, die die Geometrien (Bounding Boxes) beinhaltet. Diese umhüllen die erkannten Objekte im Input-Raster.
Model Definition <i>in_model_definition</i>	<u>Der Pfad zur Deep Learning-Modelldatei:</u> Außerdem beinhaltet dies auch den Pfad zur angewendeten Rasterfunktion von Python sowie zu anderen Modellparametern wie die Kachelgröße. Zulässige Formate für diesen Parameter sind eine EMD-Datei (Esri Model Definition), ein JSON-String oder ein DLPK-Paket (Deep Learning Model).
Arguments <i>arguments</i>	<u>Die Informationen aus dem Parameter Model Definiton werden angewendet:</u> Die Argumente variieren je nach Architektur der vortrainierten, in ArcGIS trainierten Modelle. Folgend werden Argumente aufgelistet. • padding: Die Pixelanzahl am Bildrahmen aller Kacheln, aus denen Werte bzw. Vorhersagen festgelegt werden, wie der Rand aufgefüllt werden soll. Benachbarte Kacheln werden vereinigt. Maximal kann als Wert die Hälfte der Kachelgröße angegeben werden. Zudem kann das Ergebnis mit einem höheren Padding-Wert geglättet werden. • threshold: Objekterkennungen, die einen höheren Konfidenzwert als der angegebene Schwellenwert haben, werden für das Resultat herangezogen. Zugelassen sind Werte, die zwischen 0 und 1.0 liegen. • batch_size: Hiermit ist die Anzahl der Bildkacheln gemeint, die bei der Erkennung bzw. Inferenz verarbeitet werden. Hier sollte dieselbe Batchgröße verwendet werden wie auch schon während dem Training, da dies vom Speicher der GPU abhängig ist. • exclude_pad_detections: Potenziell abgestumpfte Erkennungen in Kantennähe im Padding-Bereich der Bildchips werden gefiltert, wenn dieses Argument mit "True" festgelegt wird. Das Argument ist für SSD, RetinaNet, YOLOv3, MMDetection und Faster RCNN verfügbar.

Class Value Field <i>class_value_field</i>	<u>Der Name des Klassenwertefeldes in der Input-Feature-Class:</u> Das Feld <i>Classvalue</i> oder <i>Value</i> wird als Feldname verwendet, wenn keines angegeben wird.
Max Overlap Ratio <i>max_overlap_ratio</i>	<u>Maximale Überlappungsrate:</u> Diese Rate gilt für zwei überlappende Features, welche als Verhältnis von Schnittfläche zu Vereinigungsfläche beschrieben ist. Standardmäßig ist 0 eingestellt.
Non Maximum Suppression <i>run_nms</i>	<u>Durchführung einer Non-Maximum-Unterdrückung:</u> Eine NMS wird durchgeführt. Wenn doppelte Objekte identifiziert werden, werden jene Features mit einem niedrigeren Konfidenzwert entfernt.

Tab. 14: Erläuterung der Parameter des Werkzeugs „Detect Objects Using Deep Learning“

7.1 Vorstellung des Testgebiets

Das Testgebiet (siehe Abbildung 91) dient als Input Raster, das in Tabelle 14 erwähnt wird, für das Werkzeug **Detect Objects Using Deep Learning** in ArcGIS Pro. Es wird verwendet, um die Detektionsergebnisse der verschiedenen Modelle zu analysieren und deren Leistung zu bewerten. Das ausgewählte Testgebiet umfasst die Gemeinde Hennersdorf in Niederösterreich, südöstlich von Wien, die sich durch eine hohe Dichte an Photovoltaikanlagen (PVA) auszeichnet. Auch Solarthermieranlagen (STA) sind hier vertreten.



Abb. 91: Testgebiet „Hennersdorf“, eigene Darstellung, erstellt mit ArcGIS Pro 3.2
(Bundesamt für Eich- und Vermessungswesen (BEV), 2023)

Das Testgebiet wurde, analog zu den Trainingsgebieten, aus dem Orthomosaik mittels **Clip Raster** ausgeschnitten. Die Wahl dieses Gebiets basiert auf mehreren Faktoren:

- **Hohe Dichte an Photovoltaikanlagen:** Laut den Daten des Statistik-Atlas „STATatlas“ der Statistik Austria weist Hennersdorf eine überdurchschnittlich hohe Anzahl an Photovoltaikanlagen pro 1000 Einwohner auf. Dies ermöglicht eine umfangreiche Evaluierung der Modelle unter realen Bedingungen (Statistik Austria, 2024).
- **Vielfältige Typen von Photovoltaikanlagen:** Das Gebiet enthält sowohl PVA auf Hausdächern als auch großflächigere Anlagen auf Industrie- und Gewerbeflächen. Diese Mischung stellt sicher, dass das Modell nicht nur für kleinere Anlagen, sondern auch für größere zusammenhängende PV-Flächen getestet wird.
- **Vorkommen von Solarthermieranlagen:** Neben Photovoltaikanlagen sind in diesem Gebiet vereinzelt auch STA vorhanden, was eine zusätzliche Herausforderung für den kombinierten Datensatz (Datensatz 2) darstellt.
- **Geographische Nähe:** Als sekundäres Kriterium wurde die geographische Nähe zum Wohnort berücksichtigt, um eine Vor-Ort-Validierung zu erleichtern.

Bevor die Trainings und Detektionen der einzelnen Modelle und Datensätze durchgeführt wurden, wurde das Testgebiet zunächst digital im GIS auf dem Orthomosaik sowie mit Google Maps betrachtet. Ziel war es, bereits im Vorfeld eine erste Klassifikation der Anlagen vorzunehmen – insbesondere jener, die mit bloßem Auge eindeutig identifizierbar waren. Einige Objekte konnten jedoch nicht sofort einer Klasse zugeordnet werden, weshalb eine Vor-Ort Kontrolle erforderlich war, um zuverlässige **Ground Truth-Daten** zu erfassen. Ziel war es ohnehin, das gesamte Testgebiet systematisch zu überprüfen, um falsche Daten zu vermeiden, die später die Modellbewertung verfälschen könnten.

Vor der eigentlichen Datenerhebung wurde in ArcGIS Pro eine separate Karte erstellt, in der alle eindeutig identifizierbaren Objekte bereits annotiert wurden – unterteilt in PVA und STA. Zusätzlich wurde eine Klasse 0 für nicht eindeutig zuordenbare Objekte angelegt, bei denen unklar war, ob es sich um eine PVA, eine STA oder ein anderes Objekt handelt. Diese Karte samt zugehöriger Feature Class wurde in das Web Mercator-Koordinatensystem (EPSG:3857) umgewandelt, um sie anschließend auf einem mobilen Endgerät via *ArcGIS Field Maps* zu nutzen. So konnten die bereits kartierten Klassen vor Ort überprüft und gegebenenfalls editiert werden.

Nur wenige Objekte waren ursprünglich falsch annotiert. Einige Objekte der Klasse 0 stellten sich beispielsweise nicht als PVA oder STA heraus, sondern als Fenster oder andere Glasflächen. Diese Korrekturen wurden in den Attributdaten vermerkt – dafür wurde ein eigenes Feld dafür im String-Format angelegt, in das die tatsächliche Objektklasse eingetragen wurde, um eine Nachbearbeitung und Korrektur der Objekte im GIS zu ermöglichen. Das Gebiet umfasst insgesamt 219 Features, 160 davon sind PVA und 59 STA.

Unten sind Abbildungen von relevanten Abschnitten aus dem Testgebiet zu sehen, die die annotierten Ground Truth-Daten zeigen.



Abb. 92: Ground Truth-Daten 1/2
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

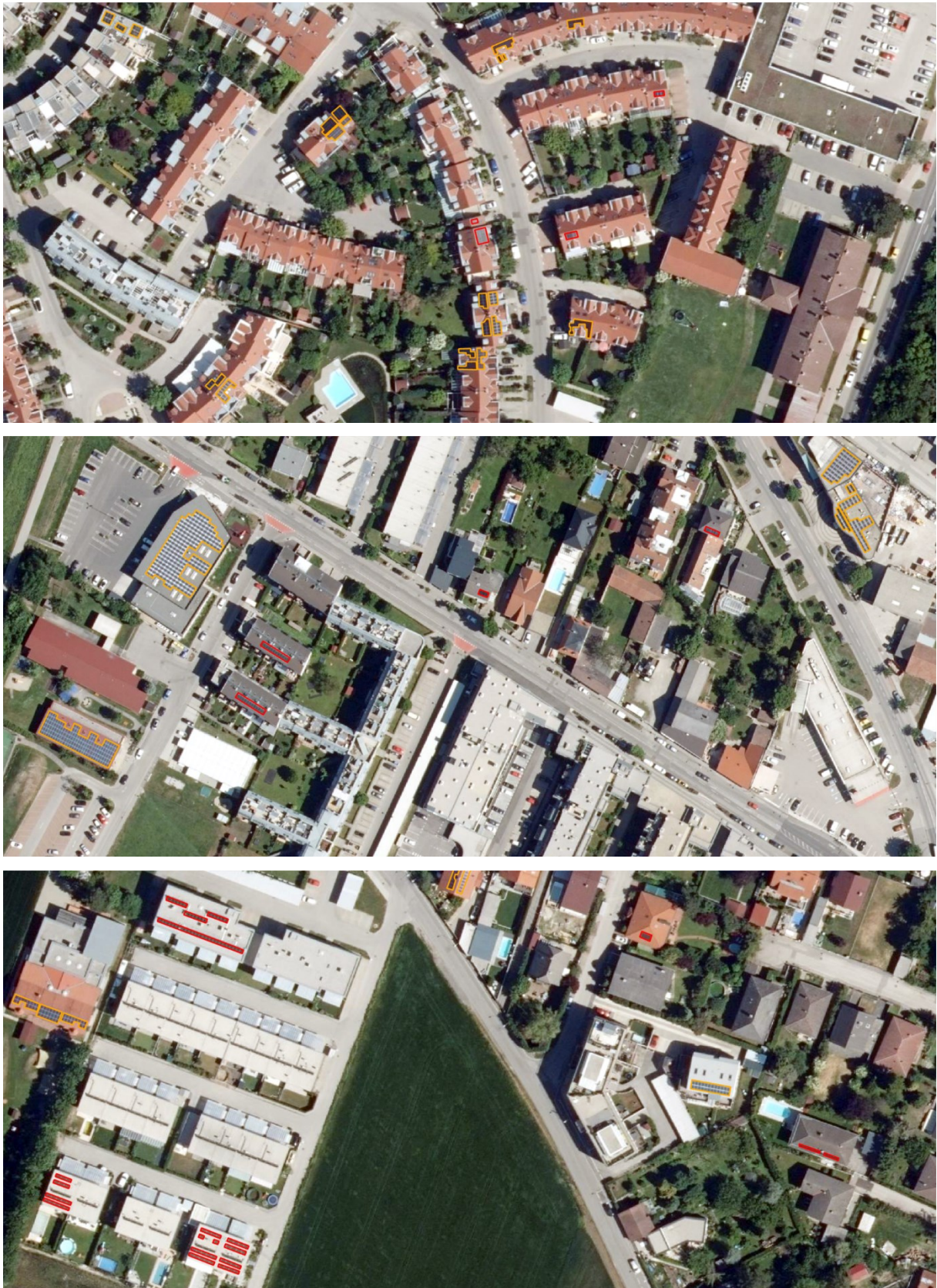


Abb. 93: Ground Truth-Daten 2/2
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

7.1.1 Erste visuelle Erkennungsergebnisse

Eine detaillierte Analyse der Erkennungsergebnisse in ArcGIS Pro zeigt, dass insbesondere die Faster R-CNN-Modelle häufig falsche Objekte als PVA identifizieren. Zu den am häufigsten falsch erkannten

Objekten zählen Lagermaterialien wie Paletten, parkende Autos auf Parkplätzen, Glas- bzw. Gewächshäuser sowie verschiedene Überdachungen aus Aluminium oder Schattenwürfe. Besonders interessant ist, dass rechteckige Muster, die beispielsweise auf Parkplätzen durch Markierungen entstehen, Zebrastreifen oder gar gemähte Flächen in landwirtschaftlichen Feldern fälschlicherweise als PVA interpretiert wurden. Auch Solarthermieranlagen wurden häufig als PVA klassifiziert, was darauf hindeutet, dass das Modell Schwierigkeiten hat, zwischen diesen beiden Anlagentypen zu differenzieren.



Abb. 94: Falsch als Photovoltaikanlage detektierte Objekte (Faster R-CNN)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Trotz der grundsätzlich soliden Erkennungsleistung von Faster R-CNN für PVA wird die Gesamtgenauigkeit durch die hohe Anzahl an Falschpositiven erheblich reduziert. Dies legt nahe, dass das Modell zwar eine starke Sensitivität für die Erkennung von PV-Anlagen besitzt, jedoch an einer unzureichenden Präzision leidet, da es strukturell oder farblich ähnliche Objekte ebenfalls als PVA klassifiziert.



Abb. 95: Richtig als Photovoltaikanlage detektierte Objekte (Faster R-CNN)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Mask R-CNN hingegen zeigt ein anderes Fehlerverhalten: Während das Modell insgesamt deutlich weniger Falscherkennungen produziert, treten Schwächen bei der Maskierung auf. Besonders auffällig ist, dass bei größeren Anlagen die segmentierten Masken oft unvollständig sind oder bei kleinen PVA die Masken gänzlich fehlen. Dies könnte darauf hindeuten, dass die gewählten Hyperparameter oder die Art der Maskierung nicht optimal für die Variabilität der PVA-Flächen im Datensatz sind. Dennoch bleibt festzuhalten, dass Mask R-CNN durch seine geringere Anzahl an Falscherkennungen eine insgesamt höhere Genauigkeit aufweisen sollte.

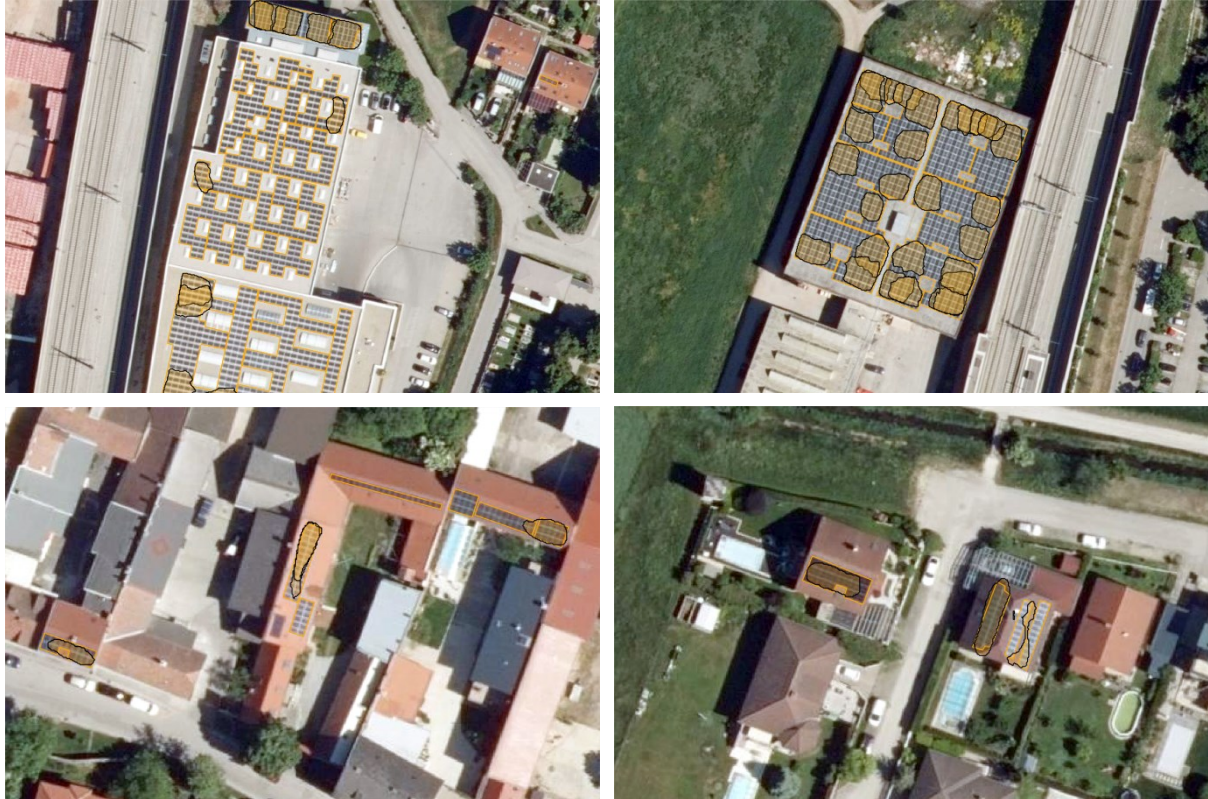


Abb. 96: Unzureichende Erkennung von Photovoltaikanlagen (Mask R-CNN)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Ein besonders interessanter Aspekt ergibt sich aus dem Vergleich der Ergebnisse zwischen Datensatz 1 (nur PVA) und dem kombinierten Datensatz (PVA und STA). Obwohl in letzterem Fall STA explizit als eigene Klasse trainiert wurde, zeigte sich, dass das Hinzufügen der zusätzlichen Klasse nur einen geringen Nutzen brachte.

- Die Erkennungsrate von STA blieb insgesamt sehr niedrig, unabhängig davon, ob Faster R-CNN oder Mask R-CNN verwendet wurde.
- Es gab nur wenige Fälle, in denen Objekte, die zuvor mit Datensatz 1 als PVA klassifiziert wurden, im kombinierten Datensatz korrekt als STA erkannt wurden.
- Das bedeutet, dass das Training einer eigenen Klasse für STA nicht zu einer signifikanten Verbesserung der Erkennungsleistung geführt hat.

Diese Beobachtung könnte darauf hindeuten, dass entweder die Anzahl der STA-Instanzen im Trainingsdatensatz zu gering war, um ein verlässliches Modell zu trainieren, oder dass die Unterscheidungsmerkmale zwischen STA und PVA für das Modell nicht ausreichend klar sind. Eine weitere Möglichkeit ist, dass die Architektur der verwendeten Modelle nicht optimal für diese

spezifische Klassifikationsaufgabe geeignet ist und möglicherweise alternative Netzwerke oder Anpassungen der Trainingsstrategie erforderlich wären.



Abb. 97: Erkennung von Photovoltaikanlagen im kombinierten Datensatz (Mask R-CNN)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)



Abb. 98: Erkennung von Solarthermieranlagen im kombinierten Datensatz (Mask R-CNN)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

7.2 Auswertung der Modellleistung

Um die Leistung von Deep Learning-Modellen bei der Objekterkennung und Segmentierung von Photovoltaik- (PVA) und Solarthermieranlagen (STA) systematisch und objektiv bewerten zu können, ist die Anwendung geeigneter Evaluationsmetriken unerlässlich. Im Verlauf des Trainingsprozesses werden Metriken wie Average Precision (AP), Train Loss (Trainingsverlust) und Validation Loss (Validierungsverlust) zur internen Optimierung des Modells verwendet. Sie liefern erste Hinweise auf die Lernkurve und den Anpassungsgrad an die Trainingsdaten. Allerdings erlauben diese Trainingsmetriken nur eine begrenzte Aussage über die tatsächliche Modellleistung auf neuen, unbekannten Daten. Daher sind aussagekräftigere Validierungsmetriken notwendig, um die Modellgüte im Anwendungskontext differenziert zu bewerten. Eine zentrale Methode zur Beurteilung der Vorhersagegüte stellt dabei die Konfusionsmatrix dar. Diese Matrix gibt einen detaillierten Überblick darüber, wie gut ein Modell zwischen den tatsächlichen Klassen (Ground Truth) und den vorhergesagten Klassen unterscheidet. Sie zeigt nicht nur die korrekt klassifizierte Objekte (True Positives), sondern auch verschiedene Fehlerarten, wie False Positives (falsch positiv erkannte Objekte) und False Negatives (nicht erkannte Objekte), auf. Dadurch erlaubt sie eine differenzierte Analyse der Modellleistung über alle Klassen hinweg. Eine zweite zentrale Beurteilungsmethode der Modelleffizienz ist Intersection over Union (IoU). Sie bewertet die geometrische Übereinstimmung

zwischen den vorhergesagten und den tatsächlichen Objekten (Ground Truth) auf Pixel- oder Flächenniveau.

7.2.1 Precision, Recall und F1-Score

Die Konfusionsmatrix bildet die Grundlage zur Berechnung weiterführender Klassifikationsmetriken wie Precision, Recall und dem darauf basierenden F1-Score. Diese Metriken ermöglichen eine differenzierte Bewertung der Modellleistung hinsichtlich Genauigkeit, Sensitivität und Ausgewogenheit der Vorhersagen über verschiedene Klassen hinweg. Insbesondere bei unausgewogenen Datensätzen oder bei mehreren Klassen kann eine allein auf die Genauigkeit (Accuracy) gestützte Bewertung irreführend sein. Die Auswertung der Konfusionsmatrix stellt daher einen essenziellen Schritt dar, um Stärken und Schwächen der eingesetzten Modelle zu identifizieren und ihre Eignung für den jeweiligen Anwendungsfall fundiert einschätzen zu können.

7.2.1.1 Precision (dt.: Präzision)

Die **Precision** gibt an, wie viele der vom Modell positiv vorhergesagten Objekte tatsächlich korrekt klassifiziert wurden. Ein hoher Precision-Wert bedeutet, dass das Modell wenig falsch-positive Vorhersagen trifft, also nur dann eine Klasse erkennt, wenn sie tatsächlich vorhanden ist.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

7.2.1.2 Recall (dt.: Trefferquote)

Der **Recall** (auch „Sensitivity“ oder „True Positive Rate“) misst den Anteil der tatsächlich vorhandenen Objekte einer Klasse, die auch korrekt erkannt wurden. Die Kombination dieser Metriken erlaubt eine ganzheitliche Bewertung der Modellergebnisse. Während Precision eher die Zuverlässigkeit der Vorhersagen misst, ist Recall ein Maß für die Vollständigkeit der Erkennung. Der F1-Score kombiniert diese beiden Sichtweisen und hilft, ein ausgewogenes Modell zu identifizieren.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

7.2.1.3 F1-Score (dt.: F1-Maß)

Der **F1-Score** stellt einen Kompromiss zwischen Precision und Recall dar und ist ihr harmonisches Mittel. Der F1-Score ist besonders nützlich, wenn ein Gleichgewicht zwischen Precision und Recall angestrebt wird oder wenn das Klassenverhältnis unausgewogen ist. Ein Modell mit hoher Precision und geringem Recall (oder umgekehrt) wird einen moderaten F1-Score aufweisen. Je näher der F1-Score an 1 liegt, desto besser ist die Gesamtleistung des Modells hinsichtlich der erkannten Klasse.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

7.2.2 Evaluierung der Modellleistung mittels Konfusionsmatrix

Zur Beurteilung der Modellleistung wurden die Metriken Precision, Recall und F1-Score für vier verschiedene Modellvarianten berechnet: Mask R-CNN mit kombinierten Klassen (PVA und STA), Mask R-CNN mit einer Klasse (PVA), sowie Faster R-CNN in beiden Varianten. Darüber hinaus wurde die Klasse „none“ als Kategorie für Falschklassifizierungen berücksichtigt, sofern ein Objekt erkannt wurde, das im Ground Truth nicht existiert. Erfolgt dies mithilfe der Bibliotheken **pandas**, **Seaborn**,

Matplotlib und **sklearn.metrics**. Im Anhang F befindet sich das Skript, mit dem die Konfusionsmatrizen erstellt wurden (Scikit-learn Developers, o.J.; The Matplotlib Development Team, o.J.; The pandas Development Team, 2024; Waskom, o.J.).

7.2.2.1 Mask R-CNN – Kombiniertes Datensatz

Die Ergebnisse zeigen, dass das Modell eine vergleichsweise hohe Precision für die Klasse PVA (0.88) erreicht. Das bedeutet, dass die meisten als PVA klassifizierten Objekte tatsächlich korrekt identifiziert wurden. Gleichzeitig ist der Recall mit 0.56 jedoch deutlich niedriger, was darauf hinweist, dass eine beträchtliche Anzahl an tatsächlichen PVA-Anlagen nicht erkannt wurde (False Negatives). Für die Klasse STA liegen sowohl Precision (0.50) als auch Recall (0.10) auf niedrigem Niveau. Der F1-Score bestätigt diese Tendenz (für PVA 0.69 und für STA 0.16).

Klasse	Precision	Recall	F1-Score
PVA	0.88	0.56	0.69
STA	0.50	0.10	0.16

Tab. 15: Ergebnisse der Klassifikationsmetriken für Mask R-CNN (Kombinierter Datensatz)

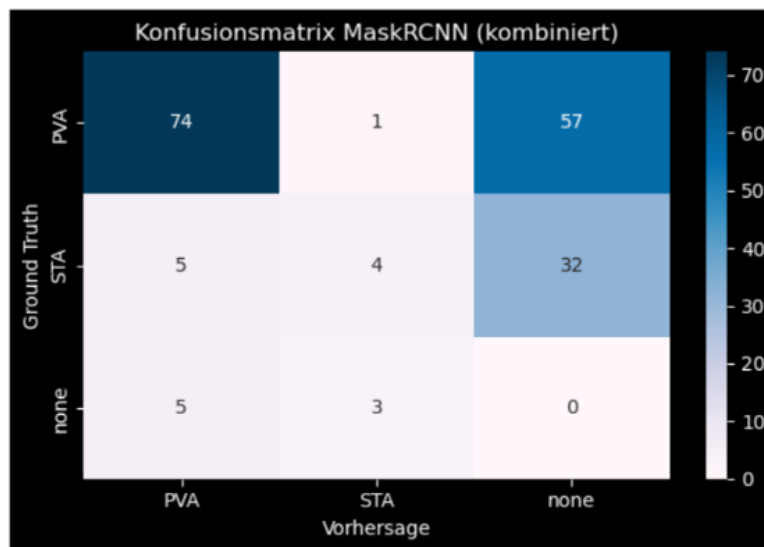


Abb. 99: Konfusionsmatrix des Modells Mask-RCNN (Kombinierter Datensatz)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Das Modell erkennt PVA-Anlagen relativ zuverlässig (74 richtig erkannt), verfehlt jedoch viele STA-Anlagen (32 nicht erkannt) und erkennt zahlreiche Objekte gar nicht (57 PVA und 32 STA), was durch die hohe Anzahl an „none“-Zuweisungen bestätigt wird - hier gibt es also viele False Negatives. Möglicherweise leidet das Modell unter einer unausgeglichene Klassenverteilung oder unzureichender Unterscheidbarkeit beider Klassen. Insgesamt ist es eine solide Precision, aber ein schwacher Recall, da das Modell zu wenig erkennt.

7.2.2.2 Mask R-CNN – Datensatz 1

Bei diesem Modell liegt die Precision für PVA bei 0.87, jedoch fällt der Recall mit 0.58 erneut deutlich geringer aus. Der F1-Score liegt bei 0.70. Objekte der Klasse „none“ – d. h. falsch erkannte Objekte, die in den Ground-Truth-Daten nicht vorhanden waren – wurden zu 0% korrekt identifiziert.

Klasse	Precision	Recall	F1-Score
PVA	0.87	0.58	0.70

Tab. 16: Ergebnisse der Klassifikationsmetriken für Mask R-CNN (Datensatz 1)

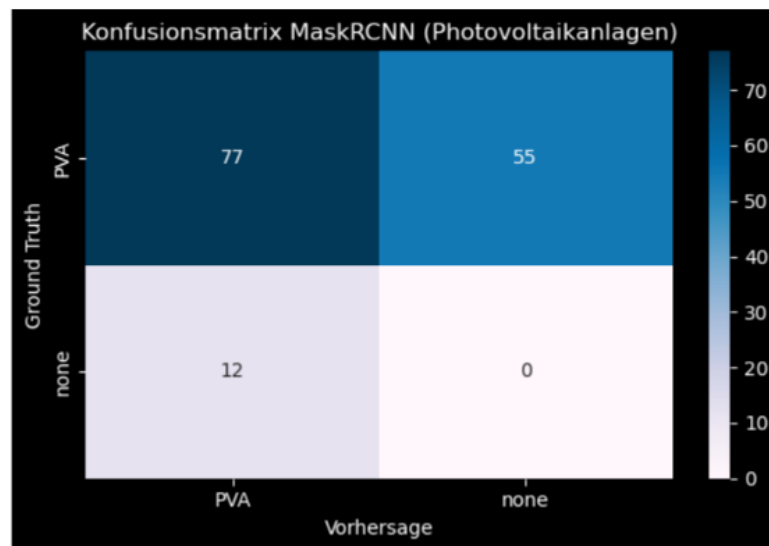


Abb. 100: Konfusionsmatrix des Modells Mask-RCNN (Datensatz 1)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Das Modell klassifiziert PVA-Anlagen präzise, zeigt jedoch erneut Schwächen bei der vollständigen Erkennung (Recall), denn viele PVA wurden übersehen (False Negatives). Die hohe Precision zeigt, dass das Modell nur wenige False Positives generiert. Die fehlerhafte Erkennung von „none“ als PVA ist gering (nur 12-mal).

7.2.2.3 Faster R-CNN – Kombiniertes Datensatz

Die Precision für PVA fällt mit 0.50 deutlich ab, der Recall ist jedoch mit 0.86 deutlich höher als bei Mask R-CNN. Das deutet darauf hin, dass das Modell viele tatsächliche PV-Anlagen erkennt, dabei aber auch zahlreiche Fehlklassifikationen erzeugt. Für STA liegt die Precision bei lediglich 0.17, der Recall hingegen bei 0.59. Der F1-Score ist für beide Klassen vergleichsweise niedrig (für PVA 0.64 und für STA 0.26). Die Klasse „none“ wurde in über 200 Fällen vom Modell fälschlich als PVA oder STA klassifiziert.

Klasse	Precision	Recall	F1-Score
PVA	0.50	0.86	0.64
STA	0.17	0.59	0.26

Tab. 17: Ergebnisse der Klassifikationsmetriken für Faster R-CNN (Kombinierter Datensatz)

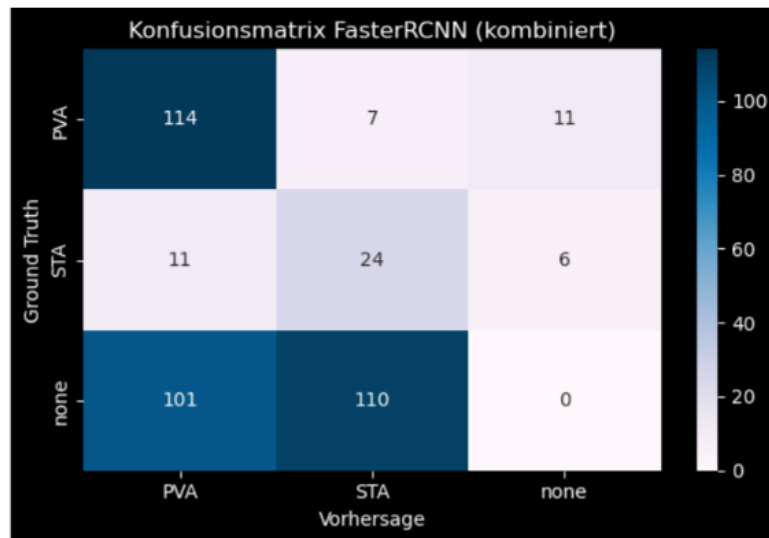


Abb. 101: Konfusionsmatrix des Modells Faster-RCNN (Kombinierter Datensatz)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Faster R-CNN erkennt somit einen größeren Anteil der tatsächlichen Objekte, geht dabei jedoch zu Lasten der Vorhersagegenauigkeit – viele von den PVAs sind falsch erkannt (niedrige Precision), der Recall ist hoch. STA wird deutlich besser erkannt als bei Mask R-CNN, aber es ist immer noch ein schwaches Ergebnis. Das Modell neigt generell zu einer Übergeneralisierung, viele Objekte werden fälschlich klassifiziert, was die Anzahl der False Positives erhöht.

7.2.2.4 Faster R-CNN – Datensatz 1

Dieses Faster-RCNN Modell erreicht einen hohen Recall von 0.89 für die Klasse PVA, bei gleichzeitig niedriger Precision von 0.45 im Gegensatz zum vorherigen Modell. Dies bedeutet, dass das Modell fast alle tatsächlichen PVA-Anlagen erkennt, aber dabei eine große Anzahl an Falsch-Positiven erzeugt. Der F1-Score liegt somit bei 0.60.

Klasse	Precision	Recall	F1-Score
PVA	0.45	0.89	0.60

Tab. 18: Ergebnisse der Klassifikationsmetriken für Faster R-CNN (Datensatz 1)

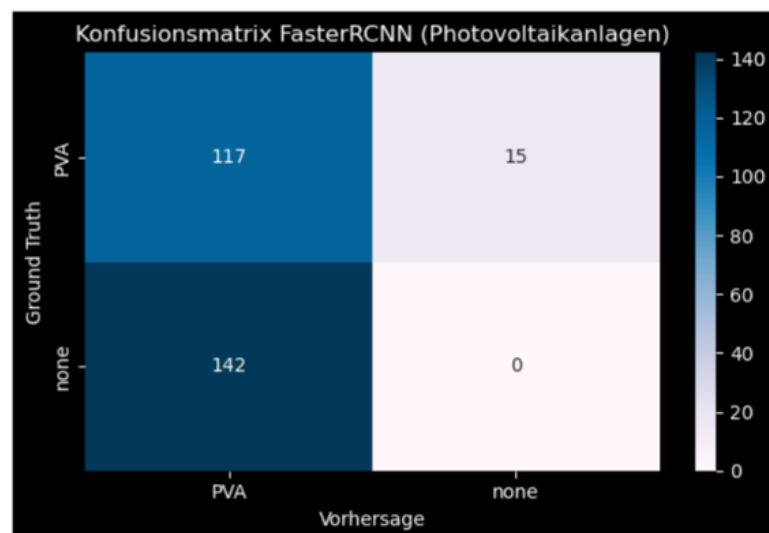


Abb. 102: Konfusionsmatrix des Modells Faster-RCNN (Datensatz 1)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Diese Modellvariante ist besonders sensitiv (hoher Recall), erkennt also fast alle relevanten PVA-Objekte, aber auf Kosten einer erhöhten Fehlerquote (142 fehlerhafte Erkennungen). In Anwendungen, bei denen Vollständigkeit entscheidend ist, kann aber ein solches Verhalten gewünscht sein (z. B. zur Vorab-Detektion mit anschließender manueller Prüfung). „None“ wird gar nicht erkannt, sondern fälschlich als PVA identifiziert.

Das Balkendiagramm (siehe Abbildung 103) vergleicht die Klassifikationsmetriken Precision, Recall und F1-Score für die Klasse PVA über alle vier Modell-Varianten hinweg. Das Skript zur Erstellung der Darstellung ist in Anhang G aufgeführt (NumPy Developers, o.J.; The Matplotlib Development Team, o.J.).

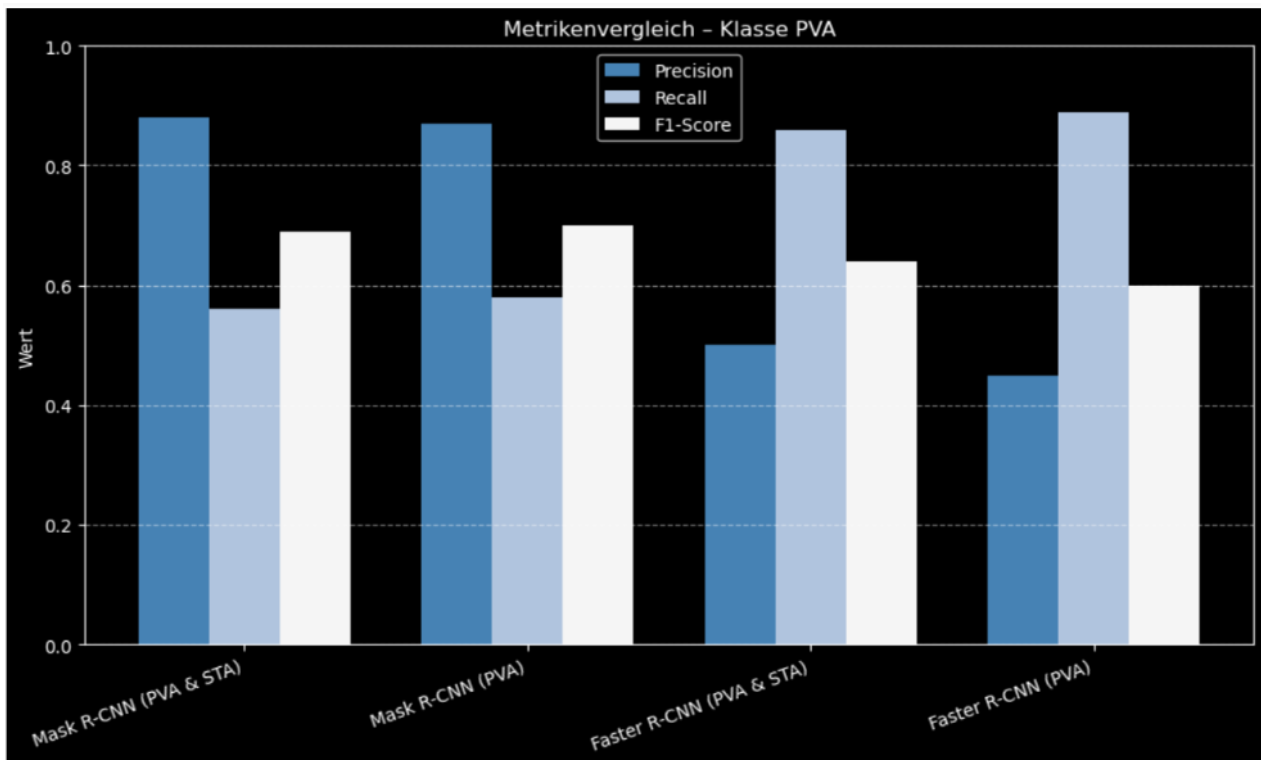


Abb. 103: Darstellung vom Vergleich der drei Metriken, bezogen auf PVA
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Tabelle 19 zeigt dieses Mal zusammenfassend alle Gesamtergebnisse der angewendeten Metriken:

Modell	Precision PVA	Recall PVA	F1-Score PVA	Precision STA	Recall STA	F1-Score STA
Mask R-CNN PVA & STA	0.88	0.56	0.69	0.50	0.10	0.16
Mask R-CNN PVA	0.87	0.58	0.70	---	---	---
Faster R-CNN PVA & STA	0.50	0.86	0.64	0.17	0.59	0.26
Faster R-CNN PVA	0.45	0.89	0.60	---	---	---

Tab. 19: Gesamtergebnis der Klassifikationsmetriken (für alle Modelle)

Die beiden Modelle Mask-RCNN und Faster R-CNN zeigen unterschiedliche Stärken: Mask R-CNN liefert tendenziell präzisere Vorhersagen, erkennt jedoch weniger Objekte. Sie sind besser geeignet, wenn False Positives als kritisch erachtet werden. Faster R-CNN hingegen zeigt eine höhere Trefferquote (Recall), ist aber „aggressiver“ und produziert daher mehr Fehlklassifikationen. Solche Modelle sind eher für Szenarien geeignet, in denen Recall wichtiger ist als Precision. In der Kombination mit beiden Klassen (PVA und STA) sinkt die Gesamtleistung bei beiden Modellen, insbesondere zugunsten von Recall oder Precision, aber nicht beidem gleichzeitig.

Die Wahl des optimalen Modells hängt daher stark vom Anwendungszweck ab: Für automatisierte, qualitätsgesicherte Anwendungen, in denen falsche Klassifikationen vermieden werden sollen, ist Mask R-CNN zu bevorzugen. Für vollständige Erkennung mit nachgelagerter Kontrolle, z. B. in der Bestandserhebung, ist Faster R-CNN aufgrund seines hohen Recalls geeigneter.

7.2.3 Evaluierung der Modellleistung mit Intersection over Union (IoU)

Intersection over Union (IoU) ist eine gängige Bewertungsmetrik, um die Genauigkeit von Objektdetektions- und Segmentierungsmodellen zu bewerten. IoU berechnet das Verhältnis der Schnittfläche zwischen der vorhergesagten und der tatsächlichen Bounding Box oder Segmentierung (Ground Truth) zur Vereinigungsfläche dieser beiden Flächen. Die mathematische Definition lautet:

$$IoU = \frac{\text{Fläche der Überlappung}}{\text{Fläche der Vereinigung}}$$

IoU-Werte liegen stets im Bereich zwischen 0 und 1, wobei 0 bedeutet, dass keine Überlappung zwischen Vorhersage und Ground Truth existiert, und 1, dass die vorhergesagte Region exakt mit der tatsächlichen Region übereinstimmt. Ein hoher IoU-Wert zeigt also eine präzise Vorhersage, während ein niedriger Wert auf eine ungenaue Modellleistung hindeutet.

IoU bietet eine kontinuierliche **Bewertung der räumlichen Übereinstimmung** zwischen Vorhersage und tatsächlichem Wert (Ground Truth) - daher ist es nicht falsch, eine zusätzliche Metrik auszuwählen, um die räumliche Genauigkeit detaillierter zu analysieren. Dies ist insbesondere für die Segmentierung von Photovoltaikanlagen entscheidend, da eine exakte Abgrenzung der Objekte von großer Bedeutung sein kann. In dieser Arbeit wurde der IoU-Wert für komplette Klassen (Photovoltaik- und Solarthermieranlagen) berechnet, anstatt ihn nur auf Basis einzelner Feature-Instanzen zu bestimmen. Weil es pro Modell nur eine IoU-Messung gibt, ist es nicht möglich, Verteilungen oder Streuungen zu untersuchen.

7.2.3.1 Durchführung der Bewertung in ArcGIS Pro

Zur Evaluierung der Modelle kamen die Werkzeuge **Detect Objects Using Deep Learning**, **Intersect** und **Union** für die IoU-Berechnung sowie **Minimum Bounding Geometry** in ArcGIS Pro zum Einsatz. Hierbei kamen die trainierten Modelle auf den Testdaten zum Einsatz, und eine Gegenüberstellung der Ergebnisse mit den Ground Truth-Daten erfolgte. Alle beteiligten Daten befinden sich als Feature Classes in der File Geodatabase. Da Mask R-CNN Segmentierungsergebnisse als Binärmasken ausgibt, musste für die IoU-Berechnung eine Umwandlung in Bounding Boxes erfolgen. Dies vereinfacht die Berechnung, da IoU ursprünglich für Bounding Box-basierte Objektdetektion entwickelt wurde. Die Transformation der Masken in umschließende Rechtecke (Bounding Boxes) ermöglichte eine konsistente Bewertung über beide Modelltypen hinweg (Faster R-CNN und Mask R-CNN).

Die Werkzeuge wurden automatisiert mittels **ArcPy** durchgeführt. Ein Skript dazu befindet sich in Anhang H (Esri, 2023c, 2023e, 2023g). Die berechneten IoU-Werte wurden anschließend extrahiert und in den nächsten Kapiteln für die Analyse als Balkendiagramme dargestellt (siehe Anhang I), um die Leistung der Modelle zu vergleichen (NumPy Developers, o.J.; The Matplotlib Development Team, o.J.). Die IoU-Werte der Modelle wurden auf zwei verschiedenen Datensätzen ausgewertet: Datensatz 1 und kombinierter Datensatz.

Ergebnisse für Datensatz 1 (PVA):

Modell	IoU
Faster R-CNN	0.2763
Mask R-CNN	0.4710

Tab. 20: IoU-Ergebnisse (Datensätze 1)

Ergebnisse für kombinierten Datensatz (PVA und STA):

Modell	IoU für PVA	IoU für STA
Faster R-CNN	0.3510	0.0725
Mask R-CNN	0.6172	0.0524

Tab. 21: IoU-Ergebnisse (Kombinierte Datensätze)

7.2.3.2 Vergleich der IoU-Werte auf dem Datensatz 1

Das Balkendiagramm zeigt die IoU-Werte der Modelle Faster R-CNN und Mask R-CNN, die auf Datensatz 1 trainiert wurden, welcher ausschließlich Photovoltaikanlagen (PVA) enthält. Da in diesem Fall nur eine Klasse vorhanden ist, war keine Auftrennung der Ergebnisse in einzelne Klassen erforderlich. Aus den Resultaten geht hervor, dass Mask R-CNN eine höhere IoU erreicht als Faster R-CNN. Dies deutet darauf hin, dass die Segmentierungsqualität bei Mask R-CNN besser ist. Dies entspricht den Erwartungen, da Mask R-CNN neben der Objekterkennung auch eine pixelgenaue Segmentierung vornimmt, was zu einer genaueren Abgrenzung der Anlagen führt. Die Abweichungen in den IoU-Werten können unter anderem auf die folgenden Faktoren zurückzuführen sein:

- Mask R-CNN erstellt Masken für die Objekte, während Faster R-CNN nur Bounding Boxes liefert, die eine weniger präzise Abgrenzung der Objekte bieten.
- Die IoU-Werte von Faster R-CNN werden durch potenzielle Mehrfachdetektionen oder falsch identifizierte Objekte negativ beeinflusst.
- Auch die Trainingsstrategie und die Hyperparameter könnten dazu beigetragen haben, dass Mask R-CNN insgesamt eine bessere Datenanpassung erreicht hat.

Die Ergebnisse zeigen insgesamt, dass Mask R-CNN für die Segmentierung von PVA in diesem Datensatz besser geeignet ist als Faster R-CNN. Dies bestätigt die Annahme, dass ein Modell mit expliziter Maskenbildung eine höhere Präzision bei der Abgrenzung von Objekten bietet.

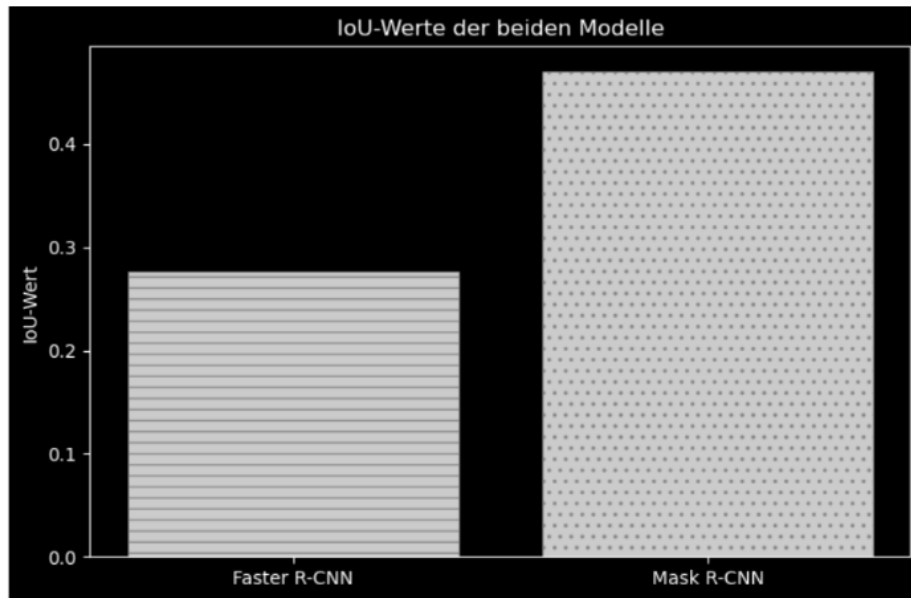


Abb. 104: Darstellung der IoU-Ergebnisse der Modelle im Vergleich (Datensatz 1)
 (Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

7.2.3.3 Vergleich der IoU-Werte auf dem kombinierten Datensatz

Das Balkendiagramm (siehe Abbildung 105) zeigt die IoU-Werte der Modelle Faster R-CNN und Mask R-CNN, die auf einem kombinierten Datensatz trainiert wurden. In diesem Datensatz sind sowohl Photovoltaikanlagen (PVA) als auch Solarthermieranlagen (STA) enthalten. Die IoU-Werte für die beiden Klassen wurden getrennt dargestellt, um eine bessere Analyse zu ermöglichen. Die Resultate zeigen, dass die Genauigkeit der Segmentierung für PVA (hellblaue Balken) deutlich höher ist als für STA (mittelblaue Balken), unabhängig vom eingesetzten Modell. Der IoU-Wert für Solarthermieranlagen (STA) fällt besonders auf, da er gering ist. Weitere Unterschiede zwischen den Modellen sind:

- Die IoU-Werte von Mask R-CNN sind insgesamt höher als die von Faster R-CNN, vor allem für PVA. Der Grund dafür ist, dass Mask R-CNN neben der Objektdetektion auch eine pixelgenaue Segmentierung vornimmt, was eine genauere Abgrenzung der Anlagen ermöglicht.
- Durch die Kombination mehrerer Klassen könnten Inhomogenitäten in den Features entstanden sein, was dem Modell Schwierigkeiten bereitet haben kann, einheitliche Features zu extrahieren.
- Bei beiden Modellen ist die Erkennung von STA unzureichend, was möglicherweise auf Übersegmentierung oder Mehrfachdetektion zurückzuführen ist. Dadurch könnte es passieren, dass das Modell die Umrisse von Solarthermieranlagen nicht richtig erfasst oder falsche bzw. keine Objekte als STA klassifiziert.

Mask R-CNN weist im kombinierten Datensatz (PVA und STA) für PVA die höchste IoU auf und erzielt somit die beste Erkennungsleistung.

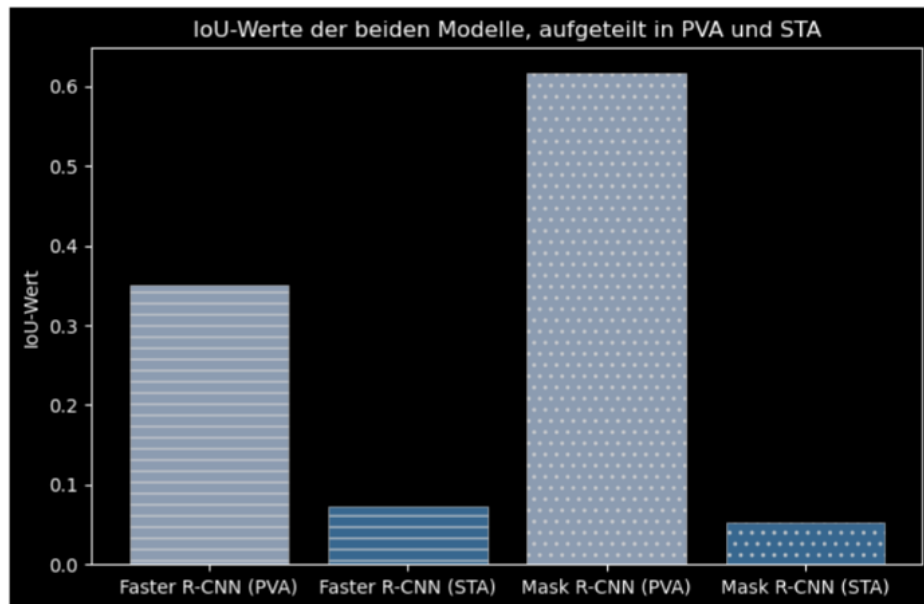


Abb. 105: Darstellung der IoU-Ergebnisse der Modelle im Vergleich (Kombinierter Datensatz)
(Eigene Darstellung, erstellt mit ArcGIS Pro 3.2)

Die Evaluierung der Modellleistung zeigt, dass Mask R-CNN in beiden Szenarien für die Klasse PVA die besten Ergebnisse erzielt hat. Im Datensatz 1 (nur PVA) erzielte Mask R-CNN mit einer höheren IoU deutlich bessere Resultate als Faster R-CNN, was darauf hinweist, dass das Modell ideal für diese Klasse geeignet ist. Im kombinierten Datensatz liegt der IoU-Wert von Mask R-CNN bei PVA mit 0.6172 noch einmal über dem Wert aus Datensatz 1 (0.4710). Die geometrische Genauigkeit der Erkennung von Photovoltaikanlagen ist in diesem Datensatz mit Mask R-CNN somit am höchsten. Es fällt jedoch auf, dass die IoU-Werte für STA bei beiden Modellen sehr gering sind. Die IoU-Werte von Faster R-CNN und Mask R-CNN für STA liegen jeweils unter 0.1, was auf eine geringe Erkennungsleistung beider Modelle in dieser Klasse hindeutet.

7.3 Modellbewertung und Ergebnisinterpretation

Für die gründliche Beurteilung der Modellergebnisse wurden zwei wesentliche Metriken verwendet: Die Konfusionsmatrix und die Intersection over Union (IoU). Die beiden Methoden fangen verschiedene Facetten der Modellgüte ein. Die Konfusionsmatrix legt den Fokus auf die richtige Klassifizierung auf der Ebene von Objekten und zeigt auf, wie verlässlich ein Modell komplette Objekte oder Features identifiziert. Im Gegensatz dazu bewertet die IoU die geometrische Genauigkeit der Modellvorhersagen, indem sie die Überlappung zwischen den vorhergesagten und den tatsächlichen Objektflächen auf Pixel- bzw. Bounding Box-Ebene quantifiziert.

Durch die Kombination beider Verfahren wird es möglich, das Modellverhalten differenziert zu beurteilen: Welche Modelle identifizieren die Zielobjekte genauer? Welche Fehler treten häufiger auf und worin könnten ihre Ursachen liegen? In den folgenden Unterkapiteln werden diese Fragen detailliert analysiert und die Ergebnisse im Hinblick auf die Stärken und Schwächen der jeweiligen Modelle interpretiert.

7.3.1 Analyse der Klassifikationsmetriken (Konfusionsmatrix)

Die vier evaluierten Modelle (Mask R-CNN und Faster R-CNN, jeweils auf Einzelklassen- und kombinierten Datensätzen) zeigen unterschiedliche Stärken und Schwächen im Hinblick auf Precision, Recall und F1-Score:

Mask R-CNN (kombinierter Datensatz):

- Höchste Precision für PVA (0.88), jedoch schwacher Recall für STA (0.10).
- Das Modell erkennt PVA mit hoher Präzision, hat jedoch Probleme bei der korrekten Identifizierung von STA.
- Potenzielle Ursache: Ungleichgewicht in der Klassenverteilung oder niedrige Trennschärfe zwischen PVA und STA.

Mask R-CNN (Datensatz 1):

- Gute Precision (0.87) bei einer deutlich besseren Balance im Vergleich zu STA.
- Einige Objekte werden fälschlicherweise als PVA (False Positives) klassifiziert, was durch die „none“-Klasse deutlich wird.

Faster R-CNN (kombinierter Datensatz):

- Deutlich schlechtere Leistung: Precision beträgt 0.50 (PVA) und 0.17 (STA), es gibt sehr viele Falschklassifikationen.
- Das Modell wirkt weniger robust in Bezug auf die Generalisierung und Klassentrennung.

Faster R-CNN (Datensatz 1):

- Mäßige Precision (0.45), da viele Objekte fälschlich erkannt werden. Es gibt einen hohen Anteil von False Positives und „none“.
- Der Recall-Wert für PVA beträgt 0.89, was zeigt, dass viele PVA-Objekte erkannt wurden – allerdings auf Kosten der Genauigkeit.

7.3.2 Analyse der Modelle anhand von IoU-Werten

Für alle Modelle wurde zusätzlich ein aggregierter IoU-Wert bestimmt, der die geometrische Überlappung zwischen vorhergesagten und tatsächlichen Objekten widerspiegelt. Die IoU-Werte bestätigen die Ergebnisse der Konfusionsmatrix: Mask R-CNN erreicht deutlich höhere geometrische Genauigkeiten im Vergleich zu Faster R-CNN. Darüber hinaus zeigen die Ergebnisse, dass Mask R-CNN auch bei gesteigerter Klassendiversität (PVA und STA) hohe geometrische Genauigkeit bei der Lokalisierung von PVA-Objekten erreichen kann – was auf eine robuste Segmentierungsfähigkeit trotz erhöhter Modellkomplexität hinweist.

7.3.3 Zusammenfassung der Ergebnisse

Die kombinierte Analyse über die Konfusionsmatrix und die IoU-Werte zeigt:

- Mask R-CNN ist im Vergleich zu Faster R-CNN in beiden Bewertungsmethoden überlegen.
- Bei Mask R-CNN konnten im kombinierten Datensatz (PVA und STA) sogar bessere geometrische Genauigkeiten (höhere IoU) für PVA erreicht werden als im Einzelklassendatensatz.
- Die STA-Klassen bereiten in den kombinierten Modellen größere Schwierigkeiten, was sich in niedrigen Recall- und IoU-Werten widerspiegelt.

- Die Verwendung beider Bewertungsmethoden liefert ein ganzheitlicheres Bild: Während die Konfusionsmatrix Fehleinschätzungen auf Objektebene aufdeckt, zeigt die IoU, wie exakt die Lokalisation geometrisch getroffen wurde.

Zusammenfassend ist **Mask R-CNN** die bessere Wahl für die Erkennung von Photovoltaikanlagen (PVA), da es durchgehend eine höhere Klassifikationsleistung sowie eine bessere Lokalisationsgenauigkeit erzielt. Die geringe Performance bei der Erkennung von Solarthermieranlagen (STA) weist unabhängig vom Modell darauf hin, dass weitere Optimierungen wie eine Datenerweiterung oder spezialisierte Modellanpassungen (z. B. Anpassung der Modellarchitektur, Hyperparameter-Tuning) erforderlich sein könnten. Mit dieser nuancierten Herangehensweise ist eine präzisere Beurteilung der Modellergebnisse möglich. Sie stellt zudem die Basis für die Ableitung potenzieller Optimierungsmaßnahmen dar. Durch die Kombination beider Bewertungsmethoden konnten zum einen die inhaltliche Korrektheit der Erkennung (richtige Klassenzuordnung) und zum anderen die räumliche Genauigkeit (exakte Lokalisierung) umfassend beurteilt werden.

8 Diskussion der Ergebnisse

In diesem Kapitel werden die wesentlichen Erkenntnisse aus den Modellbewertungen im Hinblick auf die zentrale Forschungsfrage sowie die begleitenden Arbeitsfragen reflektiert. Ziel ist es, die Ergebnisse in den größeren Kontext der Forschungsarbeit einzuordnen, Stärken und Schwächen der Modelle zu beleuchten und Rückschlüsse auf die Eignung der eingesetzten Methoden zur Erkennung und Klassifizierung von Photovoltaikanlagen zu ziehen. Dieses Kapitel ist entscheidend, da hier eine kritische Auseinandersetzung mit den Ergebnissen erfolgt und nicht nur eine Auswertung.

8.1 Beantwortung der Forschungsfrage

Die zentrale Forschungsfrage dieser Masterarbeit lautete, ob innerhalb der Domäne der Computer Vision die Objekterkennung oder die Bildsegmentierung die geeignetere Methode zur Identifikation und Klassifizierung von Photovoltaikanlagen (PVA) darstellt. Im Zuge der empirischen Analyse wurden zwei repräsentative Modelltypen beider Ansätze evaluiert: Faster R-CNN als objektbasiertes Verfahren und Mask R-CNN als segmentierungsbasiertes Verfahren. Beide Modelle wurden jeweils auf einem Einzelklassendatensatz (nur PVA) sowie einem kombinierten Datensatz (PVA und STA) trainiert, wodurch vier Szenarien miteinander vergleichbar gemacht wurden.

Die Bewertung basierte auf den klassifikationsorientierten Metriken (Precision, Recall und F1-Score) und der geometriebasierten Metrik Intersection over Union (IoU). Die Ergebnisse weisen auf ein konstantes Leistungsplus der segmentierungsbasierten Mask R-CNN-Modelle im Vergleich zu den objekterkennungsbasierten Faster R-CNN-Modellen – vor allem in Bezug auf die geometrische Genauigkeit (IoU) und die Fähigkeit, vollständige Anlagenflächen zu erkennen. Besonders hervorzuheben ist dabei, dass Segmentierungsmodelle nicht nur die Lokalisierung eines Objektes leisten, sondern auch dessen exakte Form erfassen können. Dies ist für eine weiterführende quantitative Auswertung – beispielsweise zur Flächenberechnung oder Bestandserhebung – von zentraler Bedeutung. Zwar konnten in einigen Fällen auch mit Mask R-CNN nicht alle PVA-Flächenbereiche korrekt segmentiert werden. Teilweise wurden speziell große Objektbereiche nur unvollständig erkannt oder es blieben ganze Segmente innerhalb der tatsächlichen Anlagen leer. Diese Einschränkungen verdeutlichen, dass auch segmentierungsbasierte Modelle fehleranfällig sein können – etwa bei stark variierenden Erscheinungsformen, Abschattungen oder bei geringer Trainingsdatenbasis – und die Ergebnisse stets mit Vorsicht interpretiert werden sollten. Dennoch fiel die flächenhafte Abdeckung signifikant besser aus als bei den objektbasierten Modellen, die lediglich rechteckige Bounding Boxes liefern.

Zusammenfassend lässt sich feststellen, dass im untersuchten Kontext der flächenhaften Detektion und Klassifikation von PVA segmentierungsbasierte Ansätze (Bildsegmentierung) wie Mask R-CNN – trotz ihres höheren Rechenaufwands – überlegen sind und die qualifiziertere Methode darstellen, da sie die Objekte nicht nur lokalisieren, sondern auch deren exakte Form auf Pixelebene erfassen können. Diese Einschätzung wird sowohl durch die quantitativen Metriken als auch durch die qualitative Auswertung der Vorhersagen untermauert.

8.2 Beantwortung der Arbeitsfragen

Dieses Kapitel bietet eine systematische Beantwortung der in Kapitel 1 festgelegten Arbeitsfragen, gestützt auf die zuvor vorgestellten Ergebnisse. Das Ziel besteht darin, die wesentlichen Einsichten zur

Wahl geeigneter Modelle, zur genutzten Entwicklungsumgebung und zu bedeutsamen Modellparameter zusammenzutragen und diese im Hinblick auf die Forschungsfrage zu kontextualisieren.

8.2.1 Geeignete Deep Learning-Modelle

Für die beiden Ansätze – Objekterkennung und Bildsegmentierung – wurden etablierte Deep Learning-Modelle herangezogen: Faster R-CNN für die Objekterkennung sowie Mask R-CNN für die semantische bzw. Instanzsegmentierung. Beide Modelle sind klassische Vertreter ihrer Kategorien und basieren auf dem R-CNN-Ansatz mit einem gemeinsamen ResNet-Backbone. Wie in Kapitel 8.1 bereits erläutert, wies Mask R-CNN im Vergleich zu Faster R-CNN eine höhere geometrische Genauigkeit und eine verbesserte Erfassung komplexer Anlagenformen auf. Die objektbasierten Modelle zeichneten sich hingegen durch eine effiziente Lokalisierung und eine gute Klassifikation bei klar abgegrenzten Objekten aus. Sie sind jedoch bei überlappenden oder unregelmäßigen Strukturen benachteiligt. Diese Erkenntnisse belegen, dass die Wahl des Modells stark vom jeweiligen Anwendungsziel abhängt – ob eine grobe Detektion oder eine detaillierte Flächenanalyse angestrebt wird.

- Faster R-CNN ist gut geeignet, um Objekte mit klar abgegrenzten Formen zu erkennen und zu lokalisieren. Es bietet eine präzise Klassifizierung innerhalb von Bounding Boxes, weist eine vergleichsweise hohe Effizienz auf und findet nicht nur im Bereich der Fernerkundung, sondern auch in vielen anderen Anwendungsgebieten (wie Medizin, Verkehrsüberwachung und Landwirtschaft) erfolgreiche Anwendung. Weil sie sich als verlässliches Standardverfahren bei der Objekterkennung auch mit kleinen Datenmengen bewährt hat, eignet sie sich aufgrund ihrer Stabilität und Genauigkeit dafür.
- Mask R-CNN baut auf Faster R-CNN auf und fügt einen zusätzlichen Zweig für die Segmentierung auf Pixelebene hinzu. Diese Architektur eignet sich besonders für flächenhafte, nicht rechteckige Strukturen, wie sie bei PV-Anlagen und Solarthermieanlagen vorkommen können. Diese Arbeit hat gezeigt, dass Mask R-CNN genauere Segmentierungsergebnisse liefert, insbesondere bei überlappenden oder komplex geformten Anlagen.

8.2.2 Entwicklungsumgebung und Umsetzung

Zur praktischen Umsetzung der Modelle wurde die Plattform ArcGIS Pro Notebook zusammen mit der Python-basierten Bibliothek `arcgis.learn` als Hauptbibliothek verwendet. Diese Umgebung erlaubte es, georeferenzierte Daten nahtlos zu integrieren, die Ergebnisse visuell zu überprüfen und die Modelle benutzerfreundlich anzuwenden. `arcgis.learn` bietet vortrainierte Modelle auf Basis von PyTorch an, was besonders bei kleinen Datensätzen vorteilhaft ist. Ergänzend dazu kamen selbstverständlich weitere Python-Bibliotheken wie z. B. `fastai`, `TensorFlow`, `os`, `pathlib`, `numpy`, oder `pandas` zum Einsatz, um Daten vorzubereiten, Analysen durchzuführen und Visualisierungen zu erstellen. Insgesamt hat sich die Verwendung von ArcGIS Pro als geeignete Plattform für diese Arbeit erwiesen, insbesondere durch ihre Stärken in der Kopplung von Fernerkundungsdaten mit Deep-Learning-Modellen sowie ihrer engen Integration in bestehende GIS-Arbeitsumgebungen – dies ist effizient für praxisnahe Anwendungen oder nicht tief informatikaffine Zielgruppen.

8.2.3 Einflussfaktoren auf die Modellleistung

Bei der Bewertung der Modelle traten verschiedene Einflussfaktoren zutage, die sich auf die Erkennbarkeit und Klassifizierbarkeit von PVA auswirkten. Darunter fallen unter anderem:

- **Anzahl und Vielfalt der Trainingsdaten:** Eine große Menge an Trainingsdaten und eine hohe Varianz innerhalb der Klassen sind für die Lernfähigkeit eines Deep Learning-Modells entscheidend. Es ist ausschlaggebend, dass der Trainingsdatensatz verschiedene Erscheinungsformen, Ausrichtungen, Lichtverhältnisse und Objektgrößen umfasst, um Photovoltaikanlagen effektiv zu erkennen. Das Modell kann relevante Merkmale zuverlässig abstrahieren und auf neue Daten generalisieren, wenn eine breite Abdeckung realistischer Szenarien gegeben ist. Ein Mangel an Diversität kann Überanpassungen hervorrufen und die Leistung in realen Anwendungsszenarien erheblich begrenzen.
- **Klassendiversität im Datensatz:** Je nach Modell variierte der Einfluss der Anzahl der Klassen, die erkannt werden sollten. Bei der Erkennung einer einzelnen Klasse (PVA) wies Mask R-CNN einen leicht höheren F1-Score auf. Im kombinierten Datensatz war die Intersection over Union (IoU) für PVA deutlich besser, was auf eine genauere Segmentierung hindeutet – trotz der zusätzlichen Komplexität durch eine zweite Klasse (STA). Obwohl die IoU für STA deutlich niedriger war, erreichten die Modelle insgesamt robuste F1-Scores bei der gleichzeitigen Erkennung beider Klassen. Das lässt darauf schließen, dass die Auswirkungen einer höheren Klassendiversität auf die Erkennungsleistung je nach Modellarchitektur variieren können.
- **Einfluss der Modellarchitektur:** Die Erkennungsgenauigkeit und Detailtreue werden stark von der Architektur des Modells beeinflusst. Im Vergleich zu den objektbasierten Faster R-CNN-Modellen lieferten die segmentierungsbasierten Mask R-CNN-Modelle insgesamt robustere und geometrisch genauere Ergebnisse. Während Faster R-CNN nur rechteckige Bounding Boxes verwendet, ermöglicht Mask R-CNN eine Segmentierung der Objekte auf pixelgenauer Ebene. Dank dieser Fähigkeit wird vor allem bei flächenhaften Strukturen wie Photovoltaikanlagen eine höhere geometrischen Präzision erreicht. Die Vorzüge der Segmentierung zeigen sich auch bei Objekten, die nah beieinander liegen oder teilweise überlappen, da dort die Objektklassifikation häufig an ihre Limitationen gelangt.
- **Räumliche Auflösung der Eingabebilder:** Die konstante Pixelauflösung von 0,2m hat einen erheblichen Einfluss auf die Fähigkeit der Modelle, kleinere oder strukturell weniger auffällige Photovoltaikanlagen zuverlässig zu erkennen. Objekte, die nur wenige Pixel groß erscheinen, können vom Modell schwer als zusammenhängende Struktur interpretiert werden, was zu Unsicherheiten oder Fehldetektionen führt. Die Bildauflösung ist für die Detektionsleistung besonders wichtig, wenn es sich um schmale oder kleinteilige Anlagen handelt.
- **Datenaufbereitung und Augmentation:** Durch die gezielte Vorbereitung der Inputdaten, insbesondere mittels Dataaugmentation, wird die Robustheit und Generalisierbarkeit des Modells erheblich verbessert. In dieser Arbeit wurden unter anderem Transformationen wie Rotation, Helligkeitsanpassung und Spiegelung verwendet. Dank dieser Erweiterung des Trainingsdatensatzes kann das Modell auch bei wechselnden Umgebungsbedingungen zuverlässige Vorhersagen machen. Augmentation ist ein unverzichtbarer Teil, um Overfitting zu verhindern und die Erkennungsleistung zu steigern, insbesondere wenn die Datenmenge begrenzt ist.

- **Berücksichtigung von Falschklassifikationen:** Ein weiterer wesentlicher Punkt bei der Bewertung war die Schaffung einer zusätzlichen Kategorie „none“, die jedoch keine echte Klasse im Kontext des Trainingsdatensatzes ist. Sie wurde vielmehr in die Auswertung einbezogen, um Falschklassifizierungen besser zu identifizieren, bei denen das Modell ein Objekt erkannte, das keiner der tatsächlichen Klassen (PVA oder STA) entsprach. Diese nachträgliche Kategorisierung ermöglicht eine realistischere Einordnung des Modellverhaltens, da auch Fehldetektionen sichtbar werden und in die Klassifikationsmetriken einfließen können.

8.3 Stärken und Schwächen

Dieses Kapitel widmet sich der Ausarbeitung der grundlegenden Vorzüge und Nachteile der verwendeten Modelle sowie der fundamentalen Computer Vision-Ansätze Objekterkennung und Bildsegmentierung. Die Absicht besteht darin, die Methoden hinsichtlich ihrer Leistungsfähigkeit differenziert zu bewerten und ihre spezifischen Vor- und Nachteile im Zusammenhang mit der Aufgabenstellung einzuordnen.

8.3.1 Stärken und Schwächen der Modelle

Wie in Kapitel 8.1 angemerkt, liefern segmentierungsbasierte Modelle wie Mask R-CNN grundsätzlich detailliertere und flächentreuere Ergebnisse als objektbasierte Modelle wie Faster R-CNN. Dies ist besonders vorteilhaft für Anwendungen, bei denen die genaue räumliche Ausdehnung und Form der Objekte von Bedeutung ist – wie beispielsweise bei der Flächenabschätzung von Photovoltaik- oder Solarthermieranlagen. Objektbasierte Modelle wie Faster R-CNN brillieren bei der robusten Lokalisierung und effizienten Klassifizierung mittels Bounding Boxes, haben jedoch Nachteile bei unregelmäßigen oder Formen und kleinen Objekten. Segmentierungsmodelle bieten zwar eine höhere geometrische Genauigkeit, benötigen jedoch mehr Rechenleistung und sind empfindlicher gegenüber Datenungleichgewichten oder qualitativ minderwertigen Trainingsdaten.

8.3.2 Stärken und Schwächen der CV-Ansätze Objekterkennung und Bildsegmentierung

Es zeigen sich grundlegende Unterschiede zwischen den beiden methodischen Ansätzen. Die Objekterkennung ermöglicht eine rasche und ressourcenschonende Analyse. Sie ist jedoch für unregelmäßige oder komplexe Objektformen nur bedingt geeignet. Deshalb ist sie vor allem für Szenarien nützlich, in denen eine grobe Lokalisierung ausreicht.

Dagegen hat die Instanzsegmentierung den Vorzug, dass sie Objektgrenzen und -flächen exakt abbildet. Sie eignet sich besonders für Aufgaben, bei denen die genaue räumliche Ausdehnung entscheidend ist und sie zeigt bei kleinen oder dicht beieinander liegenden Objekten eine bessere Leistung. Sie haben den Nachteil, dass sie mit höherem Rechenaufwand verbunden und bei geringerer Datenqualität potenziell stärker fehleranfällig sind.

8.3.3 Methodenkritik und Grenzen der Arbeit

Obwohl verschiedene Modelle zur Objekterkennung und Segmentierung mit Bedacht ausgewählt und angewendet wurden, bestehen methodische Einschränkungen, die die Aussagekraft der Ergebnisse relativieren. Die beschränkte Menge an Trainings- und Testdaten stellt eine wesentliche Einschränkung dar. Trotz der Datenaugmentation konnten nicht alle Erscheinungsformen von PVA sowie alle potenziellen Störfaktoren in ausreichender Vielfalt in den Trainingsdaten repräsentiert werden. So

blieb das Modell anfällig für Fehlklassifikationen in Bezug auf bestimmte Ausprägungen oder Störfaktoren. Außerdem gab es ein Ungleichgewicht in der pixelbasierten Klassenverteilung der kombinierten Datensätze. Obgleich die Bildchipanzahl für beide Klassen identisch war, beanspruchte die Klasse PVA flächenmäßig deutlich mehr Raum als STA. Dieses Ungleichgewicht auf Pixelebene könnte eine Verzerrung der Modellbewertung zur Folge gehabt haben.

Ein weiterer methodischer Kritikpunkt bezieht sich auf die Aggregation der IoU-Werte. Aufgrund technischer Einschränkungen von ArcGIS Pro war es nicht möglich, eine objektweise IoU-Berechnung für Segmentierungen vorzunehmen. Daher wurde lediglich ein durchschnittlicher IoU-Wert pro Modell berechnet. Dies verringert die Aussagekraft bei der Bewertung einzelner Vorhersagen – ob bestimmte Objekte sehr gut erkannt wurden, während andere völlig fehlerklassifiziert wurden – und macht die Feinanalyse von Modellfehlern schwieriger. Eine feingranulare Analyse wäre besonders für die differenzierte Bewertung der Modellleistung vorteilhaft gewesen.

Ein weiterer Aspekt betrifft die manuelle Kontrolle der Vorhersagen. Die visuelle Darstellung erfolgte anhand der Bounding Boxes bzw. Segmentierungen in der Attributtabelle und im Kartenfenster von ArcGIS Pro. Diese Methode beansprucht nicht nur viel Zeit, sie ist auch von Subjektivität geprägt, da sie von der individuellen Beurteilung der Person abhängt, die die Bearbeitung vornimmt. Eine auf Objektebene automatisierte Validierungsmethode wäre hier objektiver, konnte jedoch im gegebenen Rahmen nicht umgesetzt werden.

Zudem wurde bewusst auf eine detaillierte Fehlerklassifizierung verzichtet. Falschklassifikationen, Vorhersagen der Klasse „none“ oder fälschlich zugewiesene Objektklassen, wurden summarisch erfasst, ohne eine weitergehende qualitative Analyse der zugrunde liegenden Ursachen (z. B. Schattenwurf, Dachstruktur, saisonale Unterschiede) vorzunehmen.

Ein weiterer limitierender Faktor liegt in der Wahl der verwendeten Bibliothek `arcgis.learn`. Zwar bietet diese durch die einfache Implementierung und die enge Integration in ArcGIS Pro eine geringe Einstiegshürde, jedoch zeigte sich im Laufe der Arbeit, dass die Bibliothek im Vergleich zu spezialisierten Deep Learning-Frameworks wie `PyTorch` oder `TensorFlow` gewisse Einschränkungen in Bezug auf Modellanpassung, Auswertungsmöglichkeiten und Flexibilität aufweist. Dies könnte sich auf die erreichbare Modellperformance ausgewirkt haben und sollte in zukünftigen Arbeiten bei der Auswahl des Toolstacks berücksichtigt werden.

Die Ergebnisse sind nicht ohne Weiteres auf andere Regionen, Bilddaten oder Anlagenstrukturen übertragbar. Daher sollten künftige Arbeiten eine größere Datenvielfalt und differenzierte Fehleranalysen berücksichtigen.

8.4 Positionierung im Forschungsfeld

Die vorliegende Arbeit unterscheidet sich sowohl hinsichtlich der Datenbasis als auch in Bezug auf den methodischen Aufbau von den in Kapitel 5.8 vorgestellten Studien. Der wesentliche Unterschied besteht in der Verknüpfung eines kombinierten Datensatzes (PVA und STA) mit zwei bewährten Deep Learning-Architekturen (Faster R-CNN und Mask R-CNN), die jeweils auf zwei verschiedene Klassenszenarien angewendet wurden. Hier wurde neben dem Fokus auf Photovoltaikanlagen der

zusätzliche Aspekt der Solarthermieranlagen berücksichtigt, um das Potenzial der Modelle zur differenzierten Erkennung verwandter Objekte zu bewerten.

Bei der Wahl des Modells wurde Faster R-CNN, ein reines Objekterkennungsmodell, und Mask R-CNN, ein segmentierungsbasiertes Modell, evaluiert. Andere Untersuchungen, wie zum Beispiel Ge et al. (2022), kombinierten zwei spezialisierte Netzwerke (EfficientNet-B5 und U²-Net) für die Objekterkennung und Segmentierung. Dies bietet potenziell eine höhere Modellgüte, ist jedoch auch komplexer in der Umsetzung. Perry und Campos (2023) verwendeten ebenfalls Faster R-CNN mit ResNet-50 als Backbone, differenzierten aber zwischen vier sehr spezifischen Installationsarten. Dies wurde in dieser Arbeit zugunsten einer klassenspezifischen Trennung (PVA vs. STA) vereinfacht.

Die eigene Arbeit weist im Vergleich dazu eine etwas höhere Fehlerrate auf. Dies könnte teilweise an der Datenbasis (z. B. begrenzte Variabilität oder Qualität der Orthofotos) sowie an der Modellarchitektur liegen. Vor allem bei komplexen Hintergründen offenbarte Mask R-CNN eine gewisse Unsicherheit, obwohl das Modell in anderen Untersuchungen (z. B. Díaz et al., 2020) mit Thermalbildern signifikante Verbesserungen im Vergleich zu klassischen Verfahren wie SVM erzielte. Die Verwendung unterschiedlicher Bilddaten (Sensorquellen) könnte unter anderem dazu beitragen, diese Diskrepanz zu erklären – UAV-Thermalbilder bieten eine deutlichere Trennung von Hintergrund und Objekt im Vergleich zu orthogonalen RGB-Aufnahmen. Ein weiterer potenzieller Einflussfaktor ist die Bildauflösung: Petrik und Hamme (2020) arbeiteten mit hochauflösenden Orthofotos (0,1–0,3m), während in dieser Masterarbeit Bilder mit einer ähnlich hohen Auflösung von 0,2m verwendet wurden. Die Resultate weisen darauf hin, dass kleinere Objekte (vor allem STA) in bestimmten Fällen nicht eindeutig identifiziert werden konnten. Dies könnte daran liegen, dass sie im Vergleich zu großflächigen PV-Anlagen eine geringe Größe aufweisen.

8.4.1 Methodische Zielsetzung und Rahmenbedingungen der vorliegenden Masterarbeit

Ein Ziel dieser Arbeit war es nicht nur, die Leistungsfähigkeit verschiedener Modelle zu vergleichen, sondern auch aufzuzeigen, dass bildsegmentierende Ansätze wie Mask R-CNN grundsätzlich ein höheres Potenzial für die Erkennung und Klassifizierung von Photovoltaikanlagen aufweisen – insbesondere im Hinblick auf die Forschungsfrage, ob die Segmentierung der Objekterkennung überlegen ist. Mask R-CNN bietet einen methodischen Vorteil, da die exakte räumliche Ausdehnung der Anlagen durch die präzise Trennung der Objektgrenzen erkannt werden kann. Dies führt insbesondere bei komplizierten Strukturen oder sich überlappenden Objekten tendenziell zu besseren Resultaten.

Die Rahmenbedingungen dieser Arbeit wurden gleichzeitig durch eine bewusste Entscheidung zur Beschränkung auf eigene Daten geprägt: Es standen keine externen Trainingsdatensätze zur Verfügung, die direkt für das spezifische Anwendungsszenario verwendet werden konnten. Die Trainingsdaten wurden daher unabhängig und vollständig manuell erstellt – ein aufwändiger Prozess, der die Anzahl und Varianz der annotierten Objekte einschränkte. Die Trainingsmöglichkeiten und die generalisierbare Aussagekraft der Ergebnisse werden naturgemäß durch die begrenzte Verfügbarkeit von Daten beeinflusst.

Zusätzlich verfolgte diese Arbeit das Ziel, die Hypothese zu prüfen, ob eine höhere Genauigkeit und Differenzierung erreicht werden kann, wenn mehrere Klassen (PVA und STA) kombiniert sowie ein

segmentierungsbasiertes Modell wie Mask R-CNN verwendet werden. Nur ein Teil dieser Annahme fand Bestätigung. Obwohl Mask R-CNN bei der Lokalisierung der Klasse PVA im kombinierten Datensatz eine deutlich höhere IoU als im Einzelklassenszenario erreichte, war die Trennschärfe zur zusätzlichen Klasse STA unzureichend – dies äußerte sich in einer sehr niedrigen IoU für STA. Wie die Arbeiten von Díaz et al. (2020) oder Vasku (2019) demonstrieren, könnte eine mögliche Optimierung durch die Verwendung weiterer Modalitäten wie Infrarot- oder Multispektraldaten erzielt werden – wobei Vasku aber auf traditionelle ML-Methoden zurückgriff.

Gerade unter diesen Bedingungen zeigt sich jedoch umso deutlicher, dass segmentierende Modelle wie Mask R-CNN in der Lage sind, auch mit kleineren Datensätzen gute Ergebnisse zu erzielen – insbesondere im Vergleich zu reinen Objektklassifikationsmodellen, die in dieser Arbeit ebenfalls getestet wurden. Dieses Ergebnis unterstützt die Hypothese, dass Segmentierungsansätze methodisch überlegen sein können, wenn es um die exakte und differenzierte Erkennung von PV-Anlagen geht.

8.4.1.1 Vergleich der Klassifikationsergebnisse mit verwandten Studien

Obwohl davon ausgegangen wurde, dass segmentierungsbasierte Modelle bessere Ergebnisse liefern, zeigen die Ergebnisse im Vergleich zu bestehenden Studien teilweise gegenteilige Tendenzen. Diese Diskrepanz zwischen Hypothese und empirischer Evidenz ist Teil jedes wissenschaftlichen Prozesses und erlaubt wichtige Rückschlüsse auf Optimierungspotenziale. Um die Einordnung der Ergebnisse weiter zu verfeinern, folgt ein Vergleich mit den bereits zuvor verglichenen Studien, die ähnliche Zielsetzungen verfolgten. Soweit vorhanden, werden die Metriken Precision, Recall und F1-Score gegenübergestellt. Nicht alle Arbeiten berichten über die gleichen Kennwerte, so dass der Vergleich selektiv zu interpretieren ist. In der folgenden Tabelle werden die besten Studienergebnisse (bezogen auf die Klassifikationsmetriken) miteinander verglichen:

Studie	Precision	Recall	F1-Score	Anmerkungen
Eigene Arbeit	0.87	0.58	0.70	Das Modell mit der besten Leistung in Bezug auf die Klassifikationsmetriken.
Díaz et al. (2020)	0.997 0.996	0.970 0.981	0.983 0.989	Erste Zeile: Segmentierung inkl. SVM und Postprocessing; zweite Zeile: DL und Postprocessing
Ge et al. (2022)	0.937	0.964	0.950	Haben eine eigene Methode entwickelt.
Perry & Camos (2023)	---	---	---	Es wurde nur mAP berechnet: 77.79%
Petrik & Hamme (2020)	0.722 0.210	0.689 0.457	--- ---	In ihrer Studie wird von „Genauigkeit“ und „Trefferquote“ gesprochen: Genauigkeit $\approx$ Precision, Trefferquote $\approx$ Recall; Erste Zeile: Ergebnisse mit vorhandenem Datensatz; zweite Zeile: Ergebnisse mit eigens erstelltem Datensatz
Vasku (2019)	0.716	0.549	---	Kein F1-Score; Producer's Accuracy $\approx$ Recall, Consumer's Accuracy $\approx$ Precision

Tab. 22: Vergleich der Studienergebnisse

Im Vergleich zeigt sich, dass einige Studien, insbesondere Ge et al. (2022) und Díaz et al. (2020), deutlich höhere Kennzahlen in allen drei Metriken erreichen. Dies ist vor allem auf spezialisiertere Modelle,

umfangreichere Trainingsdaten oder zusätzliche Pre- und Postprocessing-Schritte zurückzuführen. Die eigene Masterarbeit liegt insbesondere beim Recall im unteren Bereich. Dies kann auf eine im Vergleich kleinere Trainingsdatenbasis, die Heterogenität der Bilddaten sowie die Verwendung eines generalisierten Segmentierungsmodells ohne spezifische Optimierung zurückgeführt werden.

Dennoch zeigt sich, dass die Klassifikationsgenauigkeit (Precision) der eigenen Arbeit mit 0.87 in einem respektablen Bereich liegt und damit vergleichbar oder sogar besser als Arbeiten wie Petrik und Hamme (2020) abschneidet. Allerdings betrachtet die Precision nur den Anteil korrekt klassifizierter positiver Vorhersagen und berücksichtigt nicht, wie viele Objekte tatsächlich erkannt wurden. Daher erlaubt der F1-Score als harmonisches Mittel aus Precision und Recall eine aussagekräftigere Gesamteinschätzung der Modellleistung. In jedem Fall ist der F1-Score für diese Arbeit verbesserungswürdig. Der Vergleich mit Vasku (2019) zeigt, dass auch klassische Verfahren wie Random Forest im gleichen Problemfeld Schwächen aufweisen - etwa bei der Produzentengenauigkeit (Recall). Er verwendet Producer's und Consumer's Accuracy, die funktional mit Recall und Precision vergleichbar sind. Ein direkter Vergleich des F1-Scores ist aber nicht möglich.

Die kritische Reflexion der Ergebnisse zeigt, dass die erzielten Ergebnisse nicht immer den ursprünglichen Erwartungen entsprechen. Mögliche Ursachen hierfür liegen in der begrenzten Trainingsdatenbasis, der Modellarchitektur, den gewählten Parametereinstellungen oder auch externen Einflussfaktoren wie Bildauflösung oder Lichtverhältnissen. Diese Abweichungen liefern wertvolle Hinweise für zukünftige Optimierungsschritte und zeigen gleichzeitig die Potenziale und Grenzen des derzeitigen methodischen Ansatzes auf. Die gewonnenen Erkenntnisse bilden die Grundlage für gezielte Empfehlungen, um in zukünftigen Arbeiten robustere, anwendungsnähere und leistungsfähigere Modelle zu entwickeln oder anzuwenden.

8.5 Forschungsempfehlungen und Ausblick

Aufbauend auf den Ergebnissen und der in Kapitel 8.4 vorgenommenen methodischen Einordnung dieser Arbeit sollen im Folgenden konkrete Empfehlungen – verschiedene Potenziale zur Verbesserung der Modellgenauigkeit und -robustheit – für künftige Forschungsvorhaben im Bereich der automatisierten Detektion und Klassifizierung von Photovoltaikanlagen ausgesprochen werden. Diese Empfehlungen betreffen sowohl die Verbesserung der Modellarchitekturen und Trainingsdaten als auch strategische Überlegungen zur langfristigen Integration solcher Verfahren in Forschung und Praxis.

- **Erweiterung des Klassenschemas:** Um Falschklassifikationen – etwa von Glaskörpern, Dachfenstern oder anderen reflektierenden Flächen – künftig zu minimieren, sollte das Klassenschema um weitere Objektkategorien ergänzt werden. Neben Photovoltaikanlagen (PVA) und Solarthermieranlagen (STA) könnten künftig auch Schwimmbäder, (Dach-)Fenster, Gewächshäuser oder Paletten berücksichtigt werden. Diese Erweiterung würde es ermöglichen, Modelle gezielter auf trennscharfe Merkmale zu trainieren, anstatt lediglich binäre Entscheidungen zu treffen. Daraus ergibt sich die Notwendigkeit, künftig auch empirisch zu untersuchen, inwiefern zusätzliche Klassen die Trennschärfe verbessern können – ein Aspekt, der in bisherigen Studien wie bei Petrik und Hamme (2023) bisher kaum systematisch analysiert wurde.

- **Gezielte Datenanreicherung und -diversifizierung:** Eine größere und ausgewogenere Datenbasis, sprich eine gezielte Ergänzung um häufig falsch klassifizierte Objekte und Szenen (z. B. Fahrzeuge, Schatten, transparente Flächen) durch insbesondere Datenaugmentation sowie eine Variation hinsichtlich Jahreszeit und Lichtverhältnissen könnte die Modellrobustheit erhöhen.
- **Optimierung der Annotationsstrategie:** Ein weiteres Optimierungspotenzial liegt in der Gestaltung der Annotationen. Die in dieser Arbeit gewählte Herangehensweise, größere Anlagen in kleinere Einheiten zu segmentieren, erhöht die Konsistenz innerhalb des Datensatzes, birgt jedoch die Gefahr einer Übersegmentierung. Eine alternative Strategie könnte darin bestehen, große PVA-Flächen als zusammenhängende Objekte zu annotieren. Künftig wäre zu untersuchen, welche Annotationsweise – abhängig vom Anwendungskontext (z. B. Objekterkennung vs. Flächenbewertung) – zielführender ist.
- **Hybride Modellarchitekturen und moderne Ansätze:** Die Kombination von Objekterkennung und Segmentierung in einem hybriden Modellansatz stellt ein vielversprechendes Optimierungspotenzial dar. Ebenso könnten modernere Architekturen wie Transformer-basierte Modelle wie EfficientDet oder Segment Anything (SAM) geprüft werden, die in der Computer Vision zunehmend an Bedeutung gewinnen. Auch könnte das Finetuning von Backbone-Netzen (z. B. durch tiefere ResNet-Varianten) die Trennschärfe zwischen ähnlichen Objekten erhöhen.
- **Integration von Zusatzdaten und Post-Processing:** Obwohl GIS-basierte Filtermechanismen – etwa auf Basis von Gebäudeumrissen, Höhenmodellen (DSM/DTM), Dachneigungen oder Landnutzungsklassen – in der Literatur als hilfreich beschrieben werden, zeigte sich in der praktischen Umsetzung, dass diese Ansätze aufgrund fehlender aktueller und flächendeckender Daten nur begrenzt umsetzbar sind. Dennoch sollte in zukünftigen Arbeiten geprüft werden, ob sich durch qualitativ hochwertige Zusatzdaten (z. B. aus Laserscans oder amtlichen Bauwerksdaten) die Klassifikation verbessern lässt.
- **Beachtung der tatsächlichen Flächenverteilung pro Klasse:** Künftige Arbeiten sollten nicht nur auf eine ausgewogene Anzahl der Trainingsobjekte achten, sondern auch die tatsächliche Pixelbedeckung pro Klasse berücksichtigen. Andernfalls kann ein implizites Bias entstehen – etwa zugunsten flächenmäßig dominanter Klassen wie PVA gegenüber kleineren STA-Anlagen. Differenzierte Datenaugmentation, gezieltes Upsampling unterrepräsentierter Klassen sowie der Einsatz auf kleinere Objekte spezialisierter Architekturen könnten hier Abhilfe schaffen.

Langfristig eröffnet die Kombination von Fernerkundung, künstlicher Intelligenz und Geoinformationssystemen neue Perspektiven für die automatisierte Erfassung und Analyse von Infrastrukturen wie PV-Anlagen – sowohl im Kontext der Energiewende als auch in der raumbezogenen Umweltforschung. Besonders im Hinblick auf flächendeckende Potenzialanalysen, Monitoring-Aufgaben oder Nachhaltigkeitsberichte kann der Einsatz von Deep Learning einen wertvollen Beitrag leisten. Voraussetzung dafür ist jedoch eine hohe Genauigkeit und Generalisierbarkeit der Modelle sowie die Verfügbarkeit aktueller, qualitätsgesicherter Daten.

Zukünftige Studien könnten daher gezielt auf die Erkennung bislang unterrepräsentierter Objekttypen (wie STA) fokussieren oder untersuchen, wie sich regionale Unterschiede (z. B. Solarzellentypen,

Bebauungsstruktur, Dachformen) auf die Erkennungsleistung auswirken. Ebenso sollte die praktische Integration solcher Verfahren in kommunale oder institutionelle Arbeitsprozesse erprobt und evaluiert werden – etwa zur Unterstützung der Energiewende auf Gemeindeebene.

9 Fazit

Die theoretischen Kapitel dieser Arbeit legten eine wichtige Grundlage für das Verständnis der praktischen Untersuchungen. Zunächst wurde die Rolle der Photovoltaik im Kontext der österreichischen Energiewende beleuchtet. Die Analyse aktueller Entwicklungen zeigte, dass Solarenergie – insbesondere Photovoltaik – ein zentrales Element zur Dekarbonisierung der Energieversorgung darstellt. Darauf aufbauend wurden die zentralen Konzepte der künstlichen Intelligenz sowie der Entwicklungsgeschichte und Funktionsweise künstlicher neuronaler Netze dargestellt. Besonders relevant waren dabei die Lernmethoden im überwachten Machine Learning und die Wirkweise von Deep Learning-Ansätzen wie Convolutional Neural Networks (CNN), welche für bildbasierte Aufgabenstellungen wie Objekterkennung und Segmentierung konzipiert sind. Die theoretische Auseinandersetzung mit Bildklassifizierung, Objekterkennung sowie Instanz- und semantischer Segmentierung bildete die methodische Basis zur Auswahl geeigneter Modelle. Herausforderungen wie Overfitting, schwache Gradienten und eine angemessene Modellarchitektur wurden ebenso thematisiert wie Ansätze aus verwandten wissenschaftlichen Arbeiten. Insgesamt diente dieser theoretische Rahmen als essenzielles Fundament, um die Leistungsfähigkeit verschiedener Modellansätze im Kontext der automatisierten Erkennung von Photovoltaikanlagen kritisch zu analysieren und einzuordnen. Die Analyse zeigte, dass segmentierungsbasierte Ansätze wie Mask R-CNN grundsätzlich zu genaueren und differenzierteren Vorhersagen in Bezug auf die Objektform führen. Insbesondere bei der Trennung zwischen Photovoltaik- und Solarthermieranlagen konnte Mask R-CNN bessere Ergebnisse erzielen, was sich unter anderem in einer geringeren Anzahl an Falschklassifikationen zeigte. Objekterkennungsmodelle wie Faster R-CNN hingegen bieten Vorteile in der Recheneffizienz sowie in der Robustheit bei der Lokalisierung – insbesondere bei kleineren Objekten. Gleichzeitig traten bei Faster R-CNN jedoch häufiger Verwechslungen zwischen den Klassen auf, was die Trennschärfe der Vorhersagen beeinträchtigte.

Die Ergebnisse bestätigten zudem, dass der kombinierte Datensatz (PVA und STA) tendenziell zu einer höheren Generalisierbarkeit führte, gleichzeitig jedoch erhöhte Anforderungen an die Modellarchitektur stellte – insbesondere an deren Fähigkeit, visuell ähnliche Klassen zuverlässig zu unterscheiden. Während der Trainingsprozesse zeigte sich deutlich, wie komplex die Entwicklung robuster Modelle ist – sei es in der Auswahl geeigneter Backbone-Netzwerke (Architektur), in der Gestaltung der Trainingsdaten oder in der Feinjustierung einzelner Parameter. Es war ein iterativer Prozess - es wurden wiederholt Anpassungen an den Hyperparametern und Daten vorgenommen, um die Modellleistung zu optimieren. Dazu zählten unter anderem eine Variation der Epochenzahl, die Anwendung zusätzlicher Datenaugmentationen sowie die Anpassung der Lernrate. Die Modelle wurden mehrfach trainiert, wobei stets der Kompromiss (auch Trade-off genannt) zwischen Trainingszeit und Ergebnisqualität abgewogen wurde. Für die abschließende Bewertung und den Leistungsvergleich wurde das jeweils am besten performende Modell (Faster R-CNN und Mask R-CNN) sowie der Datensatz (Datensatz 1 und kombinierter Datensatz) herangezogen, je nach Modellgüte. Diese gezielte Auswahl ermöglichte eine valide Gegenüberstellung der Modellklassen und Datensätze und spiegelte den realitätsnahen Umgang mit maschinellem Lernen wider – das bedeutet, keine lineare Abfolge von Schritten, sondern ein Zusammenspiel aus Hypothesen, Tests und Entscheidungen.

Von besonderer Relevanz ist die Beobachtung, dass nicht nur die Wahl des Modells, sondern auch die Qualität und Vielfalt der Trainingsdaten maßgeblich die Erkennungsleistung beeinflussen. Falschklassifikationen traten insbesondere dort auf, wo visuelle Ähnlichkeiten zu anderen Objekten bestehen – ein Umstand, der durch Erweiterungen des Klassenschemas oder den Einsatz von Zusatzdaten potenziell reduziert werden kann.

Die Arbeit verdeutlicht, dass es keinen universell besten Ansatz gibt – vielmehr hängt die Eignung eines Modells stark vom konkreten Anwendungsfall ab. Für Aufgaben, die eine hohe Genauigkeit der Objektkontur erfordern, sind Segmentierungsmodelle zu bevorzugen, während bei der großflächigen Kartierung einfacher Strukturen auch Objekterkennungsverfahren ausreichen können. Im Kontext der Energiewende macht die Arbeit deutlich, dass die Etablierung solcher Verfahren in der Praxis noch vieler weiterer Forschungsschritte bedarf – sei es in Bezug auf (Geo-)Datenverfügbarkeit und -qualität, der Interoperabilität technischer Systeme, der rechtlich-technischen Standards sowie der Benutzerfreundlichkeit und Akzeptanz in der Praxis.

Insgesamt leistet die vorliegende Arbeit einen Beitrag zur methodischen Bewertung von Deep Learning-Verfahren, und legt eine fundierte Basis für weiterführende Untersuchungen mit der Frage, wie moderne Technologien zur Lösung drängender gesellschaftlicher Herausforderungen beitragen können. Die automatisierte Erkennung von Solaranlagen sowie weiterer erneuerbarer Infrastrukturen bietet ein zukunftsweisendes Werkzeug, um den Ausbau erneuerbarer Energien gezielter zu analysieren, zu überwachen und zu beschleunigen – insbesondere im Rahmen kommunaler Geodateninfrastrukturen. Gerade in Österreich – einem Land mit wachsendem Bewusstsein für nachhaltige Energieversorgung – kommt der Nutzung von Solarenergie ein bedeutendes Potenzial zu. Während Photovoltaikanlagen zur direkten Stromerzeugung beitragen, spielt auch die Solarthermie in vielen Regionen eine wichtige Rolle, etwa zur Warmwasserbereitung oder Heizungsunterstützung. Daher ist es umso wichtiger, beide Technologien nicht nur politisch und planerisch zu fördern, sondern auch datenbasiert sichtbar zu machen. An diesem Punkt setzen automatisierte Erkennungsmethoden an, die künftig eine fundierte Basis für Monitoring, Förderentscheidungen und Potenzialanalysen schaffen können.

Literaturverzeichnis

- Aggarwal, C. C. (2018). Neural networks and deep learning. *Springer*, 10(978), 3.
- Albelwi, S., & Mahmood, A. (2017). A framework for designing the architectures of deep convolutional neural networks. *Entropy*, 19(6), 242.
- Alpaydin, E. (2021). *Machine learning*. Mit Press.
- Ardeshir, S., Zamir, A. R., Torroella, A., & Shah, M. (2014). GIS-assisted object detection and geospatial localization. Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part VI 13,
- Bandyopadhyay, H. (2023). *The Definitive Guide to Instance Segmentation*. V7Labs. Abgerufen am 13.12.2023 von <https://www.v7labs.com/blog/instance-segmentation-guide>
- Bar, A., Wang, X., Kantorov, V., Reed, C. J., Herzig, R., Chechik, G., Rohrbach, A., Darrell, T., & Globerson, A. (2022). DETReg: Unsupervised Pretraining with Region Priors for Object Detection. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition,
- basemap.at. (2024). <https://basemap.at/>
- BEV Datenkatalog. (2023). *Digitales Orthophoto Farbe und Infrarot (DOP RGBI) Operat 2022150*. Bundesamt für Eich- und Vermessungswesen (BEV). Abgerufen am 23.04.2024 von <https://data.bev.gv.at/geonetwork/srv/ger/catalog.search#/metadata/e2300cfb-7ca1-4f92-99c5-75272f025821>
- Bhardwaj, A., Di, W., & Wei, J. (2018). *Deep Learning Essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling*. Packt Publishing Ltd.
- Bhattacharyya, S., Snasel, V., Hassanien, A. E., Saha, S., & Tripathy, B. (2020). *Deep Learning: Research and Applications* (Vol. 7). Walter de Gruyter GmbH & Co KG.
- Biermayr, P., Aigenbauer, S., Dißbauer, C., Eberl, M., Enigl, M., Fechner, H., Fink, C., Fuhrmann, M., Hengel, F., Jaksch-Fliegenschnee, M. L., K, Matschegg, D., Moidl, S., Prem, E., Riegler, T., Savic, S., Schmidl, C., Strasser, C., Wonisch, P., & Wopienka, E. (2023). *Innovative Energietechnologien in Österreich: Marktentwicklung 2022 Nachhaltig Wirtschaften*. Im Rahmen von open4innovation. Abgerufen am 02.02.2023 von https://nachhaltigwirtschaften.at/resources/iea_pdf/schriftenreihe-2023-36a-marktstatistik-2022.pdf
- BMK. (2023). *Energie in Österreich. Zahlen, Daten, Fakten*. Bundesministerium für Klimaschutz, Umwelt, Energie, Mobilität, Innovation und Technologie. Abgerufen von https://www.bmk.gv.at/dam/jcr:da4e9dfd-f51c-44b8-894c-9b049a8336cb/BMK_Energie_in_OE2023_barrierefrei.pdf
- Brownlee, J. (2019). *Overfitting and Underfitting With Machine Learning Algorithms*. Machine Learning Mastery. Abgerufen am 30.03.2024 von <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>
- Bundesamt für Eich- und Vermessungswesen (BEV). (2023). *Digitales Orthophoto Farbe und Infrarot (DOP RGBI) Operat 2022150* [Rasterdatensatz]. <https://data.bev.gv.at/geonetwork/srv/ger/catalog.search#/metadata/e2300cfb-7ca1-4f92-99c5-75272f025821>
- Bundesamt für Eich- und Vermessungswesen (BEV). (2024). *Orthophoto Farbe*. Bundesamt für Eich- und Vermessungswesen (BEV). Abgerufen am 26.04.2024 von <https://www.bev.gv.at/Services/Produkte/Luftbildprodukte/Orthophoto-Farbe.html>
- Bundesministerium für Klimaschutz, U., Energie, Mobilität, Innovation und Technologie. (2023). *Innovative Energietechnologien in Österreich Marktentwicklung 2022*.
- Burkov, A. (2019). *Machine Learning kompakt: Alles, was Sie wissen müssen*. MITP-Verlags GmbH & Co. KG.
- Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., & Xu, J. (2019). MMDetection: Open MMLab Detection Toolbox and Benchmark. *arXiv preprint arXiv:1906.07155*.

- Copernicus. (2024). *OBSERVER: Revealing hidden land patterns with AI and Copernicus*. Copernicus. Abgerufen am 20.01.2025 von <https://www.copernicus.eu/en/news/news/observer-revealing-hidden-land-patterns-ai-and-copernicus>
- Corcoll, O. (2021). Semantic Image Cropping. *arXiv preprint arXiv:2107.07153*.
- Dadhich, A. (2018). *Practical Computer Vision: Extract Insightful Information from Images Using TensorFlow, Keras, and OpenCV*. Packt Publishing Ltd.
- DataScientest. (2023). *Convolutional Neural Network (CNN): Alles, was Du wissen solltest*. Abgerufen am 01.02.2024 von <https://datascientest.com/de/convolutional-neural-network-2>
- DeepLobe. (2021). *Understanding Images from Pixel Level with Semantic Segmentation*. DeepLobe. Abgerufen am 01.10.2024 von <https://deeplobe.ai/understanding-images-from-pixel-level-with-semantic-segmentation/>
- Dey, S. (2020). *Python image processing cookbook: over 60 recipes to help you perform complex image processing and computer vision tasks with ease*. Packt Publishing Ltd.
- DWDS. (2021). *Prosumer*. Abgerufen am 08.12.2023 von <https://www.dwds.de/wb/Prosumer>
- Esri. (2023a). *Detect Objects Using Deep Learning (Image Analyst)*. Esri. Abgerufen am 17.10.2024 von <https://pro.arcgis.com/en/pro-app/3.2/tool-reference/image-analyst/detect-objects-using-deep-learning.htm>
- Esri. (2023b). *Export Training Data for Deep Learning (Image Analyst)*. Esri. Abgerufen am 14.10.2024 von <https://pro.arcgis.com/en/pro-app/3.2/tool-reference/image-analyst/export-training-data-for-deep-learning.htm>
- Esri. (2023c). *Intersect (Analysis)*. Esri. Abgerufen am 25.02.2025 von <https://pro.arcgis.com/en/pro-app/3.2/tool-reference/analysis/intersect.htm>
- Esri. (2023d). *Label objects for deep learning*. Esri. Abgerufen am 23.08.2024 von <https://pro.arcgis.com/en/pro-app/3.2/help/analysis/image-analyst/label-objects-for-deep-learning.htm>
- Esri. (2023e). *Minimum Bounding Geometry (Data Management)*. Esri. Abgerufen am 26.02.2025 von <https://pro.arcgis.com/en/pro-app/3.2/tool-reference/data-management/minimum-bounding-geometry.htm>
- Esri. (2023f). *Pyramiden berechnen (Batch-Modus) (Data Management)*. Esri. Abgerufen am 05.07.2024 von <https://pro.arcgis.com/de/pro-app/3.2/tool-reference/data-management/batch-build-pyramids.htm>
- Esri. (2023g). *Union (Analysis)*. Esri. Abgerufen am 25.02.2025 von <https://pro.arcgis.com/en/pro-app/3.2/tool-reference/analysis/union.htm>
- Esri. (2023h). *What is ArcPy?* Esri. Abgerufen am 20.10.2024 von <https://pro.arcgis.com/de/pro-app/3.2/arcpy/get-started/what-is-arcpy.htm>
- Esri Developer. (2023a). *ArcGIS API for Python v2.3*. Abgerufen am 02.05.2025 von <https://developers.arcgis.com/python-2-3/api-reference/arcgis.learn.toc.html>
- Esri Developer. (2023b). *ArcGIS API for Python v2.3 | Object Detection Models*. Abgerufen am 23.04.2025 von <https://developers.arcgis.com/python-2-3/api-reference/arcgis.learn.toc.html#object-detection-models>
- Esri Developer. (2024). *ArcGIS API for Python*. Esri. Abgerufen am 07.06.2024 von <https://developers.arcgis.com/python/>
- Faninger, G. (2007). Jährlich in Österreich installierte PV-Leistung der Jahre 2000 bis 2022. In: Erneuerbare Energie in Österreich – Marktentwicklung 2006, Bundesministerium für Verkehr, Innovation und Technologie, Berichte aus Energie- und Umweltforschung, 11/2007.
- fastai. (2023). *vision.transform*. fastai. Abgerufen am 01.12.2024 von <https://fastai1.fast.ai/vision.transform.html>
- fastai. (2024). *Welcome to fastai*. fastai. Abgerufen am 29.11.2024 von <https://docs.fast.ai/#about-fastai>
- Frei, M. (2023). *Schon über 100.000 Anträge auf Förderung von Photovoltaikanlagen*. DerStandard. Abgerufen am 26.03.2023 von <https://www.derstandard.at/story/2000144862309/schon-ueber-100-000-antraege-auf-foerderung-von-photovoltaikanlagen>

- Ge, F., Wang, G., He, G., Zhou, D., Yin, R., & Tong, L. (2022). A Hierarchical Information Extraction Method for Large-Scale Centralized Photovoltaic Power Plants Based on Multi-Source Remote Sensing Images. *Remote Sensing*, 14(17), 4211.
- GeeksforGeeks. (2024). *ML Underfitting and Overfitting*. Abgerufen am 28.03.2024 von <https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/>
- Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc."
- GitHub. (2024). *Deep Learning Libraries Installers for ArcGIS*. Esri. Abgerufen am 03.08.2024 von <https://github.com/Esri/deep-learning-frameworks/blob/main/README.md>
- Goodfellow, I. (2016). Deep learning. In: MIT press.
- Goswami, S. (2020). Deep learning—a state-of-the-art approach to artificial intelligence. *Deep learning: Research and applications*, 1-19.
- Groiß, C., Boxleitner, M., & Chochole, M. (2013). Nachhaltige Stromerzeugungspotenziale in Österreich. *e & i Elektrotechnik und Informationstechnik*, 1(130), 9-15.
- Grunau, W. (2022). *Künstliche Intelligenz in Geodäsie und Geoinformatik: Potenziale und Best-Practice-Beispiele*. Wichmann, H.
- Guo, S. (2018). Development status and prospect of solar photovoltaic power generation. *Shandong Ind. Technol*, 16, 163.
- Hassaballah, M., & Awad, A. I. (2020). *Deep learning in computer vision: principles and applications*. CRC Press.
- He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. Proceedings of the IEEE international conference on computer vision,
- Homelab233. (2021). Instalación de paneles solares. In: Wikimedia Commons, CC BY-SA 4.0.
- Jiang, X., Hadid, A., Pang, Y., Granger, E., & Feng, X. (2019). Deep learning in object detection and recognition.
- Kalita, J. (2022). *Machine Learning: Theory and Practice*. CRC Press.
- Kelleher, J. D. (2019). *Deep learning*. MIT press.
- Kerim, A., Aslan, C., Celikcan, U., Erdem, E., & Erdem, A. (2021). NOVA: Rendering virtual worlds with humans for computer vision tasks. Computer Graphics Forum,
- Kesselheld. (2018). *Photovoltaik Leistung: MWp, Kilowattstunde & Wirkungsgrad*. Abgerufen am 09.12.2023 von <https://www.kesselheld.de/photovoltaik-leistung/>
- Kettner, C., Kratena, K., & Sommer, M. (2022). The socio-economic impact of renewable electricity generation with prosumer activity. *e & i Elektrotechnik und Informationstechnik*, 139(8), 624-631.
- Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., & Inman, D. J. (2021). 1D convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151, 107398.
- Kobold AI. (2024). *Künstliche neuronale Netze und Deep Learning einfach erklärt*. Abgerufen am 18.01.2024 von <https://www.kobold.ai/kuenstliche-neuronale-netze/>
- Kumar, L. A., & Renuka, D. K. (2023). *Deep Learning Approach for Natural Language Processing, Speech, and Computer Vision: Techniques and Use Cases*. CRC Press.
- La La, F. (2019). *Wie lernen neuronale Netze?* Microsoft Learn. Abgerufen am 19.01.2024 von <https://learn.microsoft.com/de-de/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn>
- Lavallin, A., & Downs, J. A. (2021). Machine learning in geography—Past, present, and future. *Geography Compass*, 15(5), e12563.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436-444.
- Lefkowitz, M. (2019). Professor's perceptron paved the way for AI – 60 years too soon. In: Cornell Chronicle.
- Marvasti-Zadeh, S. M., Goodsman, D., Ray, N., & Erbilgin, N. (2023). Early detection of bark beetle attack using remote sensing and machine learning: A review. *ACM Computing Surveys*, 56(4), 1-40.

- Miloudi, S., & Meguenni, B. (2023). Exploring the Potential of Machine and Deep Learning Models for OpenStreetMap Data Quality Assessment and Improvement (Short Paper). 12th International Conference on Geographic Information Science (GIScience 2023),
- Ng, A., & Soo, K. (2018). *Data science—was ist das eigentlich?!* Springer.
- Niederösterreichische Landesregierung. (2020). *IZ NÖ-Süd setzt auf Photovoltaik*. Amt der Niederösterreichischen Landesregierung. Abgerufen am 13.08.2024 von https://www.noel.gv.at/noel/IZ_NOe-Sued_setzt_auf_Photovoltaik.html
- NumPy Developers. (o.J.). *NumPy Documentation*. Abgerufen am 09.05.2025 von <https://numpy.org/doc/>
- NVIDIA. (o. J.). The Ultimate CUDA Development GPU. In: nvidia Developer.
- NVIDIA Developer. (2024). *Your GPU Compute Capability*. NVIDIA Developer. Abgerufen am 30.10.2024 von <https://developer.nvidia.com/cuda-gpus>
- Ofenseite. (2024). Photovoltaik und Solarthermie kombiniert: Geht das gut? In: Ofenseite.
- Openshaw, S., & Openshaw, C. (1997). *Artificial intelligence in geography*. John Wiley & Sons, Inc.
- Paál, G. (2024). *Wie viele Gigabyte hat unser Gehirn?* SWR Wissen. Abgerufen am 26.01.2024 von <https://www.swr.de/wissen/1000-antworten/wie-viele-gigabyte-hat-unser-gehirn-106.html>
- Planche, B., & Andres, E. (2019). *Hands-On Computer Vision with TensorFlow 2: Leverage deep learning to create powerful image processing apps with TensorFlow 2.0 and Keras*. Packt Publishing Ltd.
- Poonkuntran, S., Dhanraj, R. K., & Balusamy, B. (2022). *Object Detection with Deep Learning Models: Principles and Applications*. CRC Press.
- Pu, Y., Apel, D. B., Szmigiel, A., & Chen, J. (2019). Image recognition of coal and coal gangue using a convolutional neural network and transfer learning. *Energies*, 12(9), 1735.
- Python Software Foundation. (o.J.). *The Python Standard Library*. <https://docs.python.org/3/library/>
- Ramo, K. (2019). *Hands-On Java Deep Learning for Computer Vision: Implement machine learning and neural network methodologies to perform computer vision-related tasks*. Packt Publishing Ltd.
- Ren, S. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv preprint arXiv:1506.01497*.
- Riemer, C. (2009). Solar Firmengebäude. In: Wikimedia Commons, CC BY 3.0.
- Roberts, D. A., Yaida, S., & Hanin, B. (2022). *The principles of deep learning theory*. Cambridge University Press Cambridge, MA, USA.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., & Bernstein, M. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115, 211-252.
- Sampath, V., Maurtua, I., Aguilar Martin, J. J., & Gutierrez, A. (2021). A survey on generative adversarial networks for imbalance problems in computer vision tasks. *Journal of big Data*, 8, 1-59.
- Scherhauser, S., Part, F., & Beigl, P. (2020). Das Sekundärressourcenpotenzial aus Windkraft- und Photovoltaikanlagen. The potential of secondary resources from wind power and photovoltaic plants. *Österreichische Wasser- und Abfallwirtschaft*. <https://doi.org/10.1007/s00506-020-00723-3>
- Scikit-learn Developers. (o.J.). *Scikit-learn: Machine Learning in Python*. Abgerufen am 09.05.2025 von <https://scikit-learn.org/stable/api/index.html>
- Sewak, M., Karim, M. R., & Pujari, P. (2018). *Practical convolutional neural networks: implement advanced deep learning models using Python*. Packt Publishing Ltd.
- Stadtgemeinde Traiskirchen. (2024). *Photovoltaikanlagen*. Stadtgemeinde Traiskirchen. Abgerufen am 12.08.2024 von <https://www.traiskirchen.gv.at/natur-umwelt/nachhaltige-stadtentwicklung/photovoltaikanlagen/>
- Statistik Austria. (2024). *STATatlas Photovoltaikanlagen in Österreich* [Karte]. https://www.statistik.at/atlas/?mapid=them_energie_klimafonds&layerid=layer1&sublayerid=sublayer0&languageid=0&bbox=912963,5754341,2087036,6345658,8

- Sufyan, M. (2023). *What is Unsupervised Learning: A Comprehensive Guide*. Wevolver. Abgerufen am 25.01.2024 von <https://www.wevolver.com/article/what-is-unsupervised-learning-a-comprehensive-guide>
- Sugomori, Y., Kaluza, B., Soares, F. M., & Souza, A. M. (2017). *Deep learning: Practical neural networks with java*. Packt Publishing Ltd.
- Surmenok, P. (2017). *Estimating an Optimal Learning Rate For a Deep Neural Network*. Medium. Abgerufen am 12.12.2024 von <https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0>
- SuryaCreatX. (2023). AI 101: An Introduction to Artificial Intelligence for Beginners. In: Medium.
- Talluri, R. (2017). *Conventional computer vision coupled with deep learning makes AI better*. Network World. Abgerufen am 13.12.2023 von <https://www.networkworld.com/article/3239146/conventional-computer-vision-coupled-with-deep-learning-makes-ai-better.html>
- Tang, H., Post, J., Kourtellis, A., Porter, B., & Zhang, Y. (2021). Comparison of Object Detection Algorithms Using Video and Thermal Images Collected from a UAS Platform: An Application of Drones in Traffic Management. *arXiv preprint arXiv:2109.13185*.
- Technikum Wien. (2023). Jährlich in Österreich installierte PV-Leistung der Jahre 2000 bis 2022. In: Erneuerbare Energie in Österreich – Marktentwicklung 2006, Bundesministerium für Verkehr, Innovation und Technologie, Berichte aus Energie- und Umweltforschung, 11/2007.
- The Matplotlib Development Team. (o.J.). *Matplotlib: Visualization with Python*. Abgerufen am 09.05.2025 von <https://matplotlib.org/stable/users/index.html>
- The pandas Development Team. (2024). *pandas Documentation*. <https://pandas.pydata.org/docs/>
- Vasku, M. (2019). An exploration of automatic detection of large-scale solar plants: application of machine learning-based image classification in Google Earth Engine. *Geografia-Malaysian Journal of Society and Space*, 5(1), 1-10.
- Vasuki, A., & Govindaraju, S. (2017). Deep neural networks for image classification,. *Deep Learning for Image Processing Applications* (vol. 31, p. 27). In: IOS Press.
- Wang, E. K., Li, Y., Nie, Z., Yu, J., Liang, Z., Zhang, X., & Yiu, S. M. (2019). Deep fusion feature based object detection method for high resolution optical remote sensing images. *Applied Sciences*, 9(6), 1130.
- Waskom, M. (o.J.). *Seaborn: Statistical Data Visualization*. Abgerufen am 09.05.2025 von <https://seaborn.pydata.org/>
- Wissen.de. (o. J.). *Wie viele Nervenzellen haben wir?* Abgerufen am 22.01.2024 von <https://www.wissen.de/bildwb/nervenzellen-wie-viele-nervenzellen-haben-wir>
- Wlodarczak, P. (2019). *Machine learning and its applications*. CRC Press.
- Zhang, K., Sun, Q., Zhao, C., & Tang, Y. (2023). Causal reasoning in typical computer vision tasks. *Science China Technological Sciences*, 1-16.
- Zhu, X. X., Tuia, D., Mou, L., Xia, G.-S., Zhang, L., Xu, F., & Fraundorfer, F. (2017). Deep learning in remote sensing: A comprehensive review and list of resources. *IEEE geoscience and remote sensing magazine*, 5(4), 8-36.

Hilfsmittelverzeichnis

Im Sinne der Richtlinien zur Eigenständigkeit und Transparenz wissenschaftlicher Arbeiten werden in der folgenden Übersicht alle während des Arbeitsprozesses eingesetzten Hilfsmittel und Werkzeuge auf Basis künstlicher Intelligenz aufgeführt. Die Tabelle zeigt, wofür die jeweiligen Hilfsmittel eingesetzt wurden und wie deren Ergebnisse in die Arbeit eingeflossen sind.

Hilfsmittel	Verwendungszweck	Betroffene Teile	Beispiele
ChatGPT (OpenAI)	Formulierungshilfe, Strukturierung, technische Erläuterung	Rechtschreib- und Grammatikprüfungen, Umformulierung, Hilfe bei Python-Fehlermeldungen	Inhalte wurden überarbeitet, angepasst und kritisch reflektiert
DeepL Write (DeepL SE)	Sprachliche Ausarbeitung	Verbesserung von Ausdruck, Satzstruktur und Verständlichkeit	Vorschläge wurden individuell geprüft und teilweise übernommen
QuillBot (Course Hero, Inc.)	Sprachliche Ausarbeitung	Verbesserung von Ausdruck, Stil und Lesbarkeit	Vorschläge wurden sprachlich angepasst

Tab. 23: Hilfsmittelverzeichnis

Alle inhaltlichen Erkenntnisse, Analysen und Bewertungen beruhen auf eigener Leistung und wurden eigenständig erarbeitet. Die KI wurde nicht verwendet, um wissenschaftliche Inhalte zu generieren oder Aufgaben eigenständig zu bearbeiten. Stattdessen diente sie ausschließlich als Hilfsmittel zur Unterstützung des Arbeitsprozesses.

Anhang

Aus Gründen der Übersichtlichkeit wurden im Anhang nur exemplarisch ausgewählte Skripte aufgeführt. Ähnliche Skripte mit geringfügigen Abweichungen (z. B. in Variablen oder Parametern) wurden nicht vollständig aufgenommen.

Anhang A: Umbenennen von Bildchips und Annotationen für Faster R-CNN

```
1  Angleich beider Klassen (Class_1 und Class_2) auf die gleiche Anzahl an Chips und Labels
2  # Notwendige Bibliotheken importieren
3  import os
4  import shutil
5  import arcpy
6
7  # Arbeitsumgebung setzen
8  arcpy.env.workspace =
    r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\arcgislearn\
    export_training_data\0_PASCAL\TrG"
9
10 # Pfade definieren, wo sich Trainingschips und Labels befinden
11 PVA_image_path=r"C:\1\rot180_ov10\images"
12 PVA_label_path=r"C:\1\rot180_ov10\labels"
13 STA_image_path=r"C:\2\rot90_ov10\images"
14 STA_label_path=r"C:\2\rot90_ov10\labels"
15
16 # Funktion zur Umbenennung der Dateien in einem bestimmten Ordner
17 def rename_files(image_folder, label_folder, start_index):
18     image_files = sorted([f for f in os.listdir(image_folder) if
19         f.endswith('.tif')])
20     label_files = sorted([f for f in os.listdir(label_folder) if
21         f.endswith('.xml')])
22
23     if len(image_files) != len(label_files):
24         print(f"Anzahl der Bild- und Labeldateien im Ordner {image_folder} stimmt
25             nicht überein.")
26     return start_index
27
28     for i, (img_file, lbl_file) in enumerate(zip(image_files, label_files),
29         start=start_index):
30         new_name = f"{i:012d}" # Achtstellige Nummerierung
31
32         # Bild (.tif) und zugehörige TFW-Datei umbenennen
33         os.rename(os.path.join(image_folder, img_file), os.path.join(image_folder,
34             f"{new_name}.tif"))
35
36         tfw_file=img_file.replace('.tif', '.tfw')
37         if os.path.exists(os.path.join(image_folder, tfw_file)):
38             os.rename(os.path.join(image_folder, tfw_file),
39                 os.path.join(image_folder, f"{new_name}.tfw"))
40
41         # Label umbenennen
```

```

35         os.rename(os.path.join(label_folder, lbl_file), os.path.join(label_folder,
    f"{new_name}.xml"))
36
37     return start_index + len(image_files) # Neuer Startindex für die nächste
    Klasse
38
39 # Klasse 1 (PVA) umbenennen
40 next_index=rename_files(PVA_image_path, PVA_label_path, start_index=0)
41 # Klasse 2 (STA) umbenennen - Startindex nach Klasse 1 weiterführen
42 rename_files(STA_image_path, STA_label_path, start_index=next_index)
43 print("Umbenennung für beide Klassen abgeschlossen!")
44
45 # Zielordner (gesamter Datensatz)
46 comb_folder_path=r"C:\Comb\rot180_rot90_ov10"
47 comb_image_path=r"C:\Comb\rot180_rot90_ov10\images"
48 comb_label_path=r"C:\Comb\rot180_rot90_ov10\labels"
49
50 # Sicherstellen, dass die Zielordner existieren
51 os.makedirs(comb_folder_path, exist_ok=True)
52 os.makedirs(comb_image_path, exist_ok=True)
53 os.makedirs(comb_label_path, exist_ok=True)
54
55 # Funktion zum Kopieren und Umbenennen der Dateien in das Zielverzeichnis
56 def copy_and_rename(src_folder, dest_folder):
57     for filename in os.listdir(src_folder):
58         src_file = os.path.join(src_folder, filename)
59
60         if os.path.isfile(src_file):
61             # Sicherstellen, dass der Dateiname im Zielordner einzigartig ist
62             base, ext = os.path.splitext(filename)
63             counter = 1
64             new_filename = filename
65             while os.path.exists(os.path.join(dest_folder, new_filename)):
66                 new_filename = f"{base}_{counter}{ext}"
67                 counter += 1
68
69             dest_file = os.path.join(dest_folder, new_filename)
70             shutil.copy(src_file, dest_file)
71
72 # Dateien aus Klasse 1 und Klasse 2 in den gemeinsamen Ordner kopieren
73 copy_and_rename(PVA_image_path, comb_image_path)
74 copy_and_rename(PVA_label_path, comb_label_path)
75 copy_and_rename(STA_image_path, comb_image_path)
76 copy_and_rename(STA_label_path, comb_label_path)
77 print("Zusammenführung der Klassen 1 und 2 in den Zielordner abgeschlossen!")
78
79 ## Die restlichen Dateien von Klasse 1 in den "comb"-Ordner kopieren
80 # Pfad, wo sich die zu kopierenden Dateien befinden
81 PVA_folder=r"C:\TrG\1\rot180_ov10"
82

```

```
83 # Kopieren der zusätzlichen Dateien
84 additional_files = ["esri_accumulated_stats.json", "esri_model_definition.emd",
85                    "map.txt", "stats.txt"]
86 for file in additional_files:
87     src_file = os.path.join(PVA_folder, file) # Hier befinden sich die Dateien
88     if os.path.isfile(src_file):
89         shutil.copy(src_file, combined_folder_path)
90 print("Das Kopieren der zusätzlichen Dateien ist abgeschlossen!")
```

Anhang B: Export von Trainingsdaten für Mask R-CNN (Kombinierter Datensatz)

```

90 Trainingsdaten für beide Klassen (Class_1 und Class_2) erstellen
91 # Notwendige Bibliotheken importieren
92 import arcpy
93 from arcpy import env
94 from arcpy.ia import *
95
96 # Lizenz der ArcGIS Image Analyst Extension auf Gültigkeit kontrollieren
97 arcpy.CheckOutExtension("ImageAnalyst") # Antwort ist "CheckedOut"
98
99 ## Beide Klassen (Class_1 & Class_2) exportieren - Photovoltaikanlagen (PVA) &
Solarthermieranlagen (STA)
100 # Liste der Input-Feature Classes, Zielordner und der jeweiligen Rotationswerte
101 input_values=[
102     {"out_folder":
103         r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\arcgislearn\export_training_data\1_RCNNMasks\TrG\1\rot180_ov10", "in_class_data":
104         r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\MA.gdb\Raster_All_LabeledObjects_v2_PVA", "rotation_angle": 180},
105     {"out_folder":
106         r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\arcgislearn\export_training_data\1_RCNNMasks\TrG\2\rot90_ov10", "in_class_data":
107         r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\MA.gdb\Raster_All_LabeledObjects_v2_STA", "rotation_angle": 90}
108 ]
109
110 # Pfad zum Lokalen Input-Raster definieren
111 in_raster=r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\MA.gdb\Raster_All"
112
113 # Export-Einstellungen und Schleife für den Export festlegen
114 for iv in input_values:
115     out_folder=iv["out_folder"]
116     in_class_data=iv["in_class_data"]
117     rotation_angle=iv["rotation_angle"]
118
119     # Export-Parameter konfigurieren
120     with arcpy.EnvManager(extent='-4800 316800 700 326500
121         PROJCS["MGI_Austria_GK_East".GEOGCS["GCS_MGI".DATUM["D_MGI".SPHEROID["Bessel_1841".
122         6377397.155.299.1528128]].PRIMEM["Greenwich".0.0].UNIT["Degree".0.0174532925199433]
123         ].PROJECTION["Transverse_Mercator"].PARAMETER["False_Easting".0.0].PARAMETER["False_Northing".-
124         5000000.0].PARAMETER["Central_Meridian".16.33333333333333].PARAMETER["Scale_Factor"
125         .1.0].PARAMETER["Latitude_Of_Origin".0.0].UNIT["Meter".1.0]]'):
126         arcpy.ia.ExportTrainingDataForDeepLearning(
127             in_raster=in_raster,
128             out_folder=out_folder,
129             in_class_data=in_class_data,
130             image_chip_format="TIFF",
131             tile_size_x=256,
132             tile_size_y=256,
133             stride_x=128,

```

```
125         stride_y=128,
126         output_nofeature_tiles="ONLY_TILES_WITH_FEATURES",
127         metadata_format="RCNN_Masks",
128         start_index=0,
129         class_value_field=None,
130         buffer_radius=0,
131         rotation_angle=rotation_angle, # Bilddaten werden bei dieser
    Einstellung zusätzlich um 90° gedreht und angelegt
132         reference_system="MAP_SPACE",
133         processing_mode="PROCESS_AS_MOSAICKED_IMAGE",
134         blacken_around_feature="NO_BLACKEN",
135         crop_mode="FIXED_SIZE",
136         in_raster2=None,
137         in_instance_data=None,
138         instance_class_value_field=None,
139         min_polygon_overlap_ratio=0.1
140     )
141
142     print(f"Export für {in_class_data} mit Rotation {rotation_angle} in
    {out_folder} abgeschlossen.")
```

Anhang C: Vorbereiten und Trainieren von Trainingsdaten für Faster R-CNN

Datensatz 1 (30 Epochen und Datenaugmentation)

143 **Daten vorbereiten für eine Klasse (Class_1) – Photovoltaikanlagen (PVA)**

```

144 # Notwendige Bibliotheken importieren
145 import arcgis.learn
146 from arcgis.learn import prepare_data, FasterRCNN
147 # Weitere Bibliotheken importieren
148 import subprocess
149 import time
150 import threading
151
152 # Fast.ai-Modul für Transformationen importieren
153 from fastai.vision import get_transforms
154 from fastai.vision.transform import get_transforms
155
156 # Fast.ai-Transformationen für Trainingsdatensatz definieren
157 train_tf, _ = get_transforms(
158     #rotation,                # bereits beim Export angewendet
159     flip_vert=True,          # vertikales Spiegeln
160     max_lighting=0.4         # maximale Beleuchtung
161 )
162 # Keine Transformationen für den Validierungsdatensatz
163 valid_tf = []
164
165 ## Variable definieren
166 # Pfad zu Trainingsdaten definieren, die in ExportTrainingData.ipynb erstellt
  wurden
167 path_trainingData=r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\arcgislearn\ex
  port_training_data\0_PASCAL\TrG\1\rot180_ov10"
168
169 # Trainings- und Validierungsdaten vorbereiten
170 trainingData=prepare_data(
171     path=path_trainingData,
172     chip_size=256,
173     val_split_pct=0.20,
174     batch_size=32,           # wurde bereits mittels
    arcgis.learn.estimate_batch_size abgeschätzt
175     transforms=(train_tf, valid_tf) # Transformationen für Trainings- und
    Validierungsdatensatz
176 )
177 # Modell definieren – FasterRCNN
178 # frcnn_model10tf_Lr          # FasterRCNN-Modell mit Lernrate 0.001
179 frcnn_model30tf=FasterRCNN(  # FasterRCNN-Modell mittels lr_find() ermittelter
    Lernrate
180     data=trainingData,
181     backbone="resnet50",
182     pretrained_path=None
183 )
184

```

```
185 # Für FasterRCNN verfügbare Backbones ermitteln
186 frcnn_model30tf.backbones() # "resnet18", "resnet34", "resnet152" uvm.
187
188 # Batchgrösse abschätzen
189 # arcgis.learn.estimate_batch_size(model=frcnn_model30tf) # Ergebnis ist 32
190
191 # Ein paar Trainingschips visualisieren
192 trainingData.show_batch(10)
193 # Output, um die Aufteilung der Annotationen (Label) zu sehen
194 trainingData # für Training: 5159 Label, für Validierung: 1289 Label
195
196 # Performanz des Modells anzeigen
197 frcnn_model30tf.show_results()
198
199 # Optimale Lernrate finden und Ergebnis abrufen
200 lr = frcnn_model30tf.find(allow_plot=True)
201 lr # Ergebnis ist 0.00019054607179632462
202
203 ## GPU-Auslastung - GPU-Einträge des Befehls nvidia-smi mittels Logdatei beobachten
204 # Pfad zum Speicherort der Logdatei definieren
205 path_log=r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\scripts\fasterRCNN\logf
    ile\log_gpu_frcnn_1_30ep_tf.txt"
206 # Monitoring-Funktion erstellen und Logdatei generieren
207 def monitor_gpu_usage(interval=900):
208     with open(path_log, "a") as log_file:
209         while True: # Dauerschleife, die mit dem Daemon-Thread endet
210             result=subprocess.run(["nvidia-smi"], stdout=subprocess.PIPE,
                text=True)
211             log_file.write(result.stdout)
212             log_file.write("\n" + "*" * 91 + "\nNeuer GPU-Eintrag:\n")
213 # Trennlinie generieren
214             time.sleep(interval)
215
216 # Monitoring-Thread erstellen und als Daemon festlegen (für ein automatisches
    Beenden)
217 monitor_thread=threading.Thread(target=monitor_gpu_usage, args=(900,), daemon=True)
218 monitor_thread.start()
219
220 # Für Faster R-CNN verfügbare Messgrößen ermitteln
221 frcnn_model10tf.available_metrics # valid_loss, average_precision
222
223 ## Modelltraining - Versuch 1 mit via lr_find() ermittelter Lernrate
224 # Modell trainieren - Versuch 1
225 frcnn_model10tf.fit(
226     epochs=30, # 30 Epochen
227     lr=lr, # Lernrate
228     early_stopping=True
229 )
230
231 # Verlustdiagramm/-kurve erstellen
```

```
232 frcnn_model30tf.learn.recorder.plot_losses()
233 # Ergebnisse von Trainings- und Validierungsdatensatz anzeigen (Ground Truth vs.
    Vorhersagen)
234 frcnn_model30tf.show_results()
235
236 ## Modelltraining - Versuch 2 mit Lernrate 0.001
237 # Modell trainieren - Versuch 2
238 frcnn_model30tf_lr.fit(
239     epochs=30,          # 30 Epochen
240     lr=0.001,           # Lernrate
241     early_stopping=True
242 )
243
244 # Verlustdiagramm/-kurve erstellen
245 frcnn_model30tf_lr.learn.recorder.plot_losses()
246 # Ergebnisse von Trainings- und Validierungsdatensatz anzeigen (Ground Truth vs.
    Vorhersagen)
247 frcnn_model30tf_lr.show_results()
248
249 # Monitoring nach dem Training beenden
250 monitor_running=False # Setzt das Flag auf False, um den Thread zu beenden
251
252 ## FasterRCNN-Modell abspeichern
253 # Notwendige Module importieren
254 import os
255 import pathlib
256
257 # Variablen definieren und Modell abspeichern
258 frcnn_model30tfName="model"
259 frcnn_model30tfFolder=r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\arcgislearn\save_models\fasterRCNN\1\30ep\with_trans"
260 frcnn_model30tfPath=os.path.join(frcnn_model30tfFolder, frcnn_model30tfName)
261 frcnn_model30tf.save(frcnn_model30tfPath)
```

Anhang D: Vorbereiten und Trainieren von Trainingsdaten für Mask R-CNN – Kombierter Datensatz (10 Epochen und Datenaugmentation)

262 **Daten vorbereiten für beide Klassen (Class_1 und Class_2) – PVA und STA**

```

263 # Import von Bibliotheken
264 import tensorflow as tf
265 from tensorflow.python.client import device_lib
266
267 # Verfügbarkeit von GPUs anzeigen lassen
268 print("Anzahl verfügbarer GPUs: ",
      len(tf.config.experimental.list_physical_devices("GPU")))
269
270 # GPU-Speicherverfügbarkeit prüfen
271 from tensorflow.python.client import device_lib
272 print(device_lib.list_local_devices())
273
274 # Notwendige Bibliotheken importieren
275 import arcgis.learn
276 from arcgis.learn import prepare_data, MaskRCNN
277 # Weitere Bibliotheken importieren
278 import subprocess
279 import time
280 import threading
281
282 # Fast.ai-Modul für Transformationen importieren
283 from fastai.vision import get_transforms
284 from fastai.vision.transform import get_transforms
285
286 # Fast.ai-Transformationen für Trainingsdatensatz definieren
287 train_tf, _ = get_transforms(
288     #rotation,                # bereits beim Export angewendet
289     flip_vert=True,          # vertikales Spiegeln
290     max_lighting=0.4         # maximale Beleuchtung
291 )
292 # Keine Transformationen für den Validierungsdatensatz
293 valid_tf=[]
294
295 ## Variable definieren
296 # Pfad zu Trainingsdaten definieren, die in ExportTrainingData.ipynb erstellt
  wurden
297 path_trainingData=r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\arcgislearn\ex
  port_training_data\1_RCNNMasks\TrG\CombEqu"
298
299 # Trainings- und Validierungsdaten vorbereiten
300 trainingData=prepare_data(
301     path=path_trainingData,
302     chip_size=256,
303     val_split_pct=0.20,
304     batch_size=16,           # wurde bereits mittels
      arcgis.learn.estimate_batch_size abgeschätzt

```

```
305     transforms=(train_tf, valid_tf) # Transformationen für Trainings- und
      Validierungsdatensatz
306 class_balancing=True,             # Klassenbalance aktivieren (maskiert die
      NoData-Werte)
307     class_mapping={1: "Class_1", 2: "Class_2"},
308 )
309 # Klassenverteilung prüfen
310 print("Klassenverteilung:")
311 print(trainingData.class_mapping)
312
313 # Modell definieren - MaskRCNN
314 Mrcnn2_model10tf_lr=MaskRCNN( # MaskRCNN-Modell mit Lernrate von 0.001
315     data=trainingData,
316     backbone="resnet50",
317     pretrained_path=None
318 )
319
320 # Für MaskRCNN verfügbare Backbones ermitteln
321 mrcnn2_model10tf_lr.backbones() # "resnet18", "resnet34", "resnet152" uvm.
322
323 # Batchgrösse abschätzen
324 # arcgis.learn.estimate_batch_size(model=mrcnn2_model10tf_lr) # Ergebnis ist 16
325
326 # Ein paar Trainingschips visualisieren
327 trainingData.show_batch(10)
328 # Output, um die Aufteilung der Annotationen (Label) zu sehen
329 trainingData # für Training: 9031 Label, für Validierung: 2257 Label
330
331 # Performanz des Modells anzeigen
332 mrcnn2_model10tf_lr.show_results()
333
334 # Optimale Lernrate finden und Ergebnis abrufen
335 #lr = mrcnn2_model10tf_lr.find(allow_plot=True)
336 #lr # Ergebnis ist 0.00019054607179632462
337
338 ## GPU-Auslastung - GPU-Einträge des Befehls nvidia-smi mittels Logdatei beobachten
339 # Pfad zum Speicherort der Logdatei definieren
340 path_log=r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\scripts\maskRCNN\logfil
      e\log_gpu_mrcnn_comb_10ep_tf_lr.txt"
341 # Monitoring-Funktion erstellen und Logdatei generieren
342 def monitor_gpu_usage(interval=900):
343     with open(path_log, "a") as log_file:
344         while True: # Dauerschleife, die mit dem Daemon-Thread endet
345             result=subprocess.run(["nvidia-smi"], stdout=subprocess.PIPE,
346                 text=True)
347             log_file.write(result.stdout)
348             log_file.write("\n" + "*" * 91 + "\nNeuer GPU-Eintrag:\n")
349 # Trennlinie generieren
349     time.sleep(interval)
350
```

```
351 # Monitoring-Thread erstellen und als Daemon festlegen (für ein automatisches
    Beenden)
352 monitor_thread=threading.Thread(target=monitor_gpu_usage, args=(900,), daemon=True)
353 monitor_thread.start()
354
355 # Für Mask R-CNN verfügbare Messgrößen ermitteln
356 mrcnn2_model10tf_lr.available_metrics # valid_loss, average_precision
357
358 ## Modelltraining - Versuch 2 mit Lernrate 0.001
359 # Modell trainieren - Versuch 2
360 mrcnn2_model10tf_lr.fit(
361     epochs=30,          # 30 Epochen
362     lr=0.001,          # Lernrate
363     early_stopping=True
364 )
365
366 # Verlustdiagramm/-kurve erstellen
367 mrcnn2_model10tf_lr.learn.recorder.plot_losses()
368 # Ergebnisse von Trainings- und Validierungsdatensatz anzeigen (Ground Truth vs.
    Vorhersagen)
369 mrcnn2_model10tf_lr.show_results()
370
371 # Monitoring nach dem Training beenden
372 monitor_running=False # Setzt das Flag auf False, um den Thread zu beenden
373
374 ## MaskRCNN-Modell abspeichern
375 # Notwendige Module importieren
376 import os
377 import pathlib
378
379 # Variablen definieren und Modell abspeichern
380 mrcnn2_model10tf_lrName="model"
381 mrcnn2_model10tf_lrFolder=r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\arcgis
    learn\save_models\fasterRCNN\1\30ep\with_trans"
382 mrcnn2_model10tf_lrPath=os.path.join(mrcnn2_model10tf_lrFolder,
    mrcnn2_model10tf_lrName)
383 mrcnn2_model10tf_lr.save(mrcnn2_model10tf_lrPath)
```

Anhang E: Erkennen von Objekten mittels Deep Learning – Faster R-CNN (Datensatz 1)

384 Erkennung von Photovoltaikanlagen (PVA) auf dem Testgebiet – Klasse (Class_1)

```

385 # Notwendige Bibliotheken importieren
386 import arcpy
387 from arcpy import env
388 from arcpy.ia import *
389
390 # Lizenz der ArcGIS Image Analyst Extension kontrollieren
391 arcpy.CheckOutExtension("ImageAnalyst") # Antwort ist "CheckedOut"
392
393 ## Variablen definieren
394 # Testgebiet Hennersdorf
395 in_raster="C:\Users\ace_1\Documents\ArcGIS\Projects\MA\MA.gdb\ClipRaster_Testgebiet_HE"
396
397 # Arbeitsumgebung setzen
398 arcpy.env.workspace="C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\arcgislearn"
399
400 # Erkannte Objekte (30 Epochen)
401 out_detected_objects="C:\detect_objects\fasterRCNN\1\30ep\TeG_HE\with_trans\detect_PVA_30ep_tf.shp"
402 # Modeldefinition (30 Epochen)
403 in_model_definition="C:\save_models\fasterRCNN\1\30ep\with_trans\model\model.emd"
404
405 # Objekte erkennen
406 with arcpy.EnvManager(
407     outputCoordinateSystem='PROJCS["MGI_Austria_GK_East",GEOGCS["GCS_MGI",DATUM["D_MGI",SPHEROID["Bessel_1841",6377397.155,299.1528128]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION["Transverse_Mercator"],PARAMETER["False_Easting",0.0],PARAMETER["False_Northing",-5000000.0],PARAMETER["Central_Meridian",16.3333333333333],PARAMETER["Scale_Factor",1.0],PARAMETER["Latitude_Of_Origin",0.0],UNIT["Meter",1.0]]',
408     extent='1100 329350 3200 330885.2
PROJCS["MGI_Austria_GK_East".GEOGCS["GCS_MGI".DATUM["D_MGI".SPHEROID["Bessel_1841".6377397.155.299.1528128]].PRIMEM["Greenwich".0.0].UNIT["Degree".0.0174532925199433]].PROJECTION["Transverse_Mercator"].PARAMETER["False_Easting".0.0].PARAMETER["False_Northing".-5000000.0].PARAMETER["Central_Meridian".16.3333333333333].PARAMETER["Scale_Factor".1.0].PARAMETER["Latitude_Of_Origin".0.0].UNIT["Meter".1.0]]',
409     processorType="GPU",
410     scratchWorkspace=""):
411     result=arcpy.ia.DetectObjectsUsingDeepLearning(
412         in_raster,
413         out_detected_objects,
414         in_model_definition,
415         arguments="padding 64;threshold 0,5;batch_size 32;exclude_pad_detections
True",
416         run_nms="NMS",

```

```
417     class_value_field="Class",  
418     max_overlap_ratio=0.2  
419 )
```

Anhang F: Modellbewertung mittels Konfusionsmatrix für Mask R-CNN

420 Klassifikatorische Modellbewertung mittels Konfusionsmatrix für Mask R-CNN (Kombinierter Datensatz und Datensatz 1) ausführen

```

421 import pandas as pd
422 import seaborn as sns
423 import matplotlib.pyplot as plt
424 from sklearn.metrics import confusion_matrix, classification_report
425
426 ## MaskRCNN - Kombinerter Datensatz
427 # CSV-Datei Laden
428 df=
    pd.read_csv(r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\scripts\ArcGIS\Konfu
        sionsmatrix\MaskRCNN_Comb\csv\final_maskRCNN_Comb.csv", sep=";")
429
430 # Spalten "GroundTruth" und "Prediction" heranziehen
431 y_true=df["Ground_Truth"]
432 y_pred=df["Prediction"]
433
434 # Konfusionsmatrix und Report
435 cm=confusion_matrix(y_true, y_pred, labels=["PVA", "STA", "none"])
436 print("Konfusionsmatrix:\n", cm)
437
438 # Zusätzliche Metriken
439 print("\nKlassifikationsbericht:\n")
440 print(classification_report(y_true, y_pred, labels=["PVA", "STA"]))
441
442 # Konfusionsmatrix visualisieren
443 %matplotlib inline
444 plt.figure(figsize=(6, 4))
445 sns.heatmap(cm, annot=True, fmt="d", cmap="PuBu", xticklabels=["PVA", "STA",
    "none"], yticklabels=["PVA", "STA", "none"])
446 plt.xlabel("Vorhersage")
447 plt.ylabel("Ground Truth")
448 plt.title("Konfusionsmatrix MaskRCNN (kombiniert)")
449 plt.tight_layout()
450 plt.show()
451
452 ## MaskRCNN - Datensatz 1
453 # CSV-Datei Laden
454 df=
    pd.read_csv(r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\Data\scripts\ArcGIS\Konfu
        sionsmatrix\MaskRCNN_Comb\csv\final_maskRCNN_Class1.csv", sep=";")
455
456 # Spalten "GroundTruth" und "Prediction" heranziehen
457 y_true=df["Ground_Truth"]
458 y_pred=df["Prediction"]
459
460 # Konfusionsmatrix und Report
461 cm=confusion_matrix(y_true, y_pred, labels=["PVA", "none"])
462 print("Konfusionsmatrix:\n", cm)
463

```

```
464 # Zusätzliche Metriken
465 print("\nKlassifikationsbericht:\n")
466 print(classification_report(y_true, y_pred, labels=["PVA", "none"]))
467
468 # Konfusionsmatrix visualisieren
469 plt.figure(figsize=(6, 4))
470 sns.heatmap(cm, annot=True, fmt="d", cmap="PuBu", xticklabels=["PVA", "none"],
             yticklabels=["PVA", "none"])
471 plt.xlabel("Vorhersage")
472 plt.ylabel("Ground Truth")
473 plt.title("Konfusionsmatrix MaskRCNN (Photovoltaikanlagen)")
474 plt.tight_layout()
475 plt.show()
```

Anhang G: Balkendiagramm für PVA aus den Ergebnissen der Konfusionsmatrix erstellen (Kombinierter Datensatz und Datensatz 1)

```

476 # Bibliotheken importieren
477 import matplotlib.pyplot as plt
478 import numpy as np
479
480 # Modellnamen definieren
481 models=["Mask R-CNN (PVA & STA)", "Mask R-CNN (PVA)", "Faster R-CNN (PVA & STA)",
         "Faster R-CNN (PVA)"]
482
483 # Precision, Recall, F1-Score für PVA
484 precision_pva = [0.88, 0.87, 0.50, 0.45]
485 recall_pva    = [0.56, 0.58, 0.86, 0.89]
486 f1_pva        = [0.69, 0.70, 0.64, 0.60]
487
488 x=np.arange(len(models)) # Position der Balken
489 breite=0.25 # Balkenbreite
490
491 # Plot erstellen
492 fig, ax=plt.subplots(figsize=(10, 6))
493 ax.bar(x - breite, precision_pva, width=breite, label="Precision",
         color="steelblue")
494 ax.bar(x, recall_pva, width=breite, label="Recall", color="lightsteelblue")
495 ax.bar(x + breite, f1_pva, width=breite, label="F1-Score", color="whitesmoke")
496
497 # Achsen-Beschriftungen und Design definieren
498 ax.set_ylabel("Wert")
499 ax.set_title("Metrikenvergleich - Klasse PVA")
500 ax.set_xticks(x)
501 ax.set_xticklabels(models, rotation=20, ha="right")
502 ax.set_ylim(0, 1.0)
503 ax.legend()
504 ax.grid(True, axis="y", linestyle="--", alpha=0.5)
505 plt.tight_layout()
506 plt.show()

```

Anhang H: Modellbewertung mittels Intersection over Union (IoU) für Faster R-CNN

507 Geometrische Modellbewertung mittels IoU für Faster R-CNN (Datensatz 1) ausführen

```

508 # Bibliothek importieren
509 import arcpy
510
511 # Arbeitsumgebung setzen
512 arcpy.env.workspace=r"C:\Users\ace_1\Documents\ArcGIS\Projects\MA\MA.gdb"
513
514 # Eingabe-Feature Classes für beide Werkzeuge definieren (Ground Truth &
    Modellvorhersagen)
515 inputFeatures_frcnn1=["Testgebiet_HE_LabeledObjects_PVA_bereinigt",
    "nms_detectPVA_30ep_tf"]
516
517 ## Ausgabe-Feature Classes definieren
518 # Ausgabe-Feature Class für Intersect definieren
519 outputIntersect_frcnn1="IoU_Intersect_frcnn_PVA"
520 # Ausgabe-Feature Class für Union
521 outputUnion_frcnn1="IoU_Union_frcnn_PVA"
522
523 # Intersect durchführen
524 arcpy.analysis.Intersect(inputFeatures_frcnn1, outputIntersect_frcnn1, "ALL", "",
    "INPUT")
525 print(f"Intersect wurde erstellt: {outputIntersect_frcnn1}")
526
527 # Union durchführen
528 arcpy.analysis.Union(inputFeatures_frcnn1, outputUnion_frcnn1, "ALL", "", "GAPS")
529 print(f"Union wurde erstellt: {outputUnion_frcnn1}")
530
531 # Eingabedateien definieren
532 intersectFC_frcnn1="IoU_Intersect_frcnn_PVA"
533 unionFC_frcnn1="IoU_Union_frcnn_PVA"
534
535 ## Berechnung der Gesamtfläche
536 # Berechnung der Gesamtfläche für Intersect
537 intersectSum_frcnn1=arcpy.Statistics_analysis(intersectFC_frcnn1,
    "IntersectStats_frcnn_PVA", [{"Shape_Area", "SUM"}])
538 # Berechnung der Gesamtfläche für Union
539 unionSum_frcnn1=arcpy.Statistics_analysis(unionFC_frcnn1, "UnionStats_frcnn_PVA",
    [{"Shape_Area", "SUM"}])
540
541 # Werte extrahieren
542 intersectArea_frcnn1=float(arcpy.da.SearchCursor(intersectSum_frcnn1,
    ["SUM_Shape_Area"]).next()[0])
543 unionArea_frcnn1=float(arcpy.da.SearchCursor(unionSum_frcnn1,
    ["SUM_Shape_Area"]).next()[0])
544
545 # Berechnung der IoU
546 IoU=intersectArea_frcnn1 / unionArea_frcnn1
547 print(f"IoU für Faster R-CNN (Class_1): {IoU}")

```

Anhang I: Balkendiagramme aus den Ergebnissen von IoU erstellen (Datensatz 1)

```
548 # Bibliotheken importieren
549 import matplotlib.pyplot as plt
550 import numpy as np
551
552 # IoU-Werte definieren
553 models=["Faster R-CNN", "Mask R-CNN"]
554 iouValues=[0.2763, 0.4710]
555 colors=["white", "white"]
556 hatches=["--", ".."] # Schraffuren für Faster R-CNN (--) und Mask R-CNN (..)
557
558 # Erstellung eines einfachen Balkendiagramms
559 %matplotlib inline
560 plt.figure(figsize=(8, 5))
561 bars = plt.bar(models, iouValues, color=colors, edgecolor="gray", alpha=0.8,
562               linewidth=0.5)
563
564 # Schraffuren auf die Balken anwenden
565 for bar, hatch in zip(bars, hatches):
566     bar.set_hatch(hatch)
567
568 # Achsen und Titel setzen
569 plt.ylabel("IoU-Wert")
570 plt.title("IoU-Werte der beiden Modelle")
571 plt.show()
```