



Fachhochschule Nordwestschweiz

Hochschule für Informatik

Master of Science in Engineering

Edge ML Camera for Citizen Science

Author:	Andri Wild
Advisor:	Prof. Thomas Amberg
Start Date:	16.09.2024
Submission Date:	07.02.2025

Abstract

Das SNF-Projekt „Mitwelten“ untersucht mithilfe von IoT-Technologie Biodiversität im urbanen Gebiet. Ein Teilprojekt erkennt mit Raspberry Pi Kameras Bestäuber-Arten auf Blumen, wobei die ML-basierte Detektion aus Performance Gründen in einem zentralen Cloud-Backend erfolgt.

Diese Arbeit erforscht, ob die ML-Pipeline auf eine Edge ML Kamera portiert werden kann, um Citizen-Science-Projekte durch den Einsatz von Edge-Computing ohne zentrales Backend zu ermöglichen. Die fortschreitende Entwicklung von Edge-Hardware senkt den Infrastrukturaufwand, sodass forschende Machine-Learning-Anwendungen eigenständig und unkomplizierter betreiben können.

In der ersten Phase wurden verschiedene ML-Frameworks und Hardware anhand von Performance-Messungen verglichen. Die Messungen zeigen deutliche Unterschiede zwischen den Frameworks, wobei moderne Beschleuniger signifikant bessere Ergebnisse bei geringerem Energiebedarf liefern.

Anschliessend wird eine Implementation einer Edge ML Kamera vorgestellt, welche die Bestäuber Detektion aus dem Mitwelten Projekt umsetzt. Anhand von Prototypen werden verschiedene Setups bewertet, welche für den Einsatz infrage kommen.

Die Resultate dieser Arbeit zeigen auf, dass eine Verlagerung von Cloud zu Edge inzwischen lohnenswert sein kann, wodurch eine dezentrale Datenverarbeitung ermöglicht wird, die unter anderem zu Echtzeitanalysen, erhöhter Datensicherheit und geringerem Infrastrukturaufwand führt.

Inhaltsverzeichnis

1 Einleitung	5
1.1 Ausgangslage	5
1.2 Zielsetzung	5
1.3 Abgrenzung	6
1.4 Aufbau des Berichts	6
2 Grundlagen	7
2.1 Citizen Science	7
2.2 Machine Learning Grundlagen	9
2.2.1 Frameworks	9
2.2.2 Anatomie von Machine Learning Modellen	11
2.2.3 Geschwindigkeit einer Inferenz	13
2.3 Edge Computing	14
2.3.1 Edge Intelligence	15
2.3.2 Hostsysteme	16
2.3.3 Beschleuniger	16
2.4 Mitwelten Biodiversitäts Monitoring	19
2.4.1 Bestäuberanalyse	20
2.4.2 Edge Architektur	21
3 Analyse	22
3.1 Zielgruppe	22
3.2 Use Cases	22
3.2.1 Referenz Szenario	22
3.3 Anforderungen	22
3.3.1 Funktionale	23
3.3.2 Nicht Funktionale	23
3.4 Evaluation	23
3.4.1 Methodik	24
3.4.2 ML Framework	25
3.4.3 Hardware	26
3.5 Edge ML Setups	29
4 Umsetzung	31
4.1 Systemarchitektur	31
4.1.1 Komponentendiagramm	31

4.1.2 Qualitäten	32
4.1.3 Konfiguration	32
4.1.4 Webinterface	33
4.2 Software Design	34
4.3 Prototypen	36
4.3.1 Mitwelten Analyse	36
4.3.2 AI-Camera	39
4.4 Versuchsaufbau und Temperaturentwicklung	40
4.5 Erweiterung und Adaptierung der Edge ML Kamera	41
5 Validierung	42
5.1 Bewertung der Prototypen	42
5.1.1 Mitwelten-Analyse	42
5.1.2 AI-Kamera	44
5.2 Konkurrenzprodukte	44
6 Fazit	46
Abbildungsverzeichnis	47
Bibliographie	49

1 Einleitung

1.1 Ausgangslage

Das SNF Projekt „Mitwelten - Medienökologische Infrastrukturen für Biodiversität“ hat IoT-Technologie in urbanen Naturgebieten installiert, um dort vorhandene Tiere, Pflanzen und Umweltbedingungen genauer zu untersuchen. Ein Teilprojekt beschäftigte sich mit der Erkennung von Bestäuber-Arten auf Blumenblüten mittels Raspberry Pi Kameras. Die Machine-Learning-basierte Detektion und Kategorisierung geschieht zurzeit in einem zentralen Cloud-Backend.

Diese Architekturlösung bringt folgende Konsequenzen mit sich: Zum Einen muss der Betrieb eines Backends sichergestellt sein. Dies erfordert einen gewissen Wartungsaufwand und insbesondere ein erhöhtes technisches Verständnis für die Projektumsetzung. Für Forschende bedeutet dies, dass zusätzliches IT-Fachpersonal für den Aufbau und Betrieb der IT-Infrastruktur benötigt wird. Zum anderen werden die aufgenommenen Bilder zur Analyse über Mobilfunknetze versendet, wodurch die Betriebskosten erhöht werden und das Einsatzgebiet auf dessen Abdeckung begrenzt. Beide der genannten Aspekte sind mit zusätzlichen Kosten verbunden und erhöhen die Gesamtkomplexität des Projekts.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist die Portierung der Machine-Learning Pipeline des SNF Projekts Mitwelten auf eine Edge ML Kamera, um Citizen Science Projekte zu ermöglichen:

Dank der fortschreitenden Entwicklung von Edge-Computing-Hardware sind auf Machine Learning spezialisierte Geräte zu einem vergleichbaren Preis wie konventionelle Edge-Computer erhältlich. Dies wirft die Frage auf, ob durch diesen technologischen Fortschritt die IT-Infrastruktur eines Edge-Kamera-basierten Systems so weit reduziert werden kann, dass dieselbe Funktionalität ohne zentrales Backend umsetzbar ist. Der Einsatz von Edge-basierten Systemen könnte den Zugang zu Machine Learning Projekten für Forschende erheblich erleichtern, da ein System ohne zentrale Abhängigkeit mit weniger Aufwand betreibbar ist. Diese Stand-alone-Lösung öffnet auch die Türen für Citizen Science Projekte. Einzelpersonen oder Gemeinschaften können so eigenständig und unabhängig Forschungsprojekte durchführen. Daraus ergeben sich die folgenden Forschungsfragen:

- Welche Hardware und Frameworks gibt es, um Machine Learning (ML) dezentral, im Feld, mittels Edge Computing umzusetzen?
- Was sind Anforderungen an eine ML-Kamera für typische Citizen Science Anwendungen im Bereich Biodiversitätsmonitoring?
- Wie sieht eine konkrete Edge ML Kamera aus, für Monitoring von Biodiversität, wie im Projekt SNF Mitwelten angewendet?

1.3 Abgrenzung

Ein System zur Analyse der Biodiversität mittels Machine Learning beinhaltet viele verschiedene Aspekte. In diesem Projekt liegt der Fokus auf der aktuellsten Hardware und darauf, wie diese für Edge-ML eingesetzt werden kann. Nicht Teil dieses Projektes ist das Trainieren von Machine Learning Modellen und deren Evaluation.

1.4 Aufbau des Berichts

Der Bericht besteht im Wesentlichen aus vier Teilen. Im ersten Teil werden die Grundlagen vermittelt, welche relevant sind für das Verständnis des Berichts. Anschliessend folgt die Analyse, welche nebstden Zielgruppen und Anforderungen auch verschiedene Software-Frameworks und Hardware untersucht. In der Umsetzung wird eine konkrete Implementierung einer Edge ML Kamera vorgestellt und abschliessend wird die Evaluation der Resultate vorgenommen.

2 Grundlagen

Dieses Kapitel beschreibt den theoretischen Inhalt dieser Arbeit. Um die späteren Analysen und Evaluationen verstehen zu können, werden die grundlegenden Konzepte, Technologien und Frameworks vorgestellt, die in diesem Projekt zum Einsatz kommen. Dazu gehören die Prinzipien des Machine Learnings, die Rolle von Edge Computing sowie die eingesetzten Hardware- und Softwarekomponenten.

2.1 Citizen Science

Das grundlegende Ziel dieser Arbeit ist die Entwicklung eines Systems, welches im Bereich von Citizen Science eingesetzt werden kann. Daher soll als erstes der Begriff erläutert werden. Eine treffende Definition von Citizen Science liefert das Grünbuch Citizen Science Strategie 2020 für Deutschland:

„Citizen Science beschreibt die Beteiligung von Personen an wissenschaftlichen Prozessen, die nicht in diesem Wissenschaftsbereich institutionell gebunden sind. Dabei kann die Beteiligung in der kurzzeitigen Erhebung von Daten bis hin zu einem intensiven Einsatz von Freizeit bestehen, um sich gemeinsam mit Wissenschaftlerinnen bzw. Wissenschaftlern und/oder anderen Ehrenamtlichen in ein Forschungsthema zu vertiefen. Obwohl viele ehrenamtliche Forscherinnen und Forscher eine akademische Ausbildung aufweisen, ist dies keine Voraussetzung für die Teilnahme an Forschungsprojekten. Wichtig ist allerdings die Einhaltung wissenschaftlicher Standards, wozu vor allem Transparenz im Hinblick auf die Methodik der Datenerhebung und die öffentliche Diskussion der Ergebnisse gehören.“

— A. Bonn u. a. [1]

Bürgerinnen und Bürger können also einen Beitrag zur Wissenschaft leisten, indem sie Daten sammeln, analysieren und interpretieren. Dies kann von der einfachen Datenerhebung bis hin zur intensiven Auseinandersetzung mit einem Forschungsthema reichen. Die Beteiligung an Citizen Science Projekten ist nicht an eine akademische Ausbildung gebunden, jedoch ist die Einhaltung wissenschaftlicher Standards unabdingbar. Es stellt sich die Frage, worin die Motivation liegt, sich freiwillig solchen Projekten zu widmen. Ein treibender Faktor ist meistens der Wissenszuwachs für ein bestimmtes Thema. Die Auseinandersetzung in einem Bereich, für den man sich interessiert. Das Buch The Science of Citizen Science [2, p 283] diskutiert die verschiedenen Aspekte des Lernens in einem Citizen Science Projekt.

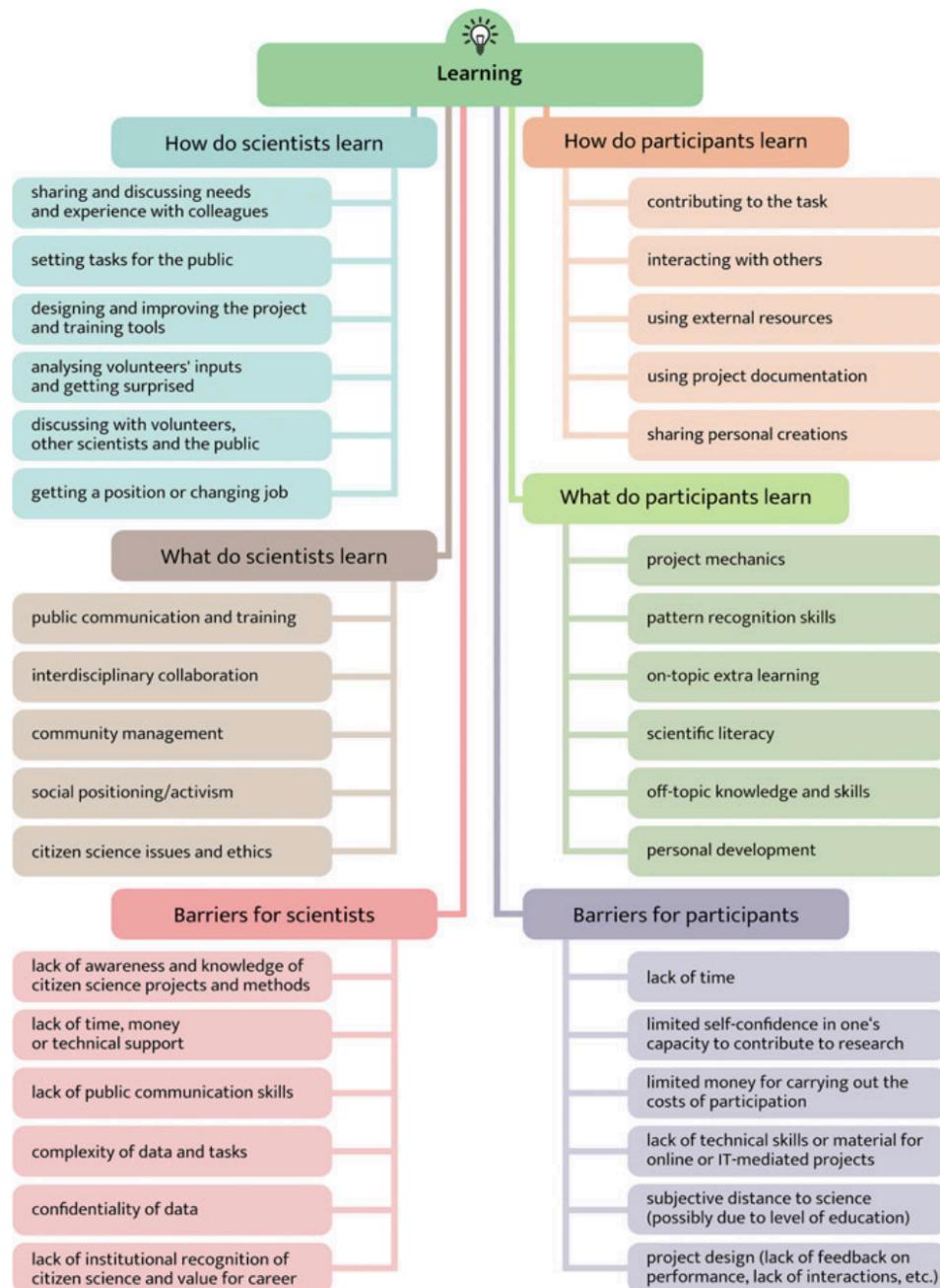


Abbildung 1: Erweiterte thematische Karte zum Lernen von Freiwilligen

(Quelle: The Science of Citizen Science, S.300)

Aus der Illustration geht hervor, dass verschiedenste Aspekte die Bürgerinnen und Bürger zu einer aktiven Beteiligung motivieren können. Neben den fachlichen sind auch soziale Themen von Bedeutung.

Auf der anderen Seite profitieren auch die Projektinitianten. Die Organisation und Zusammenarbeit mit Aussenstehenden kann eine spannende Aufgabe sein. Es stellt die Projektverantwort-

lichen vor neue Herausforderungen, die Freiwilligen motivierend in das Projekt miteinzubeziehen. Ein Grundstein dieser Angelegenheit ist die Zugänglichkeit zu Projektinstrumenten. In Bezug auf ein Projekt mit Edge-ML kann es besonders wichtig sein, dass die Freiwilligen verstehen, wie das Setup und die damit verbundene Technik funktionieren und somit Faszination entfachen können. Aus diesem Grund ist es von besonderer Bedeutung, die technischen Hürden möglichst niedrig zu halten und die notwendigen fachlichen Kenntnisse zu minimieren.

Citizen Science Projekte sind ein wichtiger Bestandteil der Forschung. In der Schweiz findet man beispielsweise auf der Seite schweizforscht.ch [3] eine Sammlung von aktiven Projekten. Besonders in der Pflanzen- und Tierwelt sind solche Projekte verbreitet, weil die Beobachtungen in der Natur häufig aufwändiger sind als in anderen Bereichen.

2.2 Machine Learning Grundlagen

2.2.1 Frameworks

Ein Machine-Learning-Framework, kurz ML-Framework, bildet die Grundlage für die Entwicklung, das Training und die Bereitstellung von Modellen. Es bietet Werkzeuge und Bibliotheken, die den gesamten ML-Workflow unterstützen. Im Gegensatz dazu ist ein Machine-Learning-Modell das konkrete Ergebnis des Trainingsprozesses, das spezifische Aufgaben wie Klassifikation oder Objekterkennung ausführt. Wird ein Modell nach dem Training benutzt, um beispielsweise Objekte zu erkennen, spricht man von Inferenz. Dies beschreibt den Prozess der Ausführung des Modells auf ungesehenen Daten.

Das Training sowie die Inferenz eines Modells können mit unterschiedlichen Frameworks vollzogen werden. Jedes Framework hat ihr eigenes Modellformat. So lassen sich beispielsweise PyTorch-Modelle nicht ohne Weiteres mit TensorFlow ausführen. Folgend sind diejenigen vorgestellt, welche in dieser Arbeit zum Einsatz kamen.

2.2.1.1 TensorFlow

TensorFlow [4] wurde ursprünglich von Google entwickelt und ist inzwischen ein Open Source Projekt. Das Framework ist eher für Systeme in Produktion gedacht.

- Vorteile: Weit verbreitet mit grosser Community und umfangreicher Dokumentation. Stellt viele Features zur Verfügung.
- Nachteile: Steile Lernkurve, insbesondere für Einsteiger komplex durch die vielen Features.

2.2.1.2 TensorFlow Lite

Stellt die für Edge-Devices optimierte Variante des TensorFlow-Frameworks dar.

- Vorteile: Speziell für den Betrieb auf ressourcenbeschränkten Geräten optimiert. Reduzierte Speicher- und Rechenanforderungen für Echtzeitanwendungen.
- Nachteile: Eine Kompression ist für die Ausführung erforderlich.

2.2.1.3 PyTorch

PyTorch [5] ist ein Open-Source Framework und hat seinen Ursprung im Forschungsteam für künstliche Intelligenz von Facebook. Das Framework eignet sich für Experimente, kann aber auch für Systeme in Produktion eingesetzt werden. ChatGPT von OpenAI arbeitet beispielsweise im Hintergrund mit dem Pytorch-Framework [6].

- Vorteile: Hat eine gute Community und eine ausführliche Dokumentation, welche viele Features und Tutorials beinhaltet.
- Nachteile: Steile Lernkurve, insbesondere für Einsteiger.

2.2.1.4 ONNX

ONNX (Open Neural Network Exchange) [7] ist ein generalisiertes Format von Deep-Learning Modellen. Dies ermöglicht den Austausch von Modellen zwischen verschiedenen Frameworks. ONNX wurde ursprünglich von Facebook und Microsoft entwickelt. Zurzeit wird das Framework von mehreren Partnern als Open-Source Projekt weiterentwickelt.

- Vorteile: Austauschformat, das Modelle zwischen verschiedenen Frameworks kompatibel macht. Optimierte Laufzeitumgebung, die speziell für Ausführungen auf der CPU geeignet sind.
- Nachteile: Begrenzte Funktionalität für Modelltraining, primär für den Einsatz trainierter Modelle gedacht. Kleinere Community im Vergleich zu TensorFlow und PyTorch.

Eine Spezialität des ONNX-Frameworks ist die Fähigkeit, Modelle von anderen Formaten in das ONNX-Format zu konvertieren. Ebenso lassen sich ONNX-Modelle wieder in andere Formate exportieren. Dies führt dazu, dass ONNX als eine Art Drehscheibe zwischen den verschiedenen Frameworks fungiert wie auf Abbildung 2 dargestellt [8].

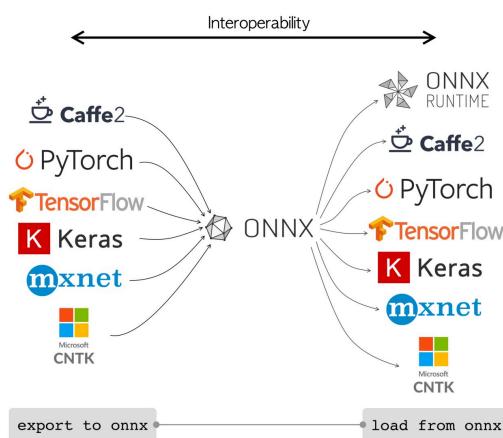


Abbildung 2: ONNX Framework als Drehscheibe
(Quelle: <https://docs.ultralytics.com/integrations/onnx/>, aufgerufen am 02.02.2025)

2.2.1.5 NCNN

NCNN [9] ist eine high performance computing platform für mobile Endgeräte. Dieses Framework wird oft auf Smartphones verwendet, weil es optimiert für Geräte mit beschränkter Rechenleistung ist.

- Vorteile: Schnelle Inferenzzeiten auf der CPU.
- Nachteile: Im Vergleich zu anderen Frameworks weniger ausführlich dokumentiert.

2.2.1.6 Hailo

Hailo, gegründet im Jahr 2017, hat sich als führendes Unternehmen im Bereich leistungsfähiger KI-Prozessoren für Edge-Anwendungen etabliert. Mit ihrer eigens entwickelten Hardware-Architektur verfolgt Hailo das Ziel, Machine Learning auch ausserhalb von leistungsstarken Systemen zugänglich und effizient nutzbar zu machen [10]. Die Nutzung der Hailo-Module erfordert eine Konvertierung der Modelle, wofür das Unternehmen eine umfassende Software-Suite bereitstellt. Diese Suite enthält alle notwendigen Werkzeuge, um Modelle auf die spezifischen Anforderungen der Hailo-Hardware abzustimmen.

- Vorteile: Ermöglicht sehr schnelle Inferenzen. Hailo bietet eine gute Dokumentation und ein grosses Ökosystem.
- Nachteile: Eine intensive Einarbeitung ist notwendig und der Konvertierungsprozess erfordert modellspezifisches Wissen.

2.2.1.7 Ultralytics

Ultralytics [11] ist ein auf PyTorch basierendes Open-Source-Framework, das vor allem für die Entwicklung von Modellen zur Objekterkennung bekannt ist. Es wurde durch die Implementierung von YOLOv5 (You Only Look Once) [12] populär und bietet eine benutzerfreundliche Umgebung für das Training und die Bereitstellung von Objekterkennungsmodellen. Inzwischen sind neuere Implementierungen der YOLO-Familie verfügbar und mit Ultralytics anwendbar.

- Vorteile: Die Verwendung des Frameworks ist sehr einfach und dadurch geeignet für Einsteiger. Modelle können in andere Formate exportiert werden. Inferenzen werden für PyTorch-Modelle angeboten.
- Nachteile: Die Flexibilität ist im Vergleich zu anderen Frameworks eingeschränkt. Zudem verwendet Ultralytics eine AGPL-3.0-Lizenz, was bei der Open-Source Community umstritten ist.

2.2.2 Anatomie von Machine Learning Modellen

Ein Machine-Learning-Modell besteht aus einer Architektur, die dessen Aufbau und Funktionsweise definiert, und einem Satz von Parametern, die während des Trainings optimiert werden. Die Architektur eines Modells legt grundlegende Eigenschaften fest, wie die Struktur der Neuronen und Layer, die Verbindungen zwischen ihnen sowie Eingabe- und Ausgabetensoren. Die Eingabetensoren bestimmen, welche Form und Dimensionen die Daten haben müssen (z. B. die Grösse von

Bildern), während die Ausgabetensoren das Ergebnis des Modells beschreiben, etwa Koordinaten für erkannte Objekte.

Es gibt verschiedene Typen von Machine-Learning-Modellen, die je nach Anwendungsfall eingesetzt werden. Zu den wichtigsten gehören Modelle für Objekterkennung, die Objekte in einem Bild lokalisieren und klassifizieren, Bildsegmentierung, die jedem Pixel eines Bildes eine Klasse zuordnet, und Posenschätzung, die die Körperhaltung von Personen oder Tieren analysiert. Die folgende Abbildung 3 zeigt eine Visualisierung der verschiedenen Disziplinen.

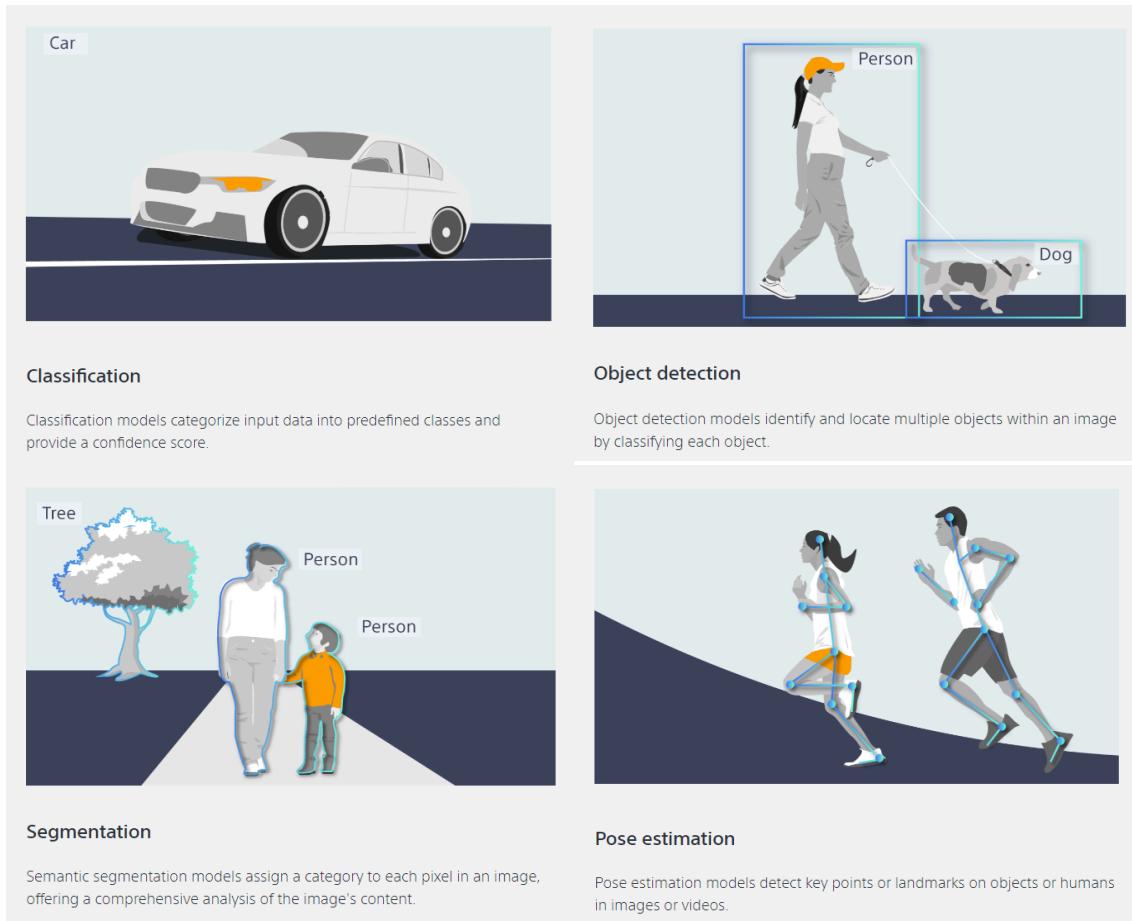


Abbildung 3: Disziplinen des Machine Learnings

(Quelle: https://github.com/sony/model_optimization, aufgerufen am 20.01.2025)

Diese Arbeit konzentriert sich auf die Objekterkennung, sodass alle folgenden Erwähnungen von Machine Learning in diesem Kontext zu verstehen sind. Für die Objekterkennung werden verschiedene Metriken verwendet, um die Modellleistung zu bewerten. Dazu zählen die Intersection over Union (IoU), die angibt, wie gut vorhergesagte und tatsächliche Detektionsfläche übereinstimmen, und die Mean Average Precision (mAP), die die Präzision über verschiedene Schwellenwerte

hinweg aggregiert. Zusätzlich spielen Leistungskennzahlen wie die Inferenzzeit eine wichtige Rolle, insbesondere bei Echtzeitanwendungen.

Das Training eines Modells umfasst die Anpassung der Parameter auf Basis eines grossen Datensatzes, um Muster und Zusammenhänge zu lernen. Diese Art des Trainings wird in der Fachsprache Supervised Learning genannt. Dabei kommen Optimierungsalgorithmen wie Gradient Descent zum Einsatz, die das Modell iterativ verbessern. Nach dem Training wird das Modell in der Inferenzphase genutzt.

Ein Modell hat definierte Input und Output Tensoren. Das heisst, dass die Daten für das Training und die Inferenz dem Input Tensor entsprechen müssen. Arbeitet das Modell mit Bildern beinhaltet dieser Schritt das modifizieren des Bildes auf die Input Grösse des Modells. Diesen Vorgang nennt man preprocessing. Ebenso variiert der Output Tensor je nach Modell und Framework. Die Output Daten im Falle von Object Detection beschreiben dabei den Ort des Objekts und die Confidence, also wie sicher sich das Modell ist, dass dieses Objekt zu einer bestimmten Klasse gehört. Je nach Datenformat müssen die Ergebnisse konvertiert und nach Bedarf der Anwendung unter Berücksichtigung des Preprocessing auf das Originalbild zurückgerechnet werden. Dieser Prozess wird als Postprocessing bezeichnet.

2.2.3 Geschwindigkeit einer Inferenz

Die Dauer einer Inferenz hängt massgeblich von den Berechnungen des Modells und der Leistungsfähigkeit des Systems ab, auf dem sie ausgeführt wird. Die benötigte Anzahl an Berechnungen wird durch die Neuronenanzahl im neuronalen Netz bestimmt. Mit zunehmender Anzahl an Neuronen – sei es durch mehr oder grössere Hidden Layers – steigt der Rechenaufwand. Die Abbildung 4 illustriert den symbolischen Aufbau eines neuronalen Netzes mit drei Hidden Layers. Ein Netz mit mehr als einem Hidden Layer wird als „Deep Neural Network“ bezeichnet.

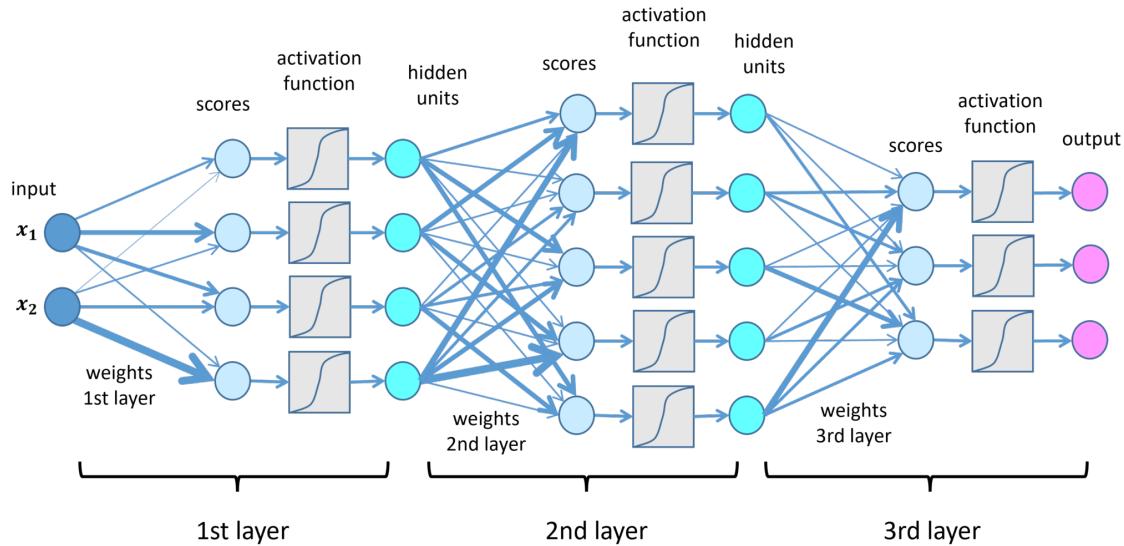


Abbildung 4: Symbolbild neuronales Netzwerk

(Quelle: lamarr-institute.org/blog/deep-neural-networks, abgerufen am 23.01.2025)

Je mehr Hidden Layers und trainierte Gewichte ein Modell besitzt, desto präziser werden seine Vorhersagen. Allerdings steigt damit auch der Rechenaufwand. Aus diesem Grund existieren Yolo-Modelle in verschiedenen Dimensionen, resp. Größen [12].

Ein weiterer entscheidender Faktor ist die Hardware, auf der die Berechnungen durchgeführt werden. Durch das Parallelisieren von Berechnungen lassen sich erhebliche Geschwindigkeitsvorteile erzielen. Besonders geeignet sind GPUs (Graphics Processing Unit), TPUs (Tensor Processing Unit) und NPUs (Neural Processing Unit). Diese Komponenten werden in Abschnitt 2.3.3 näher beschrieben.

Während für eine GPU grundsätzlich keine Vorbedingungen an das Modell gestellt werden, ist dies bei der TPU und NPU durchaus möglich. Dabei spielen zwei Strategien eine wesentliche Rolle: Quantisieren und Pruning [13]. Beim Quantisieren werden die Fließkommazahlen der Gewichte zu Ganzzahlen konvertiert, wodurch die Präzision abnimmt. Dies reduziert den Speicherbedarf und ermöglicht eine schnellere Ausführung, weil einfache Rechenoperationen ausgeführt werden können. Beim Pruning werden unwesentliche Verbindungen im Modellnetz entfernt, wodurch sich wiederum die Anzahl der Rechenoperationen reduziert. Die Ansätze werden auch kombiniert angewendet.

2.3 Edge Computing

Edge Computing und Edge Machine Learning markieren einen Paradigmenwechsel in der Systemarchitektur, indem die Datenverarbeitung von zentralen Backend-Servern oder Cloud-Systemen hin zu Geräten am Rand des Netzwerks (Edge) verlagert wird. Diese Architektur ermöglicht es,

Berechnungen und Analysen direkt vor Ort durchzuführen, anstatt die Daten für die Verarbeitung zu versenden.

Dies bringt mehrere Vorteile mit sich. So ermöglicht die lokale Verarbeitung eine geringere Latenz, was vor allem für Echtzeitanwendungen entscheidend ist. Zudem minimiert sich der Datenverkehr zu einem Backend, was nicht nur die Betriebskosten senkt, sondern auch die Change einer Abhängigkeit von einer Internetverbindung reduziert. Neben der Datenübertragung ist auch die Komplexität eines Systems von Bedeutung. Eine Systemarchitektur mit zentraler Einheit bedeutet nebst zusätzlichen Technologien auch einen erhöhten Wartungsaufwand, welcher gerade bei langer Projektdauer nicht zu unterschätzen ist. Ein weiterer wesentlicher Vorteil liegt im Bereich der Datensicherheit. Sensible Informationen bleiben vor Ort und müssen nicht über Netzwerke übertragen werden, wodurch die Privatsphäre der Nutzer besser geschützt wird.

Allerdings bringt dieser Ansatz auch Herausforderungen mit sich. Edge-Geräte verfügen häufig über eingeschränkte Rechenleistung und Speicherressourcen, was die Ausführung komplexer Machine-Learning-Modelle erschweren kann. Darüber hinaus ist die Energieeffizienz ein zentraler Aspekt, da viele Edge-Geräte batteriebetrieben sind und die Optimierung des Energieverbrauchs entscheidend für den Betrieb sein kann. Schliesslich kann die Verwaltung und Skalierung einer Vielzahl von verteilten Geräten ohne zentrale Steuerung zusätzlichen Aufwand erfordern.

2.3.1 Edge Intelligence

Edge Intelligence kombiniert die Prinzipien des Edge Computing mit den Anforderungen moderner Machine-Learning-Algorithmen, indem die Rechenleistung direkt auf die Endgeräte verlagert wird. Man spricht in diesem Falle auch von AI on Edge [14]. Diese Algorithmen werden entweder auf der CPU ausgeführt oder durch spezialisierte Hardware beschleunigt. Diese ist darauf ausgelegt, Machine Learning Tasks performant, effizient und ressourcenschonend auszuführen. Beschleuniger-Hardware ist dann erforderlich, wenn die Inferenz auf der CPU zu viel Zeit oder Ressourcen in Anspruch nimmt. Diese Hardware wird typischerweise mit einem Host-System verbunden, welches mit der Beschleuniger-Hardware kommunizieren kann.

Die Weiterentwicklung der Einplatinen-Computer sowie die auf ML spezialisierte Hardware öffnet neue Möglichkeiten im Bereich des Edge-Computing.

Zum Zeitpunkt dieser Arbeit sind die Entwicklungen im vollen Gange. Nicht nur auf der Seite der Hardware, sondern auch im Umfeld des Machine Learning wird zurzeit erforscht und entwickelt. Somit kann in diesem Projekt unter anderem neueste Hardware eingesetzt werden, welche sich in der Industrie möglicherweise erst in der kommenden Zeit etablieren wird. Die im Projekt eingesetzte Hardware ist im Folgenden aufgeführt.

2.3.2 Hostsysteme

Der einfachste Weg, Machine Learning im Bereich Edge zu betreiben, ist die CPU eines Edge-Computers. Um Erfahrungen zu sammeln, sind verschiedene Einplatinencomputer evaluiert worden. Hauptsächlich hat sich die Verwendung von Raspberry Pi Computer durchgesetzt. Sie weisen ein gutes Preis-Leistungs-Verhältnis auf und sind durch Dokumentation und Community extrem gut unterstützt.

2.3.2.1 Raspberry Pi 4

Das Raspberry Pi 4 8 GB ist für rund 85 Fr. [15] erhältlich. Mit einer CPU-Taktfrequenz von 1,5 GHz lassen sich schon viele Anwendungen realisieren. Nachteil des Modells 4 ist, dass keine PCI-Schnittstelle vorhanden ist.

2.3.2.2 Raspberry Pi 5

Das Raspberry Pi 5 stellt den Nachfolger vom Model 4 dar und ist mit 8 GB RAM und einer CPU-Taktfrequenz von 2,4 GHz ausgerüstet. Ebenso ist eine PCI-Schnittstelle verfügbar, wodurch das anschliessen von leistungsstarker Beschleuniger-Hardware möglich wird. Dieses Setup ist für ca. 86 Fr. [16] erhältlich.

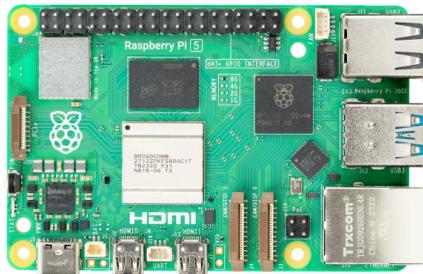


Abbildung 5: Raspberry Pi 5

(Quelle: <https://www.pi-shop.ch/raspberry-pi-5-16gb-ram?src=raspberrypi>, aufgerufen am 23.01.2025)

2.3.2.3 BeagleY-AI

Das BeagleY-AI Board hat eine CPU-Taktfrequenz von 1,5 GHz und 4 GB RAM. Das Besondere an diesem Board ist der integrierte AI-Accelerator mit 4 TOPS (Terra Operations per Second). Das Board ist für run 70 Fr. [17] erhältlich.

2.3.3 Beschleuniger

Machine-Learning-Beschleuniger sind spezialisierte Hardwarekomponenten, die entwickelt wurden, um die Ausführung von Machine-Learning-Tasks zu optimieren. Sie sind insbesondere für rechenintensive Aufgaben wie Matrixmultiplikationen und Tensorberechnungen ausgelegt, die in den ML-Algorithmen eine zentrale Rolle spielen.

Eine GPU wurde ursprünglich für die Verarbeitung von Grafikanwendungen entwickelt, hat sich jedoch aufgrund ihrer Fähigkeit zur parallelen Verarbeitung von Operationen gleichzeitig als

ideal für Machine-Learning-Aufgaben erwiesen. GPUs kommen häufig bei grossen Modellen und im Training von neuronalen Netzwerken zum Einsatz, da sie eine hohe Rechenleistung bieten.

Eine TPU [18] ist ein speziell entwickelter Prozessor, der ausschliesslich für Machine-Learning-Workloads optimiert wurde. Die TPU wurde von Google entwickelt und ist besonders effizient bei der Verarbeitung von Tensoroperationen, wie sie in Frameworks wie TensorFlow genutzt werden. TPUs sind für das Training und die Inferenz geeignet, wobei sie bei letzterem aufgrund ihrer geringen Latenz und Effizienz eine wichtigere Rolle spielen.

Die NPU [19] kann ML-Tasks hochgradig parallelisieren und effizient ausführen. In bestimmten Szenarien kann eine NPU eine GPU in Sachen Geschwindigkeit deutlich übertreffen [20].

2.3.3.1 Coral USB Accelerator

Der USB-Dongle von Google, Coral USB Accelerator, hat eine dedizierte TPU (Tensor Processing Unit). Solche Geräte lassen sich leicht in bestehende Systeme integrieren und ermöglichen die schnelle Ausführung von ML-Algorithmen ohne signifikante Anpassungen der Infrastruktur. Der Preis bewegt sich um 89,90Fr.[21].

2.3.3.2 Coral PCI-Beschleuniger

Nebst den USB-Dongles stellt Google auch über die PCI-Schnittstelle angeschlossene Beschleuniger her. Diese sind als Single oder Dual-TPU erhältlich. Der Preis für die Single-Lösung mit 4 TOPS liegt bei 47,90. [22], für die Dual-Lösung mit 8 TOPS bezahlt man 96,90 Fr. [23].



Abbildung 6: Coral PCI Accelerator

(Quelle: <https://www.coral.ai/products/m2-accelerator-dual-edgetpu>, aufgerufen am 22.01.2025)

2.3.3.3 Hailo-NPU

Die Hailo-Beschleunigermodule sind moderner als die Lösungen von Google und erreichen eine höhere Leistungsfähigkeit. Die in diesem Projekt verwendeten Beschleuniger haben 13 und 26 TOPS zu 79,90 Fr.[24] resp. zu 122,90 Fr.[25]. Die folgende Abbildung 7 zeigt einen Hailo-Beschleuniger auf einem Raspberry Pi.



Abbildung 7: Hailo Accelerator auf Raspberry Pi 5

(Quelle: <https://www.pi-shop.ch/raspberry-pi-ai-hat-13t>, aufgerufen am 22.01.2025)

2.3.3.4 AI-Kamera

Eine weitere Kategorie sind AI-Kameras, die mit integrierten ML-Beschleuniger ausgestattet sind. Diese Kameras, wie auf Abbildung 8 dargestellt, können nicht nur Bilder aufnehmen, sondern auch direkt auf der Kamera Inferenzresultate berechnen, beispielsweise durch die Erkennung und Klassifikation von Objekten. Raspberry Pi stellt eine solche Kamera zur Verfügung für 77,90 Fr. [26].



Abbildung 8: Raspberry Pi AI-Kamera

(Quelle: <https://www.berrybase.ch/raspberry-pi-ai-camera>, aufgerufen am 22.01.25)

Dieser Ansatz verlagert die intensiven Berechnungen auf die Kamera, womit der Edge-Computer weniger Ressourcen bereitstellen muss. Die Abbildung 9 verdeutlicht, wie die Architektur eines solchen Systems aufgebaut ist. Die linke Seite zeigt den Aufbau herkömmlicher Kamera-Architekturen, während rechts die Lösung mittels ML-Beschleuniger dargestellt ist.

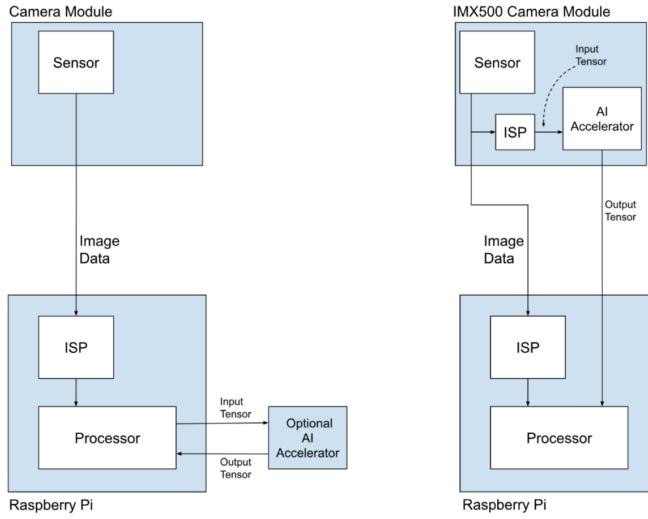


Abbildung 9: Raspberry Pi AI-Kamera Architektur

(Quelle: <https://www.raspberrypi.com/documentation/accessories/ai-camera.html>, aufgerufen am 20.01.2025)

Beim Sensor handelt es sich um einen Sony IMX500 Chip [27], welcher das Bild erfasst. Die Daten gelangen als RAW-Image zum ISP, einem kleinen Image Signal Prozessor. Dieser wandelt das RAW-Image zu einem Input-Tensor, welcher vom AI-Accelerator als Input-Grösse entgegengenommen wird. Nach Durchführen der Inferenz gelangen die Resultate, in diesem Fall der Output-Tensor, zum Prozessor des Host-Systems. Gleichzeitig gelangt auch das Bild auf das Host-System, um, wenn nötig, die Resultate anzuzeigen oder weiterverarbeitet zu werden. Die Bilder können wahlweise mit 10 FPS bei einer Auflösung von 4056×3040 oder bei 30 FPS mit 2028×1520 aufgenommen werden.

2.4 Mitwelten Biodiversitäts Monitoring

Biodiversitätsmonitoring befasst sich grundsätzlich mit dem Beobachten und Analysieren der Tier- und Pflanzenwelt.

„Die biologische Vielfalt bildet eine Lebensgrundlage der Schweiz. Deshalb ist es wichtig, ihren Zustand und ihre Entwicklung zu kennen.“
— bdm [28]

Aus diesem Grund ist es von grosser Bedeutung, die Umwelt zu beobachten und Veränderungen festzustellen. Um diese Arbeit zu vereinfachen, sind automatisierte Lösungen für ein Monitoring gefragt.

Ein System zur automatisierten Datenerfassung wurde im Rahmen des Projekts Mitwelten entwickelt [29], [30]. Hierfür wurden verschiedene Sensoren zu einem IoT-Toolkit kombiniert. Dieses

Toolkit ermöglicht es, Daten von dezentralen Systemen zu erfassen und an ein zentrales Backend weiterzuleiten.

Das IoT-Toolkit umfasst unter anderem eine Kamera, die in regelmässigen Abständen Fotos von Blüten aufnimmt, um darauf Bestäuber zu detektieren. Im Rahmen des Projekts wurden so insgesamt 1,5 Millionen Bilder generiert. Abbildung 10 zeigt eine im Feld aufgestellte IoT-Kamera.



Abbildung 10: IoT Kamera im Feld

(Quelle: mitwelten.ch, aufgerufen am 23.01.2025)

Die enorme Menge an Bilddaten lässt sich nicht manuell analysieren. Deshalb entwickelte das Projektteam eine Machine-Learning-Pipeline, die Bestäuber automatisch erkennt.

Die technische Architektur des Systems verbindet die Kamera mit einem leistungsstarken Backend. Die Kamera, bestehend aus einem Raspberry Pi und einer angeschlossenen Kameraeinheit, erfasst die Bilder und überträgt diese über einen Access Point an das Backend. Auf dem Backend-Server laufen die rechenintensiven Analysen, welche die empfangenen Bilder nach Bestäuber untersuchen.

2.4.1 Bestäuberanalyse

Die Bestäberdetektion erfolgt in den folgenden Schritten. Ein Foto von mehreren Blüten, zum Beispiel von einem Blumentopf, durchläuft in einem ersten Schritt eine Machine Learning Analyse zur Detektion von Blüten. Die auf dem Bild detektierten Blüten auf Abbildung 11 in Rot gekennzeichnet, werden anschliessend ausgeschnitten und somit zu einzelnen kleinen Bildern. Jedes dieser Bilder wird anschliessend mit einer weiteren Analyse und einem anderen Machine Learning Modell auf Bestäuber untersucht. Eine grosse Herausforderung in diesem Setup ist, dass die Anzahl Bestäberanalysen mit der Anzahl detekтирter Blüten steigt. Dies kann bei grossen Blütenzahlen zu langen Inferenzzeiten eines einzigen Bildes führen.



Abbildung 11: Vorgang Mitwelten Bestäuber Analyse

(Quelle: Automated Analysis for Urban Biodiversity Monitoring, Timeo Wullschleger)

2.4.2 Edge Architektur

Im Kontext dieser Entwicklung wurden verschiedene Architekturen untersucht [30, p 58]. Unter anderem ein System, auf dem die Machine-Learning-Pipeline auf einem Raspberry Pi 3 ausgeführt wird. Der Edge-Computer analysiert die Bilder vor Ort und schickt nur die Resultate an das Backend. Die Problematik mit der steigenden Anzahl an Inferenzen pro detektiertener Blüte kommt aber in diesem Setup besonders zur Geltung. Die Abbildung 12 zeigt das Verhältnis der detektierten Blüten zur Analysezeit auf.

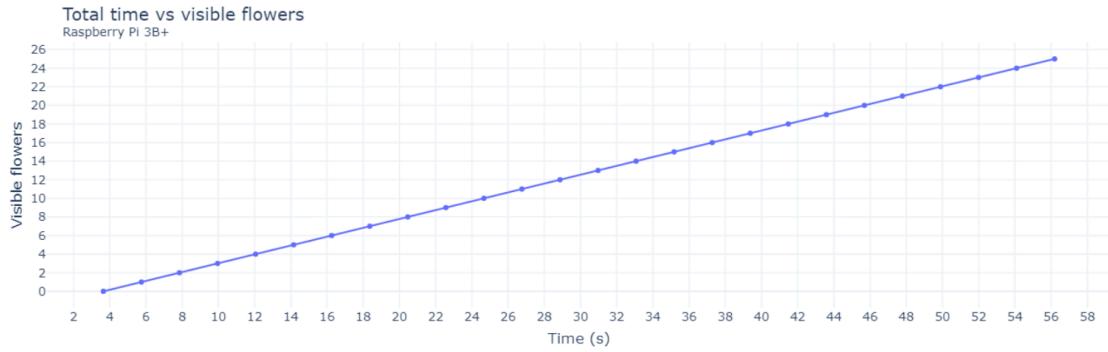


Abbildung 12: Mitwelten Analyse auf Raspberry Pi 3

(Quelle: Automated Analysis for Urban Biodiversity Monitoring S.61, Timeo Wullschleger)

Mit wachsender Anzahl Blüten auf dem Bild steigt die Zeit für die Bestäuberanalyse stark an. Da eine Kamera typischerweise immer dieselben Blüten untersucht, würde eine Perspektive mit vielen Blüten bei gleichbleibendem Bildintervall von 15 Sekunden, immer mehr in Verzögerung geraten. Selbst wenn die Aufnahmepausen aufgrund der Lichtverhältnisse zur Analyse genutzt würden, wäre diese Architektur mit diesem Setup nicht umsetzbar.

3 Analyse

Dieses Kapitel befasst sich mit der Analyse und Bewertung der Möglichkeiten für die Umsetzung einer Edge ML Kamera. Zuerst werden die Randbedingungen für das System eruiert, in dem die Zielgruppe und die Anforderungen erarbeitet werden. Das Kapitel Evaluation stellt anschliessend verschiedene Resultate zu Untersuchungen von ML Frameworks vor. Zum Abschluss dieses Kapitels werden verschiedene Möglichkeiten von Pipeline Setups vorgestellt.

3.1 Zielgruppe

Ein Edge ML Kamera, welche flexibel für verschiedene Zwecke einsetzbar ist, deckt dementsprechend eine breite Zielgruppe ab. Anwendungen könnten für die folgenden Interessengruppen von Bedeutung sein:

- Forschende
- Citizen Scientists
- Naturschutzorganisationen
- Bildungseinrichtungen
- Landwirtschaftliche Betriebe

3.2 Use Cases

Anhand der Zielgruppen gibt es verschiedene Szenarien, in dem der Einsatz einer Edge ML Kamera Anwendung findet. Die folgenden Szenarien wurden identifiziert:

- Biodiversitätsforschung: Überwachung von Bestäubern, Vögeln, Säugetieren oder Pflanzenwachstum.
- Artenschutz: Identifikation und Überwachung gefährdeter Tier- oder Pflanzenarten.
- Umweltüberwachung: Aufzeichnung und Analyse des Einflusses von Umweltbedingungen wie Temperatur, Feuchtigkeit oder Lichtverhältnissen auf die Pflanzen oder Tierwelt.
- Landwirtschaft: Erkennung von Schädlingen, Optimierung des Pflanzenwachstums.

3.2.1 Referenz Szenario

Die Mitwelten Bestäuber Analyse dient während der Entwicklung der Edge ML Kamera als Referenz-Szenario. Um das System zielgerichtet voranzutreiben, wurden Tests und Evaluationen mittels dieser Analyse bewertet. Dies bietet sich an, weil aus dem Mitwelten Projekt bereits Machine Learning Modelle bestehen und dementsprechend auch Daten, um die Modelle zu trainieren und zu testen.

3.3 Anforderungen

Wie in den Kapiteln zuvor diskutiert, gibt es eine breite Anwendung für eine Edge ML Kamera. Als Hilfestellung für die Identifikation der Anforderungen diente als Orientierung die Mitwelten Bestäuber Analyse.

3.3.1 Funktionale

Die folgenden funktionalen Anforderungen an eine Edge ML Kamera wurden identifiziert:

- Datenerfassung: Das System muss anhand einer angeschlossenen Kameraeinheit Bilder kontinuierlich erfassen können.
- Analyse Resultate: Das System muss die Analysedaten für den jeweiligen Anwendungszweck zur Verfügung stellen.
- Benutzerinteraktion: Das System muss vom Benutzer konfiguriert werden können.
- Energieversorgung: Während des Betriebs muss eine stabile Energieversorgung vorhanden sein.

3.3.2 Nicht Funktionale

Die folgenden nicht funktionalen Anforderungen an eine Edge ML Kamera wurden identifiziert:

- Analyse-Intervall: Aus dem Vorprojekt [30, p 62] geht hervor, dass die Bestäuberanalyse in einem Intervall von 15 Sekunden erfolgen sollte, um die Detektions-Change zu maximieren.
- Betrieb: Das System muss zuverlässig im Dauerbetrieb funktionieren.
- Kosten: Der Preis für das Gesamtsystem muss so tief sein, dass Citizen Science Projekte realistisch sind.
- Adaption: Die Bediener des Systems müssen die Edge ML Kamera in erster Linie für ihre Zwecke und Bedürfnisse anpassen können. Aus diesem Grund ist es wichtig, das System so einfach wie möglich zu halten. Da eine spezifische Analyse von Bildern mittels Machine Learning nicht generalisiert werden kann, müssen die Benutzer ihre eigene Implementierung in das System integrieren können. Ebenfalls muss die Erfassung von Bildern und die Verarbeitung der Analyseergebnisse flexibel gestaltet werden können. Beide Aspekte erfordern einen Eingriff in das System. Um diese Vorgänge zu vereinfachen, müssen diese anhand von Anleitungen und Beispielen unterstützt werden.

3.4 Evaluation

Im Folgenden werden Resultate der Untersuchungen bezüglich ML Frameworks und Hardware aufgezeigt. Dabei sind Inferenzen mit PyTorch, Tensorflow lite, ONNX, NCNN und Hailo Modellen implementiert worden. Die Inferenzen wurden auf Raspberry Pi's mit und ohne Beschleuniger Hardware ausgeführt und gemessen. Um die Resultate miteinander zu vergleichen und anhand von Grafiken zu untersuchen, wurde ein Dashboard entwickelt [31]. Das Dashboard basiert auf CSV-Daten, welche Test-Skripts zur Messung von Inferenzzeiten automatisch generieren. Für jedes Framework existiert ein separates Skript, welches ein Framework auf einer spezifischen Hardware mit variabler Anzahl Bilder ausführt. Die folgende Abbildung 13 zeigt den Startscreen der Applikation.

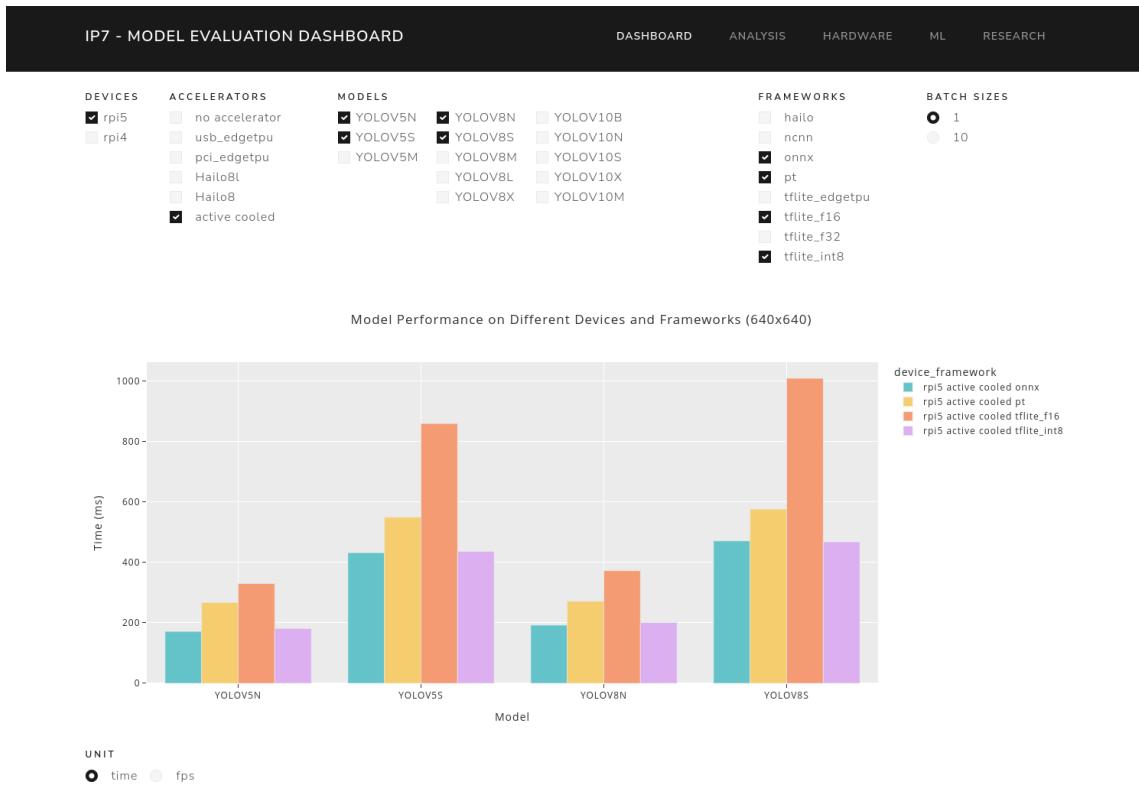


Abbildung 13: Model Evaluation Dashboard

(Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

Im Repository [32] befinden sich nebst dem Dashboard auch die Skripts zur Messung der Inferenzzeiten. Je nach Test muss die entsprechende Hardware an das System angeschlossen sein. Aufgrund Schwierigkeiten mit dem BeagleY-AI wurden keine Resultate dieses Boards zur Evaluation beigezogenen.

3.4.1 Methodik

Bei allen verwendeten Modellen handelt es sich um Yolo-Modelle verschiedener Generationen und Grösse. Die Modelle zur Bestäuberanalyse aus Abschnitt 2.4.1 sind ebenfalls Yolo-Modelle der Generation 5. Inzwischen sind die Modelle weiterentwickelt worden. Während dieser Arbeit wurde die 11. Generation veröffentlicht. Verglichen wurden die Generationen 5 und 8 um festzustellen, wie sich die Metriken über die Generationen verhalten. Die Modelle stammen von Ultralytics [33] und haben eine Input-Grösse von 640x640 Pixel. Die Inferenzen wurden jeweils auf dem COCO: Common Object in Context [34] Datensatz durchgeführt. Dies ist der state-of-the-art-Bilderdatensatz, um einheitliche Metriken für Machine Learning Tasks zu generieren. Der Fokus in den folgenden Analysen liegt auf den Inferenzzeiten, wobei die Genauigkeit eine sekundäre Rolle spielt. Die verwendeten Yolo-Modelle haben publizierte Metriken auf dem COCO-Datenset [35].

3.4.2 ML Framework

3.4.2.1 Vergleich Inferenzzeiten

Die folgende Abbildung 14 zeigt auf, wie sich die Inferenzzeiten der jeweiligen ML Frameworks in Bezug auf die YOLO-Generation verhalten. Ersichtlich ist, dass hauptsächlich die Inferenzzeit der PyTorch Inferenz mit der neuen Generation signifikant gestiegen ist. Die anderen Frameworks weisen nur geringe Unterschiede auf, wobei die Tendenz des Anstieges bei jeder Kategorie feststellbar ist. Ein weiteres Merkmal ist, dass die Inferenz mit dem ONNX-Modell bei allen Versuchen am wenigsten Zeit benötigt.

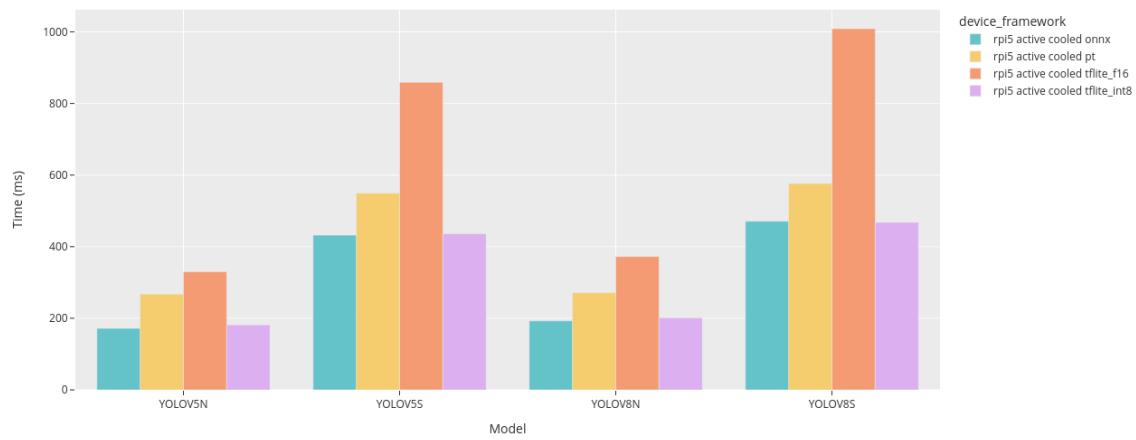


Abbildung 14: Vergleich: Inferenzzeit YOLOv5, v8 auf Raspberry Pi 5

In einem nächsten Schritt wird ONNX mit NCNN verglichen. Wie in Abschnitt 2.2.1.5 erwähnt, ist NCNN für Geräte mit begrenzter Rechenleistung und der Ausführung auf der CPU optimiert. Dies zeigt sich auch in der folgenden Abbildung Abbildung 15. Die Inferenzzeiten halbieren sich nochmals gegenüber der Inferenz mit ONNX.

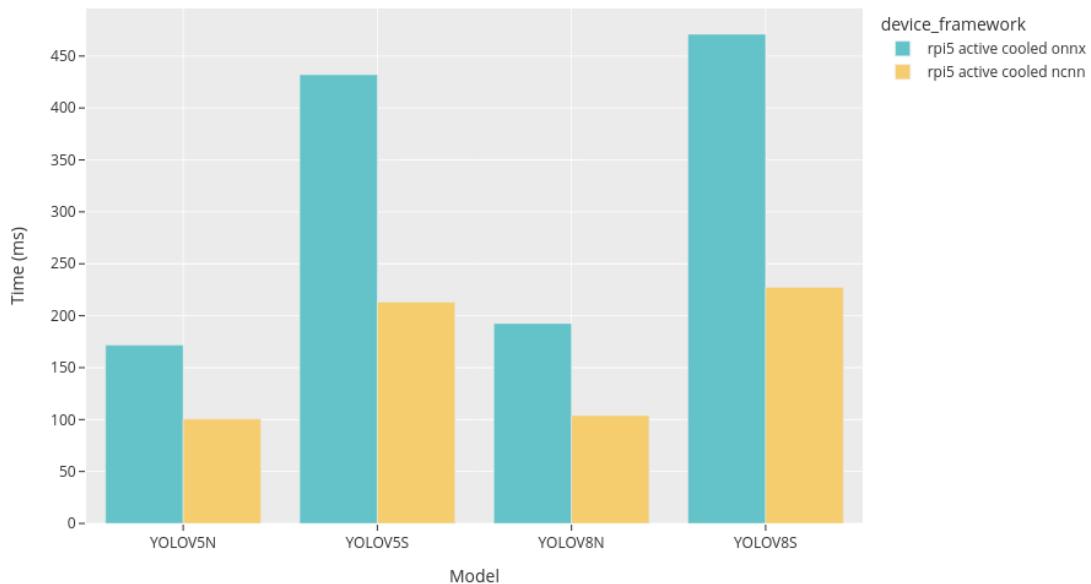


Abbildung 15: Vergleich: ONNX und NCNN auf Raspberry Pi 5

Aus diesen Analysen geht hervor, dass mit NCNN auf einer Raspberry Pi 5 CPU die schnellsten Ergebnisse erzielt werden können. Um nun die Inferenzzeiten weiter zu beschleunigen, wird zusätzliche Hardware benötigt.

3.4.3 Hardware

Im Folgenden werden verschiedene Hardware Setups miteinander verglichen. Teilweise beansprucht spezifische Hardware auch definierte ML-Frameworks. Somit ist der Vergleich von verschiedener Hardware nicht mit gleichen Frameworks möglich. Dennoch können die Inferenzzeiten verglichen werden.

3.4.3.1 Raspberry Pi Vergleiche

Der erste Vergleich zeigt den Fortschritt der Raspberry Pi Generationen auf. Die Inferenzzeiten eines ONNX-Modells sind auf einem Raspberry Pi 5 in grün mehr als doppelt so schnell.

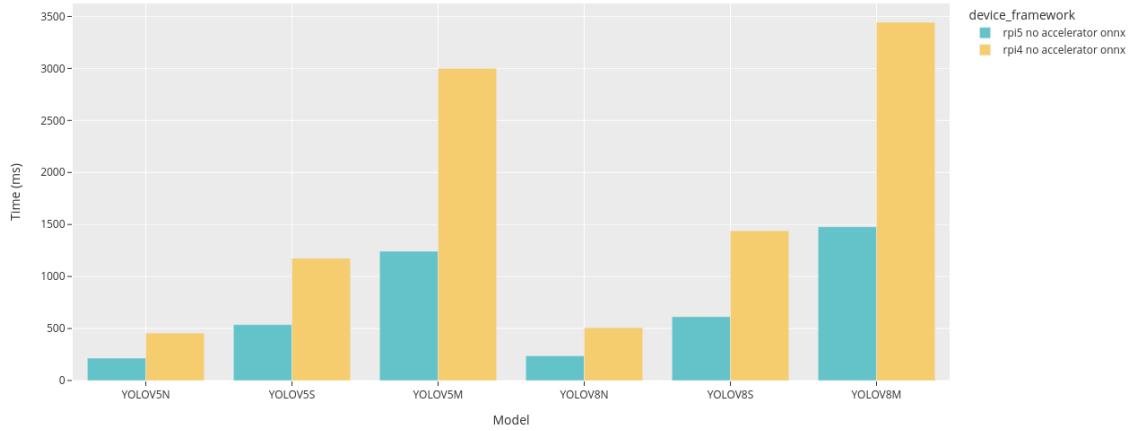


Abbildung 16: Vergleich: Raspberry Pi 4 vs. Pi 5
 (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

Ein weiterer wichtiger Punkt ist die Kühlung des Systems, ansonsten wird die CPU-Leistung des Rechners gedrosselt. In der folgenden Abbildung 17 ist ersichtlich, dass die Inferenzzeiten mit Kühlung rund 20 % schneller gegenüber der Ausführung ohne sind.

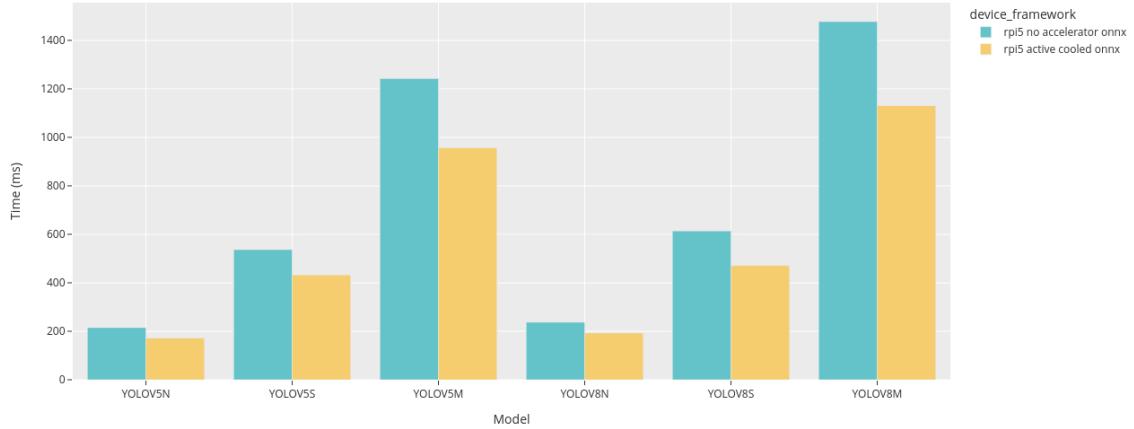


Abbildung 17: Vergleich: Raspberry Pi 5 mit und ohne Kühlung
 (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

3.4.3.2 Coral Accelerator

Das Toolkit Coral [36] wurde von Google entwickelt und stellt Beschleuniger verschiedener Art zur Verfügung. Im Kontext dieser Arbeit wurden zwei dieser Beschleuniger untersucht: ein USB-Dongle mit einer Edge-TPU von 4 TOPS und ein über PCI verbundenes Board mit einer TPU mit 8 TOPS.

Bei den Versuchen hat sich herausgestellt, dass der USB-Dongle unzuverlässig ist. Es kam während länger laufender Tests wiederholt zu Verbindungsunterbrechungen. Die Edge-TPU über PCI funktionierte zuverlässig.

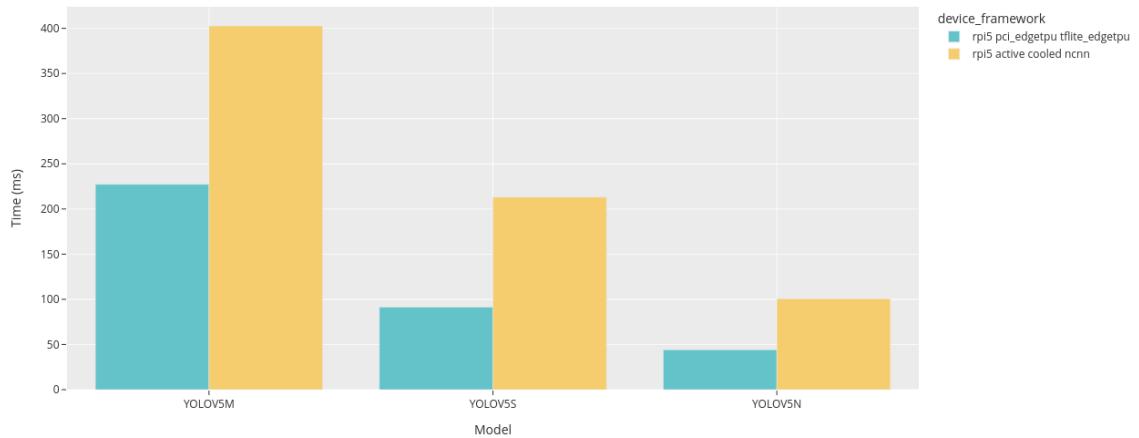


Abbildung 18: Vergleich: Raspberry Pi 5 vs. Coral PCI Edge TPU

Es ist sofort ersichtlich, dass die Inferenz auf der Edge-TPU (4 TOPS) um ein Vielfaches schneller ist als auf der CPU mit NCNN des Raspberry Pi 5. Allerdings soll hier erwähnt sein, dass die EdgeTPU nur quantisierte TensorFlow-Lite-Modelle ausführen kann und somit auch ein Verlust der Präzision gegenüber dem NCNN-Modell. Der genaue Verlust wurde an dieser Stelle nicht eruiert, da Anwendungen auf dieser Hardware-Komponente im Anschluss nicht weiterverfolgt wurden.

3.4.3.3 Hailo

In dieser Untersuchung wurden die beiden Module Hailo8l (13 TOPS) und Hailo8 (26 TOPS) eingesetzt. Die beiden Module werden jeweils über die PCI-Schnittstelle mit dem Raspberry Pi 5 verbunden. Aufgrund der benötigten PCI-Schnittstelle kann das Raspberry Pi 4 Modell nicht verwendet werden.

Für die Messungen der Inferenzzeit ist das von Hailo zur Verfügung gestellte ML-Framework verwendet worden. Zudem müssen die Modelle im Hailo-eigenen Format .hef konvertiert sein, um sie auf dem Beschleuniger auszuführen. Hailo stellt einen Model Zoo [37] zur Verfügung, welcher die Modelle YOLOv5s und YOLOv5m beinhaltet.

Die folgende Abbildung 19 zeigt den Vergleich mit der Coral Edge TPU (4 TOPS).

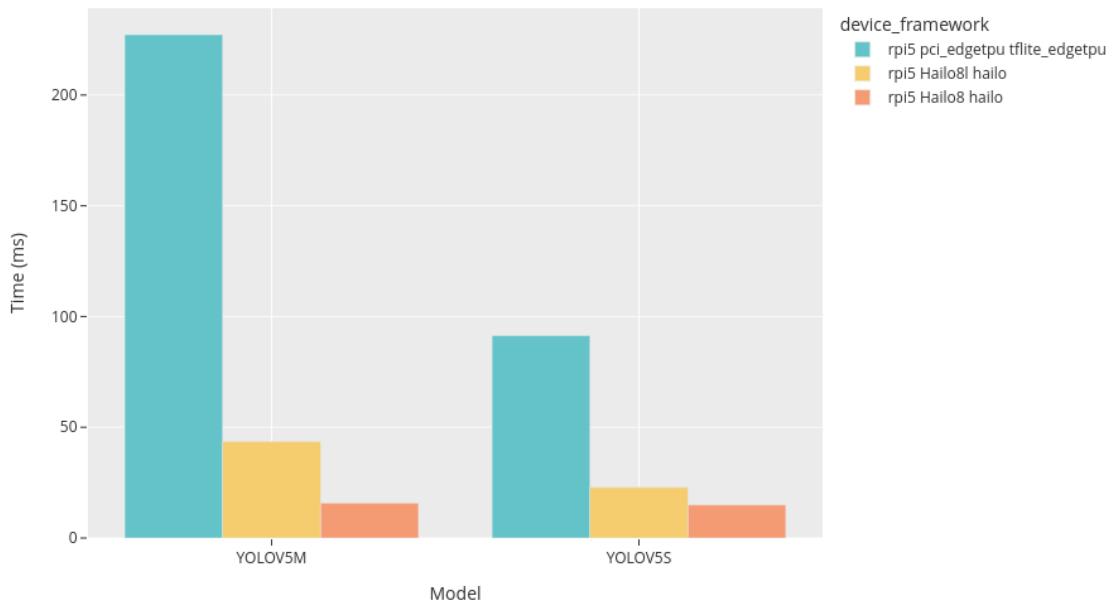


Abbildung 19: Vergleich Hailo Accelerator vs. Coral PCI Edge TPU

(Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

Die beiden Hailo-Beschleuniger sind signifikant schneller als der schon etwas ältere Coral-Beschleuniger. Ebenso ist ersichtlich, dass sich die doppelte Anzahl TOPS direkt auf die Inferenzgeschwindigkeit auswirkt. Die schnellste Inferenz wird somit mit dem Hailo8-Beschleuniger erzielt und beträgt rund 15 ms.

3.5 Edge ML Setups

Aus der Selektion der Hardware und den in diesem Kapitel aufgeführten Evaluationen von Inferenzen lassen sich nun verschiedene Konstellationen mit unterschiedlichen Stärken, resp. Schwächen definieren. Die Abbildung 20 zeigt das Verhältnis von Kosten zu Inferenzgeschwindigkeit der viel-versprechendsten Kombinationen mit dem YOLOv8n-Modell. Die Setups sind jeweils ohne Kamera kalkuliert, was beim Vergleich mit der AI-Kamera berücksichtigt werden muss.

Raspberry Pi 5 - Mitwelten ML Pipeline

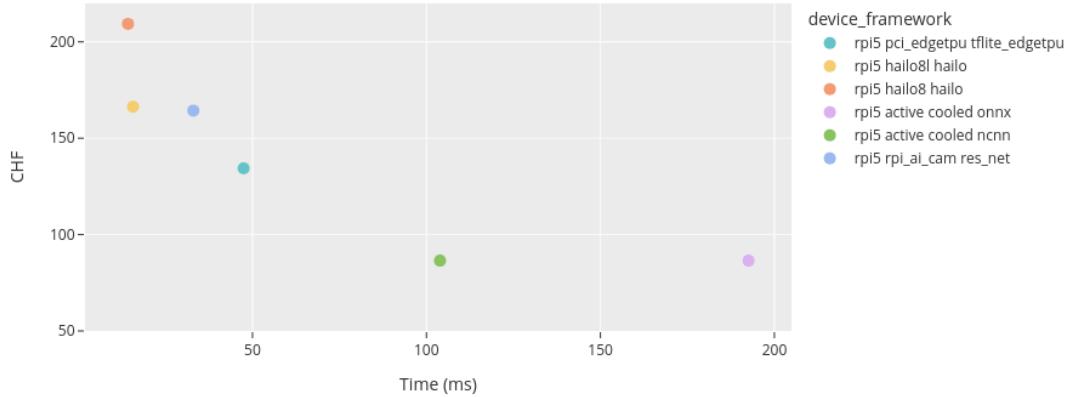


Abbildung 20: Vergleich: Kosten vs. Geschwindigkeit mit YOLOv8n

(Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)

Weil das Raspberry 4 kaum billiger ist als sein Nachfolger Pi 5, ist das Modell 4 für diese Bewertung auszuschliessen.

Die mit Abstand kostengünstigsten Varianten sind jene ohne zusätzliche Beschleuniger-Hardware. Dabei schneidet das NCNN-Framework auf dem Raspberry Pi 5 mit ca. 9,6 FPS am besten ab, gefolgt von ONNX mit rund 5,2 FPS.

Bei den Setups mit Beschleuniger sehen wir, dass mit höheren Ausgaben entsprechend mehr FPS zu erzielen sind. Die schnellsten Inferenzen liefert der teuerste Beschleuniger Hailo8 mit 26 TOPS von Hailo. Der kleinere Hailo-Beschleuniger befindet sich auf der Stufe der AI-Camera von Raspberry Pi. Etwas abgeschlagen, dafür auch kostengünstiger, ist der M.2 PCI Edge-TPU von Coral.

Um auf dem Raspberry Pi die beste Performance zu erzielen, empfiehlt sich ein Chip von Hailo. Eine Edge-TPU von Google ist aufgrund des abnehmenden Supports und der weniger ausführlichen Dokumentation nicht empfehlenswert.

Lässt es die Applikation zu, ist auch die AI-Kamera von Raspberry Pi eine gute Wahl. Dabei muss berücksichtigt werden, dass in Systemen mit mehr als einer Inferenz nur die erste auf der Kamera durchgeführt werden kann. Darauffolgende Inferenzen müssten auf der CPU oder auf einem weiteren Beschleuniger ausgeführt werden. Ein Setup mit einem weiteren Beschleuniger würde natürlich auch die Kosten deutlich erhöhen.

4 Umsetzung

Dieses Kapitel beschreibt die Umsetzung einer Software zum Betreiben einer Edge ML Kamera. Das System umfasst die Erfassung von Bildern, Ausführen einer Operation jeglicher Art, die aus einem Bild Resultate generiert, und die Weiterverarbeitung dieser Resultate. Die Entwicklung erfolgte in drei Iterationen, bei welchen jeweils die erstellte Architektur reflektiert und anhand der gewünschten Qualitäten angepasst wurde. Zuerst wird die Systemarchitektur vorgestellt, gefolgt von den wichtigsten Aspekten des Softwaredesigns. Anschliessend folgen verschiedene Prototypen zur Umsetzung konkreter Beispiele.

4.1 Systemarchitektur

Damit das System für Citizen Science geeignet ist, muss es self-contained und flexibel an neue Szenarien anpassbar sein. Dafür sind drei modular austauschbare Komponenten entscheidend: die Bildquelle mit Buffer, die Bildanalyse durch eine Operation und die Verarbeitung der Analysedaten. Besonders die Bildanalyse, die meist eine Machine-Learning-Inferenz umfasst, erfordert individuelle Anpassungen. Da Modelle spezifisches Pre- und Postprocessing benötigen, lässt sich keine allgemeine Implementierung für verschiedene ML-Frameworks realisieren.

4.1.1 Komponentendiagramm

Die folgende Abbildung 21 zeigt eine Übersicht des Systems. Eine **Source** repräsentiert die Quelle eines Bildes, wie zum Beispiel eine Kamera. Als Buffer zwischen der Analyse und der Source dient eine **Queue**. Bei der **Operation** handelt es sich um die Komponente der Analyse. Dies kann eine Machine Learning Inferenz sein oder eine andere Operation, die ein Bild verarbeitet und ein Resultat generiert. Anschliessend gelangt der Output der Operation in eine oder mehrere **Sinks**. Die Sink verarbeitet die Daten weiter, um diese beispielsweise per HTTP-Request zu versenden oder in einer Datenbank zu speichern.

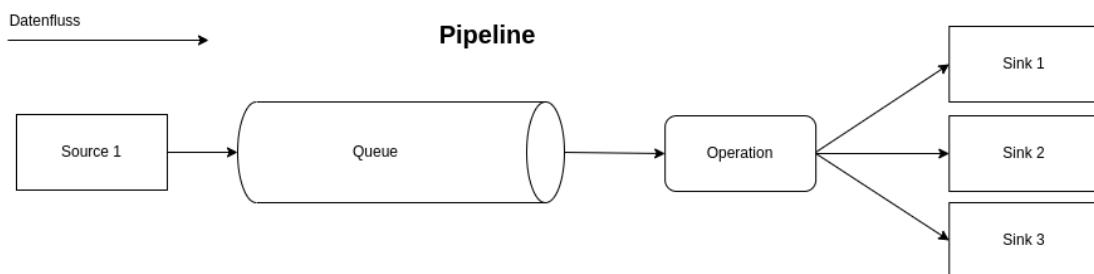


Abbildung 21: Pipeline Komponenten
(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

Grundsätzlich ist das System für Biodiversitätsmonitoring ausgelegt, weshalb die Zeit zur Erfassung von Bildern durch die Lichtverhältnisse limitiert ist. Somit ist für die Analyse potentiell mehr Zeit verfügbar als für die Aufnahme der Bilder. Defaultmäßig wird eine In-Memory-Queue

verwendet. Es ist aber durchaus möglich, dass beispielsweise eine Queue mit Zugriff auf eine externe Festplatte verwendet wird. Diese Implementierung lässt sich durch Einhalten der Queue-Schnittstelle leicht auf projektspezifische Anforderungen anpassen.

4.1.2 Qualitäten

Da die Software für verschiedenste Anwendungen zum Einsatz kommen kann und dafür jeweils auch angepasst oder erweitert werden muss, ist das Hauptaugenmerk auf die folgenden Qualitätskriterien gelegt worden:

- Einfachheit: Der Code ist in Python geschrieben, einer Programmiersprache, die bei Forschenden einen hohen Bekanntheitsgrad geniesst. Die Struktur und Implementierungen sind jeweils einfach gehalten, so dass sie beim Lesen des Codes leicht verständlich und nachvollziehbar sind.
- Modularisierung: Alle Komponenten, die austauschbar sind, sind durch abstrakte Klassen vorgegeben. Interfaces, wie man sie aus anderen Sprachen kennt, stehen in Python nicht zur Verfügung.
- Dokumentation: Die relevanten Code Abschnitte sind mit Kommentaren unterstützend erklärt.
- Typisierung: Die wichtigsten Klassen und Methoden sind mit definierten Datentypen definiert, wodurch die Lesbarkeit und das Verständnis des Codes verbessert wird.
- Logging: Die Applikation hat ein Logging, das je nach Bedarf auf verschiedenen Levels aktiviert werden kann, um Debugging und Einarbeitung zu erleichtern.

4.1.3 Konfiguration

Das System lässt sich mittels Konfigurationsfile definieren. Die Konfiguration enthält im Wesentlichen drei Komponenten, einen für jeden auswechselbaren Teil der Software. Somit lassen sich mehrere Sources, Operations und Sinks definieren. Diese Einzelteile können dann zur Laufzeit mittels Webinterface angepasst werden. Dieser Mechanismus erleichtert das angenehme Testen und Anpassen einer Pipeline.

Das folgende Listing 1 zeigt eine Beispielkonfiguration mit jeweils einer Option für jeden Komponententyp.

```

1   sources:
2     - name: webcam
3       file_path: ./source/impl/webcam.py
4       class_name: Webcam
5       parameters:
6         device: "/dev/video0"
7         width: 640
8         height: 640
9       operations:
10      - name: detect coco objects
11        class_name: UlDetect
12        file_path: ./pipe/impl/ulDetect.py
13        parameters:
14          model_path: ./resources/ml_models/yolo1In.onnx
15          label_path: ./resources/labels/coco.txt
16          confidence_threshold: 0.5
17          nms_threshold: 0.5
18      sinks:
19        - name: console
20        class_name: Console
21        file_path: ./sink/impl/console.py

```

Listing 1: Beispiel Pipeline Konfiguration

Jede Komponente definiert die folgenden Attribute:

- **name**: Der Name der Komponente, damit man ihn später über das Konfigurationsinterface identifizieren kann.
- **file_path**: Der Pfad zur Datei, in der die zu ladende Klasse implementiert ist.
- **class_name**: Die zu ladende Klasse aus dem definierten File. In Python können mehrere Klassen in einer Datei implementiert sein.
- **parameters** (optional): Parameter, die der Klasse beim Initialisieren übergeben werden. Weil diese Parameter sehr anwendungsspezifisch sind, ist die Struktur dieses Dictionaries offen gelassen.

Die Applikation lädt alle Komponenten beim Start mithilfe eines **ClassLoader** Mechanismus und speichert deren Instanzen intern. Durch die Namen können die Komponenten zur Laufzeit eingeschaltet oder ausgeschaltet werden. Ein einfaches Anwendungsbeispiel ist das Umschalten zwischen einem Kamerastream und einem Testbild.

4.1.4 Webinterface

Die Applikation startet nebst den genannten Komponenten auch ein Webinterface zur Konfiguration des laufenden Systems. Komponenten lassen sich über die Dropdown-Menüs selektieren.

Die Abbildung 22 zeigt ein Screenshot des Interfaces.

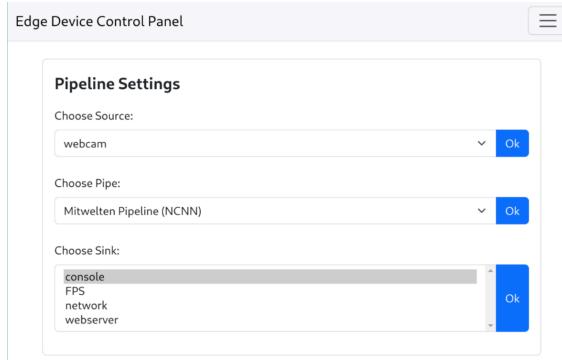


Abbildung 22: Webinterface Edge ML Kamera
(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

Das Webinterface wird standardmässig auf Port 8001 ausgeliefert.

4.2 Software Design

In diesem Abschnitt wird aufgezeigt, wie die einzelnen Komponenten miteinander kommunizieren.

Zu diesem Zweck wird folgend ein Klassendiagramm mit den relevanten Elementen dargestellt. Die zentrale Komponente stellt die Klasse Pipeline dar. Diese hat eine Beziehung zu einer Source, Operation und Sink. Die Pipeline orchestriert den Datenfluss zwischen den einzelnen Komponenten. Mittels Setter-Methoden auf der Pipeline lassen sich die austauschbaren Komponenten über das Webinterface konfigurieren.

Während der Entwicklung wurde das Design kontinuierlich angepasst um sicherzustellen, dass Abhängigkeiten reduziert werden und somit die Einfachheit gefördert wird. Gleichzeitig wurde versucht, sich auf die Kernfunktionalitäten des Systems zu beschränken und Features mit wenig Funktionalität zu beseitigen.

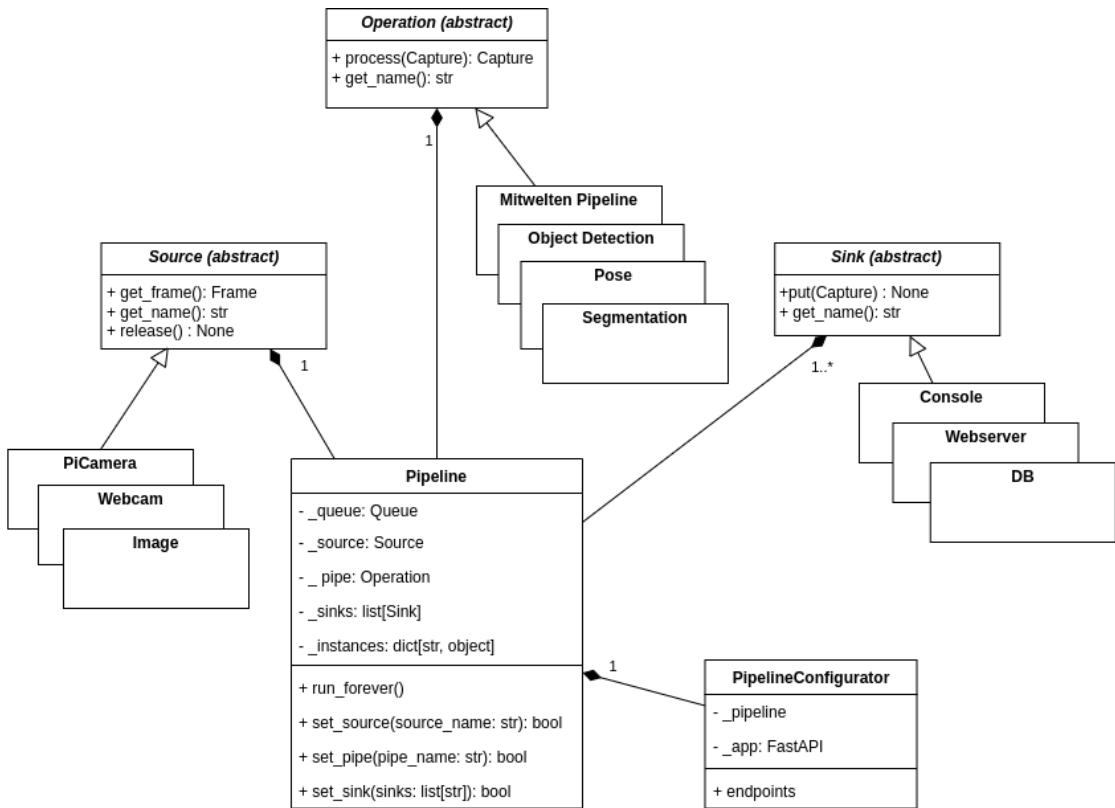


Abbildung 23: Klassen Diagramm

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

Um die zeitlichen Abläufe darzustellen, ist folgend ein vereinfachtes Sequenzdiagramm dargestellt. Die Pipeline ist dafür verantwortlich, dass ein Thread zur Erfassung von Bildern gestartet wird. Diese Bilder werden in einer Thread-Safe-Queue zwischengespeichert. Im Haupt-Thread der Applikation holt die Pipeline ein Bild von der Queue ab und führt die Operation aus. Anschliessend übergibt die Pipeline die Resultate einer oder mehrerer Sinks. Es kann mehrere Sinks geben, weil die Möglichkeit bestehen soll, Resultate abzuspeichern und gleichzeitig auf der Konsole oder auf einem Webinterface visuell darzustellen.

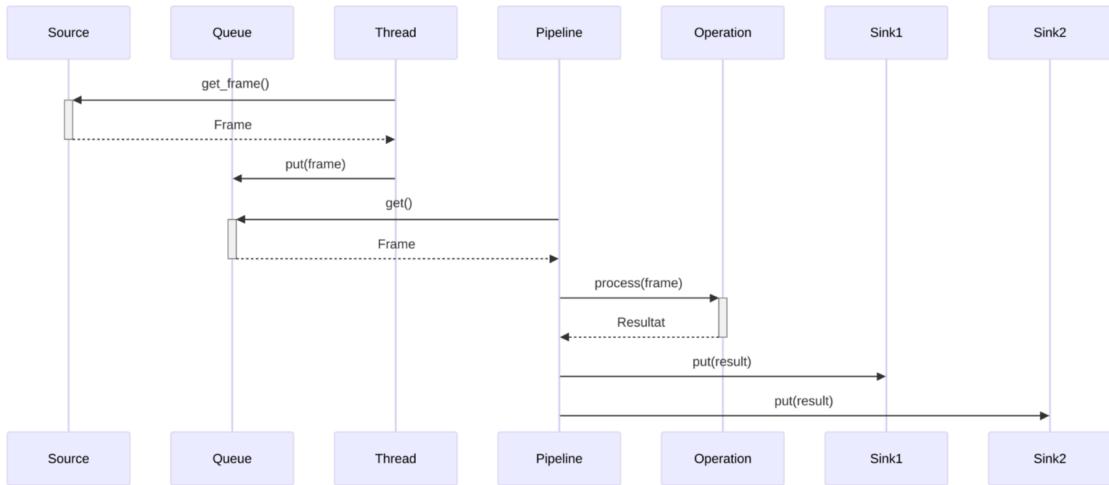


Abbildung 24: Sequenzdiagramm Edge ML Kamera Prozess

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

4.3 Prototypen

In diesem Kapitel werden unterschiedliche Prototypen vorgestellt, die verschiedene System-Setups für die Anwendung der entwickelten Applikation demonstrieren. Die Prototypen setzen Machine Learning auf dem Raspberry Pi 5 ein, unterscheiden sich jedoch in ihrer verwendeten Hardware und Konfiguration. Durch den Vergleich dieser Setups lassen sich die Vor- und Nachteile verschiedener Ansätze analysieren und bewerten.

4.3.1 Mitwelten Analyse

Dieser Prototyp setzt den Referenz-Use-Case aus Abschnitt 2.4.1 mit verschiedenen Modellen um. Die Analysen wurden jeweils mit Testbildern durchgeführt, daher ist der Energieverbrauch ohne Kamera zu betrachten. Die folgende Abbildung 25 zeigt die Beziehung zwischen der Anzahl Blumen auf einem Bild und der Zeit für die Bestäuberanalyse. Die angegebene Zeit beinhaltet die Detektierung der Blüten sowie die Detektierung von Bestäubern auf jeder detektierten Blüte.

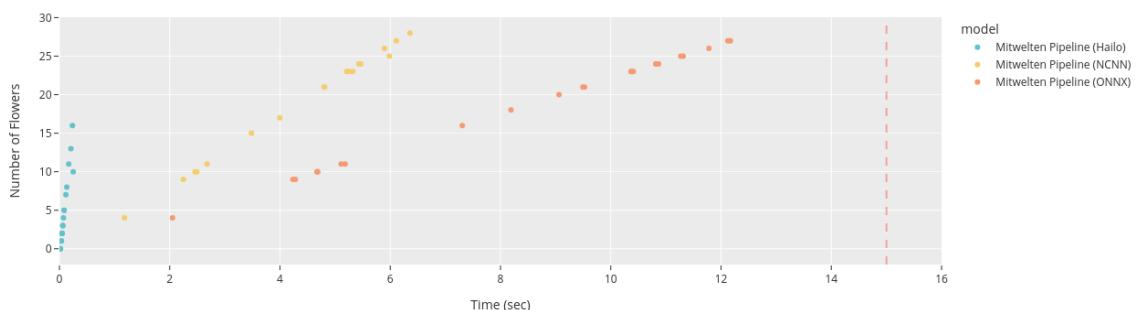


Abbildung 25: Vergleich der Frameworks mit Mitwelten-Analyse

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

Aus Abbildung 25 geht hervor, dass alle drei Pipelines die zeitlichen Anforderungen erfüllen. Dabei ist auch ersichtlich, dass sich die Ergebnisse aus der Analyse in Kapitel 3 widerspiegeln.

4.3.1.1 ONNX

Die Analyse hat gezeigt, dass die Ausführung der Modelle im ONNX-Format schon eine beträchtliche Zeiteinsparung pro Inferenz zur Folge hat. Zu diesem Zweck sind die Mitwelten-Modelle mittels Ultraalytics-Exportfunktion in das ONNX-Format konvertiert worden. Aus Abbildung 25 ist ersichtlich, dass der Zeitbedarf der Mitweltenanalyse im Bereich des Akzeptablen liegt. Der Energieverbrauch liegt bei rund 10 Watt. Durch den linearen Zusammenhang zwischen Anzahl Blumen und Inferenzzeit kann man die maximale Anzahl Blumen ermitteln, die innerhalb von 15 Sekunden berechenbar ist. Mit den Datenpunkten 4 Blumen in 2,051 Sekunden und 27 Blumen in 12,168 Sekunden lassen sich in 15 Sekunden rund 33 Blumen analysieren.

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{27 - 4}{12.168s - 2.051s} = 2.275$$

$$b = y_1 - (m * x_1) = 4 - (2.275 * 2.051s) = -0.66$$

$$y = m * x + b = 2.275 * 15 - 0.66 = 33.465$$

4.3.1.2 NCNN

Aus Kapitel 3 ist zu erwarten, dass die Umsetzung der Mitwelten-Analyse mit NCNN-Modellen nochmals deutlich schneller wird. Dies zeigt auch die Umsetzung dieses Prototyps. Der Zeitbedarf entspricht etwa der Hälfte von der Umsetzung mit ONNX-Modellen, während der Energiebedarf mit rund 10 Watt etwa gleich bleibt.

Die Konvertierung des Bestäuber-Modells in das NCNN-Format verursachte einige Schwierigkeiten. Folglich wurde das Yolov8n als alternative trainiert und verwendet. Das Yolov8n (nano) Modell ist von der Geschwindigkeit her vergleichbar mit dem Yolov5s (small), welches für die Bestäuberanalyse verwendet wurde [38]. Allerdings soll an dieser Stelle erwähnt sein, dass das ursprüngliche Modell mit einer Grösse von 480×480 trainiert worden ist. Dies wurde bei der adaptierung auf Yolov8 nicht berücksichtigt. Somit dürfte sich durch das leicht kleinere Modell die Inferenzdauer in einem ähnlichen Bereich bewegen.

Die Berechnung, wie viele Blüten in 15 Sekunden analysiert werden könnten, ergab in diesem Fall durch Einsetzen der Datenpunkte 4 Blüten in 1.179 Sekunden und 27 Blüten in 6,11 Sekunden das Resultat von 68,47 Blüten.

4.3.1.3 Hailo

Eine weitere Implementierung wurde mit der Beschleuniger-Hardware von Hailo umgesetzt. Um Modelle auf dieser Hardware auszuführen, müssen diese in das von Hailo definierte hef-Format konvertiert werden. Dies erreicht man mit der vom Hersteller zur Verfügung gestellten Software-Suite [39]. Diese beinhaltet im Wesentlichen einen Parser, Optimizer und Compiler. Diese

Prozesse müssen für jedes Modell richtig konfiguriert werden. Dafür ist die Untersuchung des zu konvertierenden Modells unerlässlich. Der folgende Befehl in Listing 2 zeigt beispielhaft, wie ein Kompilierungsprozess ausgeführt werden kann.

```
1 hailomz compile \
2   --hw-arch hailo8l \
3   --yaml ./hailo_model_zoo/hailo_model_zoo/cfg/networks/yolov8n.yaml \
4   --ckpt /local/shared_with_docker/yolov8n_pollinator_ep50_v1.onnx \
5   --classes 5 \
6   --end-node-names /model.22/cv2.0/cv2.0.2/Conv /model.22/cv3.0/cv3.0.2/Conv /model.22/cv2.1/cv2.1.2/Conv /model.22/cv3.1/cv3.1.2/
7   Conv /model.22/cv2.2/cv2.2.2/Conv /model.22/cv3.2/cv3.2.2/Conv \
    --calib-path /local/shared_with_docker/pollinators/
```

Listing 2: Bash Kommando für Hailo Modell-Konvertierung

Zur Ermittlung der `end-node-names` muss das Modell mit einem Tool wie netron.app [40] untersucht werden. Netron stellt die interne Struktur eines Machine Learning Modells anhand eines Graphen dar. Ebenso können Anpassungen in der YAML-Datei nötig sein, welche weitere Parameter zur Beschreibung des Modells definiert, wie zum Beispiel Anzahl Parameter oder Input / Output Shapes. Zusätzlich wird eine .alls-Datei geladen, welche verwendet wird, um die Output Funktionen und den Output-Node zu konfigurieren. Dieses File verweist auf eine Postprocessing-Config Datei, welche die End-Nodes nach dem Parsing-Prozess von Hailo definiert. Diese müssen je nach Modell angepasst werden. Die verschiedenen Konfigurationsfiles werden von Hailo als Templates in ihrem eigenen Model Zoo zur Verfügung gestellt [41]. Um die Nodes nach dem Parsing zu finden, stellt Hailo einen Profiler zur Verfügung. Folgende Abbildung 26 zeigt einen Screenshot des Hailo-Profiler-Interfaces. Der Ausschnitt zeigt einige Nodes eines Yolov8n-Modells.

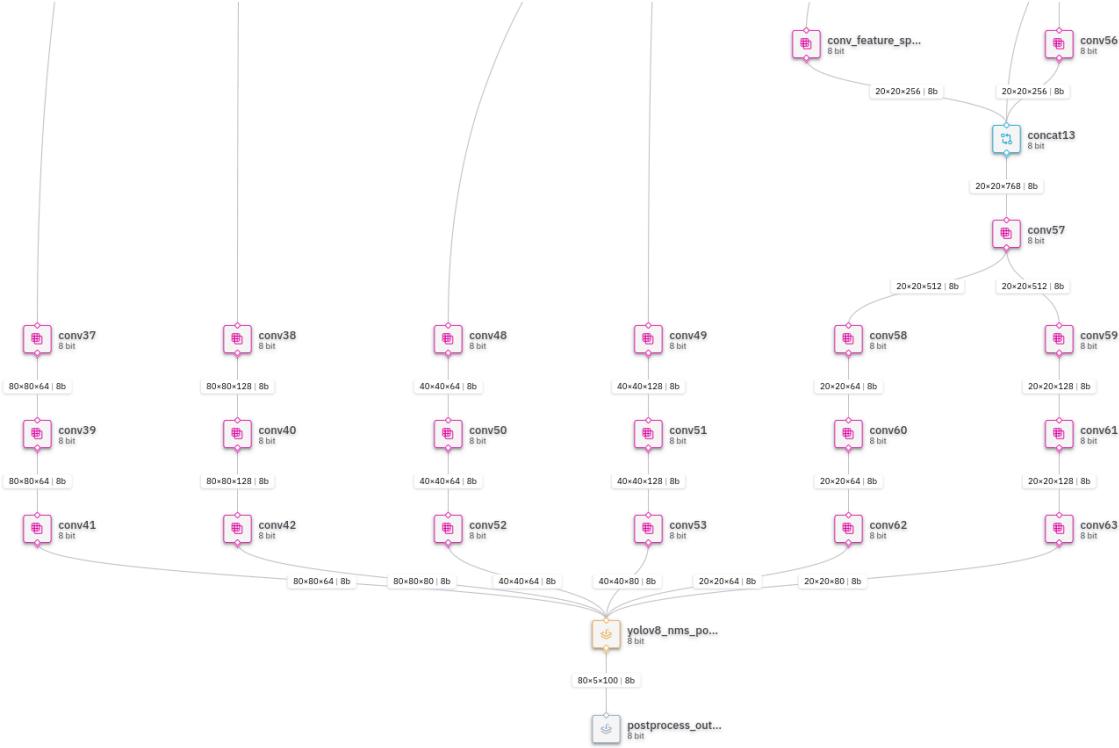


Abbildung 26: Teil eines Hailo Model Graph Yolov8n

(Quelle: Screenshot Hailo Profiler, Andri Wild, 2025)

Weil dieser Prozess für die bestehenden Yolov5-Modelle nicht erfolgreich war, kamen an dieser Stelle neu trainierte Yolov8-Modelle zum Einsatz. Das Training der Modelle lag an dieser Stelle nicht im Fokus, daher weisen diese nicht die gleiche Präzision auf wie die Yolov5-Modelle. Dies ist der Grund, dass auf Abbildung 25 weniger Blüten bei der Hailo-Pipeline ersichtlich sind. Die Untersuchung der Inferenzzeiten für die Bestäuberanalyse zeigt beeindruckende Resultate. Auch mit vielen Blüten ist die Performance sehr gut. Die Gesamtzeit der Inferenz von 13 Blüten dauert rund 200 Millisekunden. Des Weiteren ist auch der Energiebedarf bei rund 5 Watt, was der Hälfte der CPU-betriebenen Pipelines entspricht. Die Gesamtauslastung der CPU ist durch die Auslagerung allgemein sehr tief, so dass diese teilweise gedrosselt wird.

4.3.2 AI-Camera

Der zweite Prototyp verwendet die AI-Camera. Um eigens trainierte Modelle auf der AI-Camera auszuführen, müssen diese für den Sony IMX500-Chip konvertiert werden [42]. Dieser Prozess ist nicht trivial und konnte im Kontext dieser Arbeit nicht mehr erarbeitet werden. Trotzdem soll an dieser Stelle ein Use Case mit dieser Hardware vorgestellt werden. Erwähnenswert ist in diesem Setup die Konfiguration. Das Listing 3 zeigt, dass die AI-Kamera sowohl als `source` als auch als `pipe` definiert ist.

```

1   sources:
2     - name: AI-Camera
3       file_path: ./source/impl/aiCamera.py
4       class_name: AiCamera
5       parameters:
6         width: 640
7         height: 640
8     operations:
9       - name: AI-Camera
10      file_path: ./source/impl/aiCamera.py
11      class_name: AiCamera
12 sinks:
13   - name: console
14     class_name: Console
15     file_path: ./sink/impl/console.py

```

yaml

Listing 3: Konfiguration der Edge ML Kamera für AI-Kamera

Durch die Modularität der Applikation erhält hier die AI-Kamera eine Doppelrolle. Effektiv existiert in der Applikation nur eine Instanz der Klasse `AiCamera`. Die Applikation sorgt beim Laden der Klassen mittels Cache dafür, dass schon geladene Klassen nicht erneut geladen werden. Somit ist es möglich, Klassen für verschiedene Zwecke zu verwenden. Um dies zu ermöglichen, muss die jeweilige Klasse von allen entsprechenden Basis-Klassen erben.

4.4 Versuchsaufbau und Temperaturentwicklung

Besonders hervorzuheben sind die Eigenschaften der Hailo-Beschleuniger, die nicht nur für schnellere Inferenzzeiten sorgen, sondern auch die Wärmeentwicklung signifikant reduzieren. Die Abbildung 27 zeigt einen einfachen Versuchsaufbau, um die Temperaturentwicklung in einem Gehäuse zu untersuchen. Während das Gehäuse bei Einsatz der Hailo-Inferenz konstant auf einem niedrigen Temperaturniveau bleibt, führt die CPU-Inferenz zu einem rapiden Temperaturanstieg.



Abbildung 27: Versuchsaufbau: Temperatur Messung von Raspberry Pi 5
(Quelle: Eigene Aufnahme, Andri Wild, 2025)

Mit der CPU betriebenen Inferenz erhöhte sich die Temperatur im geschlossenen Gehäuse innerhalb einer Stunde auf 40 Grad, steigend. Die CPU-Temperatur lag bei ca. 85 Grad, womit die Drosselung aktiviert wird und die Rechenleistung sinkt.

Die optimierte Energieeffizienz der Beschleuniger hat positive Auswirkungen auf das Gehäusedesign – insbesondere bei Anwendungen wie Biodiversitätsanalysen, bei denen das Gerät oft im Freien und in wasserdichten Gehäusen eingesetzt wird. Der Wegfall von aktiver Belüftung eröffnet hier wesentliche Konstruktionsvorteile.

4.5 Erweiterung und Adaptierung der Edge ML Kamera

Die Installation der Anwendung ist im `README.md` des Github-Repository [43] ausführlich beschrieben und dient als Anleitung für die Inbetriebnahme. In der Anleitung werden auch die wichtigsten Schritte zur Erweiterung des Systems aufgeführt. Zudem bestehen für die auswechselbaren Komponenten Beispielimplementierungen. Um die Entwicklung zu vereinfachen, findet man im Utility-Ordner einige Helferfunktionen.

5 Validierung

In diesem Kapitel werden die implementierten Prototypen bezüglich ihrer Resultate und Praxistauglichkeit bewertet. Dabei stehen Aspekte wie Inferenzgeschwindigkeit, Energieverbrauch und Flexibilität im Fokus. Anschliessend werden zwei interessante Konkurrenzprodukte vorgestellt.

5.1 Bewertung der Prototypen

Die in Abschnitt 4.3.1 vorgestellten Prototypen zeigen, dass die Mitwelten-Bestäuber-Analyse erfolgreich auf einem Edge-Device umgesetzt werden kann. Eine solche Verarbeitung war mit Hardware im gleichen Preissegment bisher nicht realisierbar. Besonders der Hailo-Beschleuniger hat sich als gute Lösung für die Mitwelten-Analyse erwiesen. Die folgenden Vergleiche zeigen, welche Vorteile und Herausforderungen die verschiedenen Ansätze mit sich bringen und inwieweit sie sich für das Biodiversitätsmonitoring eignen.

5.1.1 Mitwelten-Analyse

Mit einem Bild- resp. Analyseintervall von 15 Sekunden sind alle der vorgestellten Varianten umsetzbar. Im Folgenden wird davon ausgegangen, dass dieses Intervall beibehalten wird, auch wenn die Bildrate aufgrund des Systems schneller sein könnte. In Anwendung auf die Mitwelten-Bestäuberanalysen gäben höhere Bildraten neue Herausforderungen, die adressiert werden müssten. Ein Bestäuber sollte pro Sichtung nur einmal gezählt werden, beispielsweise.

Die folgende Abbildung 28 zeigt das Bild mit 27 Blüten, welches für den Test verwendet wurde, mit und ohne Detektionen:



Abbildung 28: Testbild mit und ohne Inferenzresultate

(Quelle: https://github.com/sony/model_optimization, aufgerufen am 20.01.2025)

5.1.1.1 CPU Inferenz

Die Versuche mit Inferenzen auf der CPU zeigen während der Berechnungen eine sehr hohe Auslastung des Systems. Dies führt zu einem hohen Energiebedarf.

Besonders mit ONNX und vielen Blüten ist das System praktisch dauerhaft ausgelastet. Die Pausen zwischen den Inferenzen sind nur sehr kurz, wie Abbildung 29 verdeutlicht. Die CPU-Auslastung wurde während des Prozessierens des Testbildes mit 500 ms Abtastrate aufgenommen.

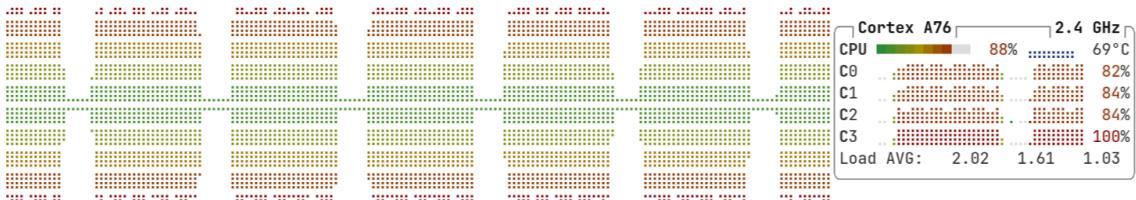


Abbildung 29: System Auslastung: ONNX-Inferenz mit 15 Sekunden Intervall

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

Mit den schnelleren Inferenzen des NCNN-Frameworks werden schon deutlich längere Pausen zwischen den Inferenzen sichtbar. Abbildung 30 visualisiert den selben Versuch.

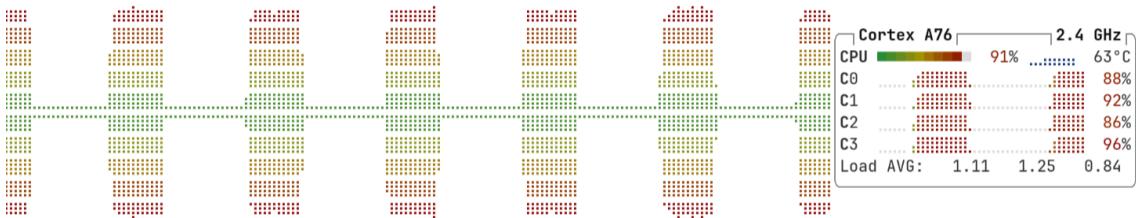


Abbildung 30: System Auslastung: NCNN-Inferenz mit 15 Sekunden Intervall

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

In beiden Fällen ist jedoch der Energiebedarf erhöht. Für Anwendungen, gerade im Bereich des Biodiversitätsmonitorings, kann ein energieeffizientes Setup wichtig sein. Zudem stellt der erhöhte Energiebedarf auch zusätzliche Anforderungen an ein entsprechend belüftetes Gehäuse. Dies kann im Einsatz auf dem Feld zu weiteren Herausforderungen führen.

Vorteile dieser Umsetzung sind die geringen Anschaffungskosten und die geringe Komplexität, da die Modelle einfach in das jeweilige Format zu konvertieren sind.

5.1.1.2 Hailo Inferenz

Die Tests mit den Hailo-Beschleunigern zeigen auf ganzer Linie gute Resultate. Die Inferenzzeiten sind extrem kurz, wodurch die Möglichkeit entsteht, wesentlich präzisere Modelle zu verwenden. Die Auslastung des Systems sinkt praktisch gegen Null, wie Abbildung 31 eindrücklich zeigt. Die kurzen Spitzen entstehen durch die Pre- und Postprocessing-Funktionen der Pipeline. In den langen Pausen zwischen den Inferenzen sinkt der Energiebedarf auf unter 4 Watt.



Abbildung 31: System Auslastung: Hailo8l-Inferenz mit 15 Sekunden Intervall

(Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)

Die gute Performance ist zu einem akzeptablen Preis erhältlich und das System überzeugt in allen Aspekten. Besonders für energiesparsame Applikationen ist dieses Setup sehr interessant. Diese Tests wurden mit dem Hailo8l 13 TOPS Beschleuniger gemacht. Für noch bessere Performance könnte man den etwas teureren Hailo8 mit 26 TOPS einsetzen. Einziger Nachteil ist die etwas komplizierte Konvertierung der Modelle, für welche ein gewisses Verständnis von ML-Modellen notwendig ist und die Zeit für Einarbeitung beansprucht.

5.1.2 AI-Kamera

Die AI-Kamera zeichnet sich durch einen niedrigen Energiebedarf aus. Zusätzlich ist das Host-System gut entlastet. Für die Mitwelten-Analyse kann diese Hardware nicht optimal eingesetzt werden, da nur eine Inferenz auf der Kamera ausgeführt werden kann. Dies hätte zur Folge, dass der Grossteil der Inferenzen auf der CPU oder einem zusätzlichen Beschleuniger betrieben werden muss. In beiden Varianten wäre es naheliegend, gerade alle Inferenzen auf derselben Recheneinheit auszuführen. Jedoch ist die Kamera für Biodiversitätsmonitoring sehr interessant, sofern nur eine Inferenz pro Bild notwendig ist. Besonders für energieeffiziente Systeme, beispielsweise solarbetriebene, könnte die Kamera in Kombination mit einem leistungsschwächeren und dadurch energieeffizienten Host-System zum Einsatz kommen.

5.2 Konkurrenzprodukte

Die Möglichkeit, Analysen direkt auf einem Edge-Device auszuführen, ist eine vielversprechende Anwendung, was durch die Verfügbarkeit von Produkten auf dem Markt bestätigt wird. Zwei besonders interessante Geräte sind die reCamera [44] und die EcoEye-Kamera [45], die beide über SeeedStudio erhältlich sind.

Die reCamera lässt sich mithilfe von Node-Red konfigurieren, wodurch die Konfiguration von Applikationen ohne Programmierkenntnisse möglich ist. Dies bietet insbesondere für Projekte ohne Informatik-affine Mitarbeitende oder für Citizen-Science-Anwendungen einen erheblichen Vorteil. Die Kamera ist äusserst kompakt, jedoch nicht wasserdicht und erfordert daher einen zusätzlichen Schutz für den Ausseneinsatz.

Die EcoEye-Kamera wurde speziell für den Ausseneinsatz entwickelt und verfügt über einen grossen Akku, der den Betrieb in Gebieten ohne externe Stromversorgung ermöglicht. Ihre Bau-

weise ähnelt der einer herkömmlichen Wildkamera, bietet jedoch zusätzliche Rechenleistung, um Analysen direkt auf dem Gerät durchzuführen. Dadurch können Daten bereits vor Ort verarbeitet werden.

6 Fazit

Die vorliegende Arbeit zeigt, dass der technologische Fortschritt im Bereich Edge-Computing die dezentrale Verarbeitung von Machine-Learning-Modellen auf Edge ML Kameras realisierbar macht. Die durchgeführten Performance-Messungen verdeutlichen, dass moderne Hardware-Beschleuniger wie Hailo oder AI-Kameras signifikante Verbesserungen in der Inferenzgeschwindigkeit bei gleichzeitig reduziertem Energieverbrauch ermöglichen. Damit bietet sich eine vielversprechende Alternative zu klassischen Cloud-basierten Architekturen. Dies gilt besonders für Citizen-Science-Anwendungen im Biodiversitätsmonitoring, wie im SNF-Projekt Mitwelten verwendet. Ohne ein zentrales Backend kann ein geschlossenes System betrieben werden, wodurch der technische Aufwand reduziert werden kann.

Die entwickelte Lösung für eine Edge ML Kamera basiert auf einer modularen Softwarearchitektur, die verschiedene Machine-Learning-Modelle unterstützt und eine einfache Anpassung für zukünftige Anwendungen im Bereich Citizen Science und Biodiversitätsforschung ermöglicht. Eine Portierung der bisherigen Mitwelten Bestäuberanalyse konnte auf der entwickelten Anwendung erfolgreich durchgeführt werden, wobei signifikante Verbesserungen der Analysegeschwindigkeit erzielt wurden.

Abbildungsverzeichnis

Abbildung 1: Erweiterte thematische Karte zum Lernen von Freiwilligen (Quelle: The Science of Citizen Science, S.300)	8
Abbildung 2: ONNX Framework als Drehscheibe (Quelle: https://docs.ultralytics.com/integrations/onnx/ , aufgerufen am 02.02.2025)	10
Abbildung 3: Disziplinen des Machine Learnings (Quelle: https://github.com/sony/model_optimization , aufgerufen am 20.01.2025)	12
Abbildung 4: Symbolbild neuronales Netzwerk (Quelle: https://lamarr-institute.org/blog/deep-neural-networks , abgerufen am 23.01.2025)	14
Abbildung 5: Raspberry Pi 5 (Quelle: https://www.pi-shop.ch/raspberry-pi-5-16gb-ram?src=raspberrypi , aufgerufen am 23.01.2025)	16
Abbildung 6: Coral PCI Accelerator (Quelle: https://www.coral.ai/products/m2-accelerator-dual-edgetpu , aufgerufen am 22.01.2025). 17	
Abbildung 7: Hailo Accelerator auf Raspberry Pi 5 (Quelle: https://www.pi-shop.ch/raspberry-pi-ai-hat-13t , aufgerufen am 22.01.2025)	18
Abbildung 8: Raspberry Pi AI-Kamera (Quelle: https://www.berrybase.ch/raspberry-pi-ai-camera , aufgerufen am 22.01.25)	18
Abbildung 9: Raspberry Pi AI-Kamera Architektur (Quelle: https://www.raspberrypi.com/documentation/accessories/ai-camera.html , aufgerufen am 20.01.2025)	19
Abbildung 10: IoT Kamera im Feld (Quelle: mitwelten.ch , aufgerufen am 23.01.2025)	20
Abbildung 11: Vorgang Mitwelten Bestäuber Analyse (Quelle: Automated Analysis for Urban Biodiversity Monitoring, Timeo Wullschleger)	21
Abbildung 12: Mitwelten Analyse auf Raspberry Pi 3 (Quelle: Automated Analysis for Urban Biodiversity Monitoring S.61, Timeo Wullschleger) ...	21
Abbildung 13: Model Evaluation Dashboard (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)	24
Abbildung 14: Vergleich: Inferenzzeit YOLOv5, v8 auf Raspberry Pi 5	25
Abbildung 15: Vergleich: ONNX und NCNN auf Raspberry Pi 5	26
Abbildung 16: Vergleich: Raspberry Pi 4 vs. Pi 5 (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)	27

Abbildung 17: Vergleich: Raspberry Pi 5 mit und ohne Kühlung (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)	27
Abbildung 18: Vergleich: Raspberry Pi 5 vs. Coral PCI Edge TPU	28
Abbildung 19: Vergleich Hailo Accelerator vs. Coral PCI Edge TPU (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)	29
Abbildung 20: Vergleich: Kosten vs. Geschwindigkeit mit YOLOv8n (Quelle: Screenshot eigene Entwicklung, Andri Wild, 2025)	30
Abbildung 21: Pipeline Komponenten (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	31
Abbildung 22: Webinterface Edge ML Kamera (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	34
Abbildung 23: Klassen Diagramm (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	35
Abbildung 24: Sequenzdiagramm Edge ML Kamera Prozess (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	36
Abbildung 25: Vergleich der Frameworks mit Mitwelten-Analyse (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	36
Abbildung 26: Teil eines Hailo Model Graph Yolov8n (Quelle: Screenshot Hailo Profiler, Andri Wild, 2025)	39
Abbildung 27: Versuchsaufbau: Temperatur Messung von Raspberry Pi 5 (Quelle: Eigene Aufnahme, Andri Wild, 2025)	41
Abbildung 28: Testbild mit und ohne Inferenzresultate (Quelle: https://github.com/sony/model_optimization , aufgerufen am 20.01.2025)	42
Abbildung 29: System Auslastung: ONNX-Inferenz mit 15 Sekunden Intervall (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	43
Abbildung 30: System Auslastung: NCNN-Inferenz mit 15 Sekunden Intervall (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	43
Abbildung 31: System Auslastung: Hailo8l-Inferenz mit 15 Sekunden Intervall (Quelle: Screenshot eigene Darstellung, Andri Wild, 2025)	44

Bibliographie

- [1] A. Bonn *u. a.*, „Grünbuch Citizen Science Strategie 2020 für Deutschland“, Berlin, 2016.
- [2] K. Vohland *u. a.*, Hrsg., „The Science of Citizen Science“. Springer International Publishing, Cham, 2021. doi: 10.1007/978-3-030-58278-4.
- [3] „Vorhaben - Schweiz forscht“. Zugegriffen: 7. Februar 2025. [Online]. Verfügbar unter: <https://www.schweizforscht.ch/projekte>
- [4] „TensorFlow“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://www.tensorflow.org/>
- [5] „PyTorch“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://pytorch.org/>
- [6] „OpenAI standardizes on PyTorch“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://openai.com/index/openai-pytorch/>
- [7] „ONNX | Home“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://onnx.ai/>
- [8] Ultralytics, „ONNX“. Zugegriffen: 2. Februar 2025. [Online]. Verfügbar unter: <https://docs.ultralytics.com/integrations/onnx>
- [9] H. Ni und The ncnn contributors, „ncnn“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://github.com/Tencent/ncnn>
- [10] „Das führende KI-Chip-Unternehmen für Edge-Geräte“. Zugegriffen: 27. Januar 2025. [Online]. Verfügbar unter: <https://hailo.ai/de/company-overview/>
- [11] Ultralytics, „Home“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://docs.ultralytics.com/>
- [12] Ultralytics, „YOLOv5“. Zugegriffen: 23. Januar 2025. [Online]. Verfügbar unter: <https://docs.ultralytics.com/models/yolov5>
- [13] „Sparsification: Compressing Neural Networks | Neural Magic Documentation“. Zugegriffen: 4. Februar 2025. [Online]. Verfügbar unter: <https://docs.neuralmagic.com/guides/sparsification/>
- [14] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, und A. Y. Zomaya, „Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence“, *IEEE Internet of Things Journal*, Bd. 7, Nr. 8, S. 7457–7469, Aug. 2020, doi: 10.1109/JIOT.2020.2984887.
- [15] „Raspberry Pi 4 Model B - 8GB“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/raspberry-pi-4-model-b-8gb>
- [16] „Raspberry Pi 5 Model B 8GB“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/raspberry-pi-5-8-gb>

- [17] „BeagleY®-AI“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.seeedstudio.com/BeagleYr-AI-beagleboard-orgr-4-TOPS-AI-Acceleration-powered-by-TI-AM67A.html>
- [18] „TPU (Tensor Processing Unit)“. Zugegriffen: 23. Januar 2025. [Online]. Verfügbar unter: <https://www.ultralytics.com/glossary/tpu-tensor-processing-unit>
- [19] „What is a Neural Processing Unit (NPU)? | IBM“. Zugegriffen: 23. Januar 2025. [Online]. Verfügbar unter: <https://www.ibm.com/think/topics/neural-processing-unit>
- [20] „NPU vs GPU: What's the Difference? | IBM“. Zugegriffen: 23. Januar 2025. [Online]. Verfügbar unter: <https://www.ibm.com/think/topics/npu-vs-gpu>
- [21] „Coral USB Accelerator“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/coral-usb-accelerator>
- [22] „Pineboards Hat AI! Dual, Edge Coral TPU für Raspberry Pi 5, Bundle - kaufen bei BerryBase“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.berrybase.ch/pineboards-hat-ai-dual-edge-coral-tpu-fuer-raspberry-pi-5-bundle>
- [23] „Pineboards Hat AI!, Coral Edge TPU Erweiterung für Raspberry Pi 5, Bundle - kaufen bei BerryBase“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.berrybase.ch/pineboards-hat-ai-coral-edge-tpu-erweiterung-fuer-raspberry-pi-5-bundle>
- [24] „Raspberry Pi AI HAT+ (13T)“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/raspberry-pi-ai-hat-13t>
- [25] „Raspberry Pi AI HAT+ (26T)“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.pi-shop.ch/raspberry-pi-ai-hat-26t>
- [26] „Raspberry Pi AI Camera - kaufen bei BerryBase“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.berrybase.ch/raspberry-pi-ai-camera>
- [27] „AI Camera - Raspberry Pi Documentation“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: <https://www.raspberrypi.com/documentation/accessories/ai-camera.html>
- [28] bdm, „Verlässliche Daten über unsere Lebensgrundlage“. Zugegriffen: 15. Januar 2025. [Online]. Verfügbar unter: <https://www.biodiversitymonitoring.ch/index.php/de/>
- [29] T. Wullschleger, „Data Acquisition for Urban Biodiversity Monitoring“.
- [30] T. Wullschleger, „Automated Analysis for Urban Biodiversity Monitoring“.
- [31] „ip7-ml-model-eval/dashboard at main · andriwild/ip7-ml-model-eval“. Zugegriffen: 2. Februar 2025. [Online]. Verfügbar unter: <https://github.com/andriwild/ip7-ml-model-eval/tree/main/dashboard>
- [32] awild, „andriwild/ip7-ml-model-eval“. Zugegriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://github.com/andriwild/ip7-ml-model-eval>

- [33] G. Jocher, A. Chaurasia, und J. Qiu, „Ultralytics YOLOv8“. [Online]. Verfügbar unter: <https://github.com/ultralytics/ultralytics>
- [34] T.-Y. Lin *u. a.*, „Microsoft COCO: Common Objects in Context“. [Online]. Verfügbar unter: <https://arxiv.org/abs/1405.0312>
- [35] M. Hussain, „YOLOv5, YOLOv8 and YOLOv10: The Go-To Detectors for Real-time Vision“. Zugriffen: 4. Februar 2025. [Online]. Verfügbar unter: <http://arxiv.org/abs/2407.02988>
- [36] „Products“. Zugriffen: 16. Januar 2025. [Online]. Verfügbar unter: <https://coral.ai/products/#prototyping-products>
- [37] T. Tapuhi *u. a.*, „Hailo Model Zoo“. Zugriffen: 16. Januar 2025. [Online]. Verfügbar unter: https://github.com/hailo-ai/hailo_model_zoo
- [38] Ultralytics, „YOLOv5 vs YOLOv8: A Detailed Comparison for Object Detection“. Zugriffen: 29. Januar 2025. [Online]. Verfügbar unter: <https://docs.ultralytics.com/compare/yolov5-vs-yolov8>
- [39] „Software Suite for AI Applications & Deep Learning | Hailo AI“. Zugriffen: 29. Januar 2025. [Online]. Verfügbar unter: <https://hailo.ai/products/hailo-software/hailo-ai-software-suite/>
- [40] „Netron“. Zugriffen: 7. Februar 2025. [Online]. Verfügbar unter: <https://netron.app/>
- [41] „hailo_model_zoo/hailo_model_zoo/cfg at master · hailo-ai/hailo_model_zoo“. Zugriffen: 29. Januar 2025. [Online]. Verfügbar unter: https://github.com/hailo-ai/hailo_model_zoo/tree/master/hailo_model_zoo/cfg
- [42] „IMX500 Converter | Sony Semiconductor Solutions Group“. Zugriffen: 29. Januar 2025. [Online]. Verfügbar unter: <https://developer.aitrios.sony-semicon.com/en/raspberry-pi-camera/documentation/imx500-converter?version=3.14.3&progLang=>
- [43] awild, „andriwild/ip7-edge-ml-cam“. Zugriffen: 7. Februar 2025. [Online]. Verfügbar unter: <https://github.com/andriwild/ip7-edge-ml-cam>
- [44] „reCamera“. Zugriffen: 1. Februar 2025. [Online]. Verfügbar unter: https://www.seeedstudio.com/recamera?srslid=AfmBOopbSxSK-BPmVHDRVm0pGJWa_aRpSmQuWH0XmIjh2ARMrHknYQyG
- [45] „EcoEye: OpenMV H7+ Camera with Sensor Integration for Environmental Monitoring“. Zugriffen: 1. Februar 2025. [Online]. Verfügbar unter: <https://www.seeedstudio.com/EcoEye-Embedded-Vision-Camera-p-5843.html>

Hiermit bestätige ich, dass die eingereichte Projektarbeit mit dem Titel «Edge ML Camera for Citizen Sciene» das Resultat meiner persönlichen Erarbeitung der Themen ist. Ich habe keine anderen als die angegebenen Quellen benutzt.

Windisch, 07.02.2025

Andri Wild