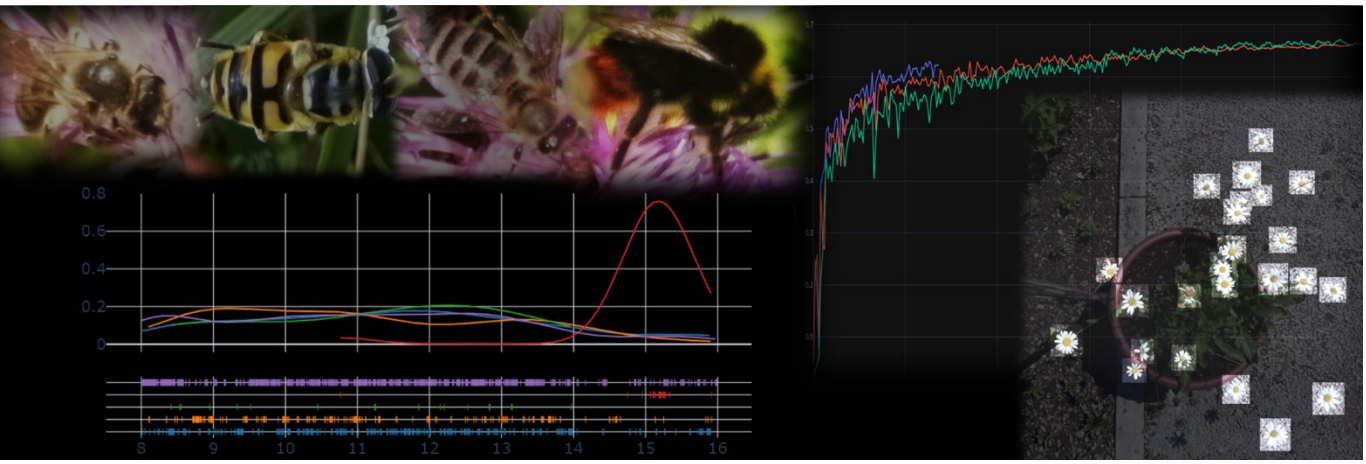
Fachhochschule Nordwestschweiz

MASTER OF SCIENCE IN ENGINEERING

Projektdokumentation P8

Automated Analysis for Urban Biodiversity Monitoring



Autor	Timeo Wullschleger timeo.wullschleger@fhnw.ch
Betreuung	Prof. Thomas Amberg
Ort, Datum	Windisch, 31.07.2022

Abstract

Human existence depends on biodiversity. Pollinators play a special role because plant reproduction depends on them. Due to human actions and environmental influences, the population of pollinators varies greatly around the world. Since they play a key role in biodiversity, monitoring pollinators is important to detect changes early on.

Within the scope of this project, an automated evaluation was developed for systematically collected image recordings. The evaluation pipeline consists of two steps, where flowers are detected in the first step and pollinators are detected on the flowers in the second step. Five classes of pollinators can be detected. The objects are reliably recognized, making scientifically relevant statements about biodiversity feasible.

Four concepts for automated evaluation were developed and tested. Three of them are carried out on edge devices, one in the backend. The smallest system is based on a single board computer and can be operated offline for long periods of time. The most efficient version is the evaluation in the backend.

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangslage

Das interdisziplinäre SNF Forschungsprojekt Mitwelten befasst sich mit dem Monitoring der Biodiversität auf urbanem Terrain. Im Rahmen des Projekts wurde ein Proof of Concept erarbeitet, womit ökologisch relevante Daten erfasst werden. Konkret wurde ein Kamerasystem zum Fotografieren von Blumen entwickelt, womit das Ziel verfolgt wird, Bestäuber auf den Blüten zu erkennen und somit eine Aussage über die Biodiversität zu machen. Damit die Ergebnisse nicht von Faktoren wie dem Nährstoffgehalt des Bodens oder der Feuchtigkeit abhängen, befinden sich die Blumen in Töpfen und können somit als Phytometer betrachtet werden. In einer ersten Fallstudie wurden während drei Monaten knapp 1.5 M Bilder erfasst, die als Ausgangslage für die Entwicklung einer automatisierten Auswertung dienen.

1.2 Aufgabenstellung

Im Rahmen der Projektarbeit soll analysiert werden, wie die Auswertung der erfassten Daten automatisiert werden kann. Konkret soll erforscht werden, ob es möglich ist, Bestäuber auf den Fotos zu erkennen und einer Klasse zuzuweisen. In einem ersten Schritt sollen die Grundlagen für die Auswertung von Bilddaten recherchiert und dokumentiert werden. Anschliessend sollen verschiedene Ansätze implementiert und verglichen werden. Für die Automatisierung der Auswertung sollen verschiedene Konzepte erarbeitet werden, die auf unterschiedlichen Plattformen durchgeführt werden.

1.3 Strukturierung der Dokumentation

Die Dokumentation ist in fünf Abschnitte gegliedert. Im Kapitel Theoretische Grundlagen werden die Hintergründe der eingesetzten Technologien beschrieben. Das Kapitel Datenexploration beschreibt die Erkundung der im Rahmen des Forschungsprojekts Mitwelten erfassten Bildern. Die Entwicklung von Deep Learning Models für die Auswertung von Aufnahmen wird im Kapitel Auswertung von Bilddaten dokumentiert. Das nächste Kapitel, Automatisierung des Auswerteverfahrens, beschreibt vier Ansätze für eine automatisierte Auswertung auf unterschiedlichen Plattformen. Im Kapitel Analyse von Resultaten werden Tools für die Visualisierung von Resultaten der Auswertung dokumentiert.

2 Theoretische Grundlagen

In diesem Kapitel werden die Hintergründe und Technologien beschrieben, die für die Entwicklung der Auswertung von Bilddaten relevant sind.

2.1 Machine Learning & Deep Learning

Künstliche Intelligenz ein Sammelbegriff für die Automatisierung von intelligentem Verhalten. Im allgemeinen wird angestrebt, mit Software eine möglichst «menschenähnliche» Intelligenz zu entwickeln [1]. Machine Learning ist eine Teilmenge der Künstlichen Intelligenz (AI) und wird für die Generierung von Wissen aus Erfahrung eingesetzt. Deep Learning ist eine Disziplin des Machine Learnings, unter Einsatz von künstlichen neuronalen Netzwerken. Abbildung 1 zeigt eine Übersicht der Einordnung von AI, ML und DL.

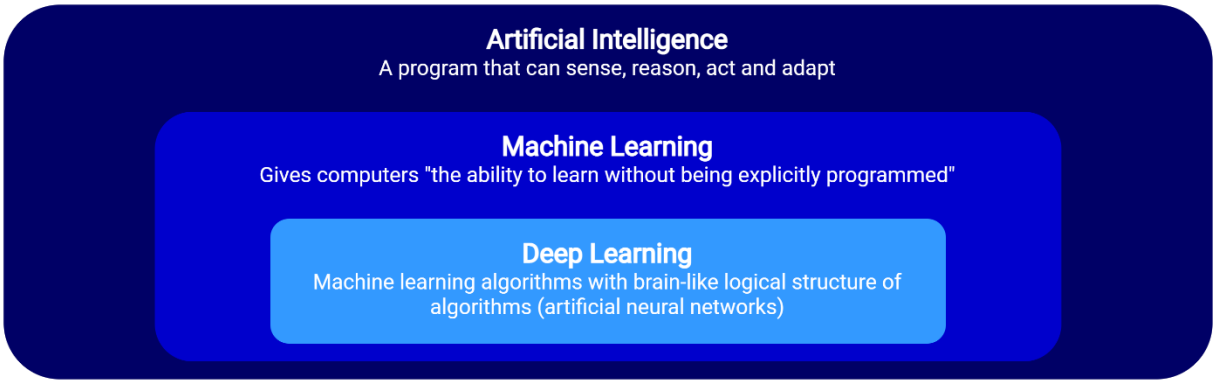


Abbildung 1 Übersicht AI, ML, DL

ML Algorithmen können in zwei Hauptkategorien unterteilt werden:

- Mit Supervised Learning werden anhand von bekannten Daten mit bekannten Klassen Merkmale und Regeln gelernt, um damit neue Daten zu klassifizieren.
- Unsupervised Learning erkundet unbekannte Datensätze, bietet Einblicke in die Struktur und kann für die Detektion von Anomalien eingesetzt werden.

Neben den Hauptkategorien gibt es weitere Kategorien wie Reinforcement Learning, wobei das Model mit der Umgebung interagiert und dabei selbständig lernt. Ein Beispiel dafür sind dynamisch aktualisierte Produktvorschläge in Onlineshops.

2.1.1 Training eines ML Models

Für das Training eines Object Detection ML Algorithmus muss ein Datenset vorbereitet werden. Ein Datenset beinhaltet Bilder und die jeweiligen Annotationen. Das Datenset wird in ein Trainings- und ein Test Set geteilt. Das Test Set wird für die Validierung verwendet und darf dem Algorithmus während dem Training nicht bekannt sein. Bei sehr grossen Datensets kann das Test Set in Test- und Validierungsset aufgeteilt werden, um die Resultate der Validierung anhand des Test Sets zu bestätigen. Abbildung 2 zeigt einen Train/Test Split und einen Split mit zusätzlichem Validierungsset. Die Grösse des Test Set ist abhängig von der Grösse des Datensets. Das Test Set muss zwei Konditionen erfüllen:

- Es muss gross genug sein, um statistisch relevante Resultate zu erzeugen.
- Es soll das gesamte Datenset repräsentieren und nicht über andere Charakteristiken als das Trainingsset verfügen.

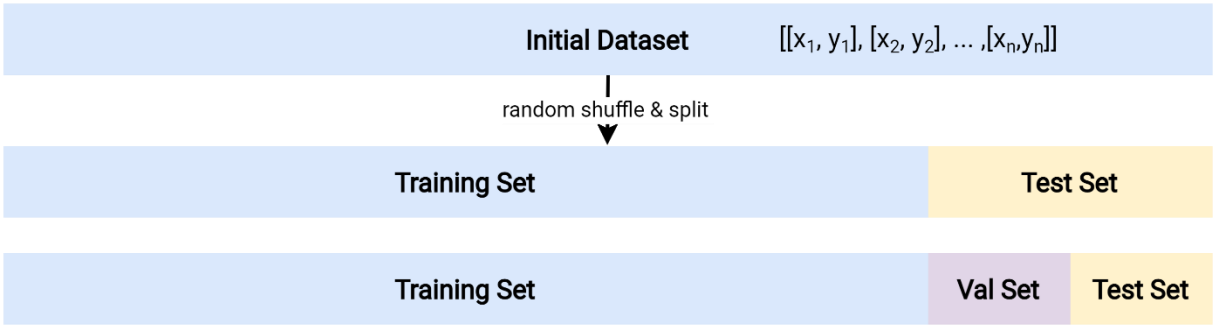


Abbildung 2 Splitting eines Datensets

Während dem Training probiert das Model verschiedene Parameter aus und führt einen Inferenzdurchlauf auf dem Trainingsset durch. Danach wird überprüft, wie weit entfernt die vorgeschlagenen Werte von den realen Werten liegen. Diese Distanz wird als Loss bezeichnet. In iterativen Schritten werden die Parameter so angepasst, dass der Loss minimiert wird. Abbildung 3 zeigt den Ablauf des Trainingsvorgangs.

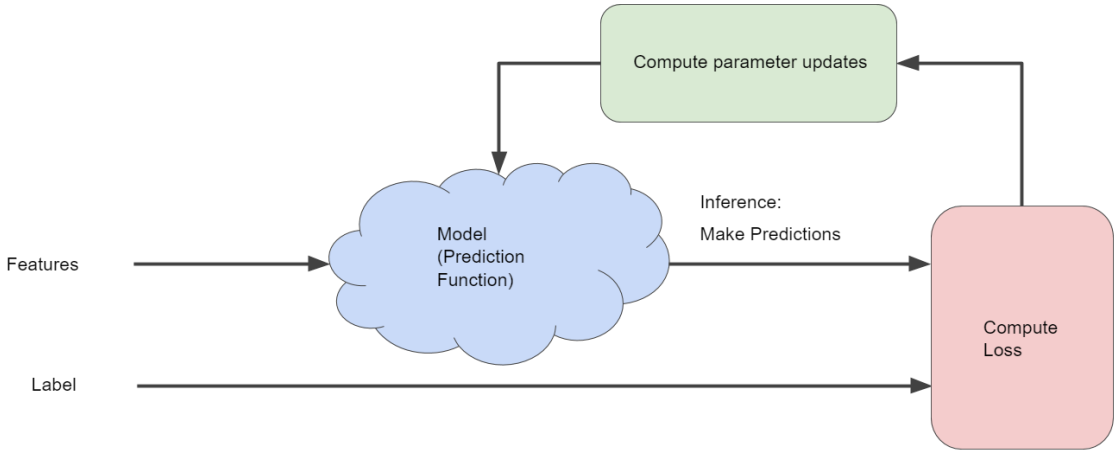


Abbildung 3 Iteratives Training eines ML Models [2]

2.1.2 Generalisierung des Models

Wenn ein ML Model trainiert wird, will man nicht nur das Trainingsset lernen. Das Ziel ist es, dass das Model generalisiert wird und auch auf unbekannte Daten angewendet werden kann. Um die Generalisierung zu überprüfen, wird das Model während dem Training regelmässig mit dem Test Set validiert. Je länger das Model trainiert wird, desto komplexer wird es. Der Loss (Error) des Trainingssets verschwindet nach einer bestimmten Anzahl Durchläufe fast ganz. Sobald der Loss des Test Sets wieder zunimmt, geschieht Overfitting, wie es in Abbildung 4 visualisiert wird.

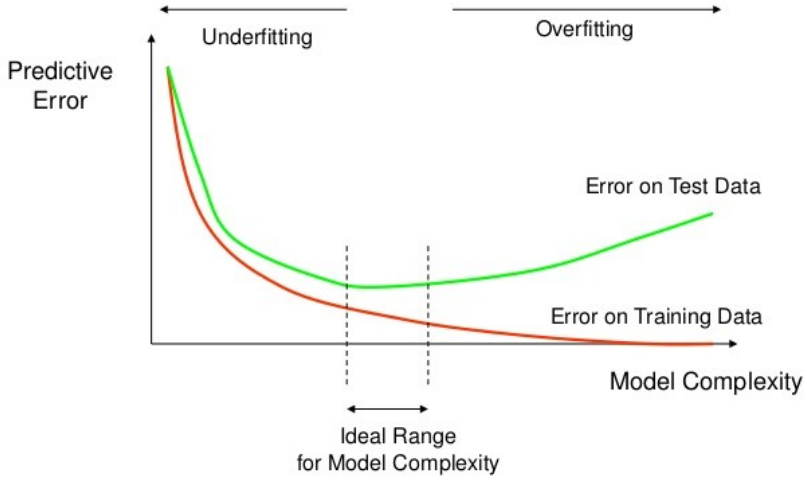


Abbildung 4 Underfitting & Overfitting [3]

Solange der Loss des Test Sets abnimmt, befindet sich das Model im Underfitting Bereich und kann noch weiter trainiert werden. Massnahmen gegen Overfitting sind:

- Early Stopping, wobei das Training frühzeitig beendet wird.
- Dropout Layers, die einen festgelegten Prozentsatz der gelernten Parameter wieder zurücksetzen.
- Data Augmentation zum Diversifizieren der Trainingsdaten.
- Stochastische Regulationen.

2.1.3 Hyperparameter

Hyperparameter sind Parameter, mit welchen der Trainingsprozess kontrolliert wird. Die wichtigsten sind:

- Die Learning Rate. Sie bestimmt, welchen Prozentsatz an neu gelernten Regeln pro Epoche in das Model übernommen werden.
- Die Gewichtung von verschiedenen Loss Parametern.
    - Object Loss, der entsteht, wenn ein Objekt nicht erkannt wird
    - Box Loss, der bei einer ungenauen Position einer Prediction auftritt.
    - Class Loss, wenn eine falsche Klasse bestimmt wird.
- Parameter für die automatische Data Augmentation.

Die optimalen Hyperparameter sind abhängig von der Art der Models, der Grösse des Models und vom Datenset. Durch eine Hyperparameteroptimierung, wovon eine praktische Anwendung im Kapitel Hyperparameter Evolving beschrieben ist, können die optimalen Hyperparameter angenähert werden.

2.1.4 Metriken

In den Bereichen Machine- und Deep Learning gibt es verschiedene Metriken, um Eigenschaften und Performance eines Models zu beschreiben. Die wichtigsten davon werden folgend dokumentiert.

2.1.4.1 Precision & Recall

Die Precision und der Recall werden bei der Validierung eines Models berechnet. Die Ergebnisse können True Positives, True Negatives, False Positives und False Negatives enthalten. Um die Werte zu visualisieren, wird oft eine Confusion Matrix, wie in Abbildung 5, generiert.

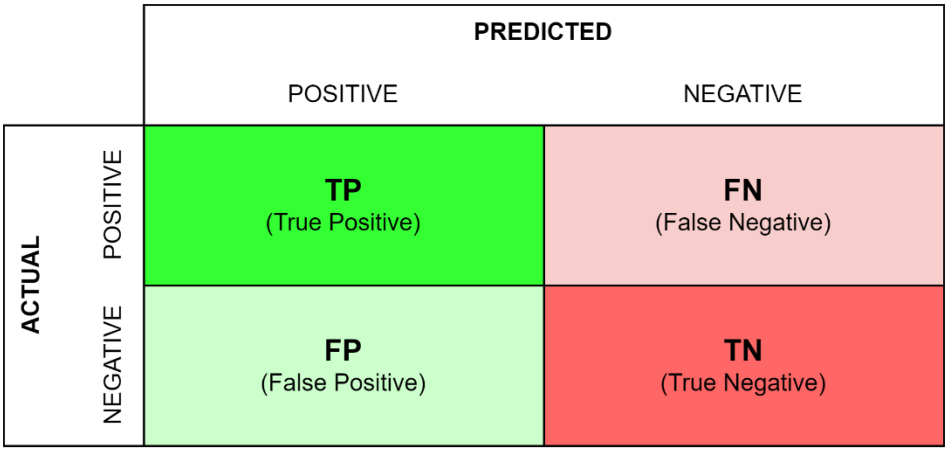


Abbildung 5 Confusion Matrix

Wenn ein Algorithmus Blumen auf einem Bild erkennen soll und eine Blume als Blume erkennt, ist das ein True Positive. Wenn er einen Stein als Blume erkennt, ist es ein False Positive. Wenn er eine Blume nicht als Blume erkennt, ist es ein False Negative. Wenn er ein Stein nicht als Blume erkannt, ist es ein True Negative.

Für die Berechnung der Precision wird die Anzahl der TP durch die Summe aller TP und FP geteilt:

$$Precision = \frac{TP}{(TP + FP)}$$

Sie sagt aus, wie viele der erkannten Blumen auch wirklich Blumen sind. Wenn sich fünf Blumen auf einem Bild befinden und der Algorithmus eine Blume richtig erkannt und sonst nichts, ist die Precision 1. Der Recall beschreibt, wie viele der vorhandenen Blumen erkannt wurden. Für die Berechnung werden die TP durch die Summe der TP und FN geteilt:

$$Recall = \frac{TP}{(TP + FN)}$$

Der Recall vom vorherigen Beispiel beträgt 0.2, da nur eine von fünf Blumen erkannt wurde. Falls auf dem Bild mit fünf Blumen alle Blumen plus fünf Steine als Blumen erkannt werden, ist der Recall eins und die Precision 0.5.

Ein optimales Model hat eine hohe Precision und einen hohen Recall. Der F1 Score kombiniert die beiden Werte durch die Berechnung des harmonischen Mittels:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Ein F1 Score von 1.0 wird erreicht, wenn Precision und Recall 1.0 sind. Ein Model mit einer Precision von 1.0 und einem Recall von 0.2 erreicht einen F1 Score von 0.33, ein Model mit Precision 0.5 und Recall 1.0 erreicht einen F1 Score von 0.66.

2.1.4.2 Intersection over Union

Der IoU Wert sagt aus, wie fest zwei Bounding Boxen überlappen. Für die Berechnung wird die überlappende Fläche durch die Gesamtfläche von zwei Boxen geteilt, wie es in Abbildung 6 ersichtlich ist. Anwendung findet die IoU unter anderem bei der Non Maximum Suppression (NMS).

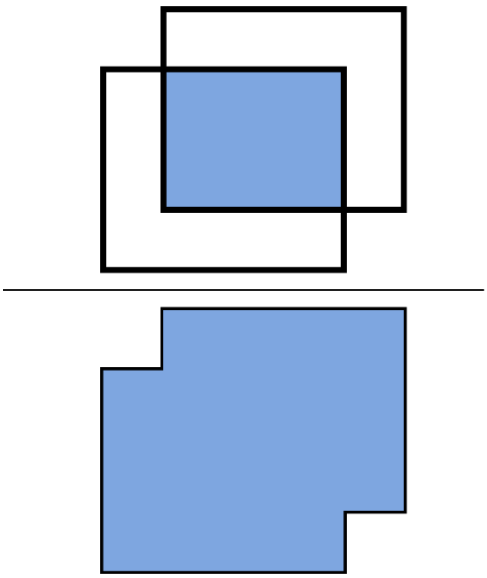


Abbildung 6 Grundlagen IoU

Für die Validierung von Object Detection Models wird oft ein IoU Grenzwert gesetzt. Für alle detektierten Objekte wird die IoU zu den Groundtruth Labels berechnet. Falls die erkannten Objekte zu wenig mit den realen Objekten überlappen, werden sie als False Positive gezählt. Mit einem sehr geringen IoU Threshold ist der Recall hoch und die Precision tief, mit einem hohen IoU Threshold ist die Precision hoch und der Recall tief.

2.1.4.3 Mean Average Precision

Für die Beschreibung der Genauigkeit von Object Detection Models wird die mean Average Precision (mAP) mit definiertem IoU Threshold berechnet. Weit verbreitet ist mAP@iou.5:0.95. Für jede Klasse werden Precision und Recall bei einem IoU Threshold von 0.5 bis 0.95 in 10 Schritten berechnet. Daraus resultiert eine Precision-Recall Curve, wie sie in Abbildung 7 visualisiert ist. Die Average Precision einer Klasse ist die Fläche unter der blauen Kurve.

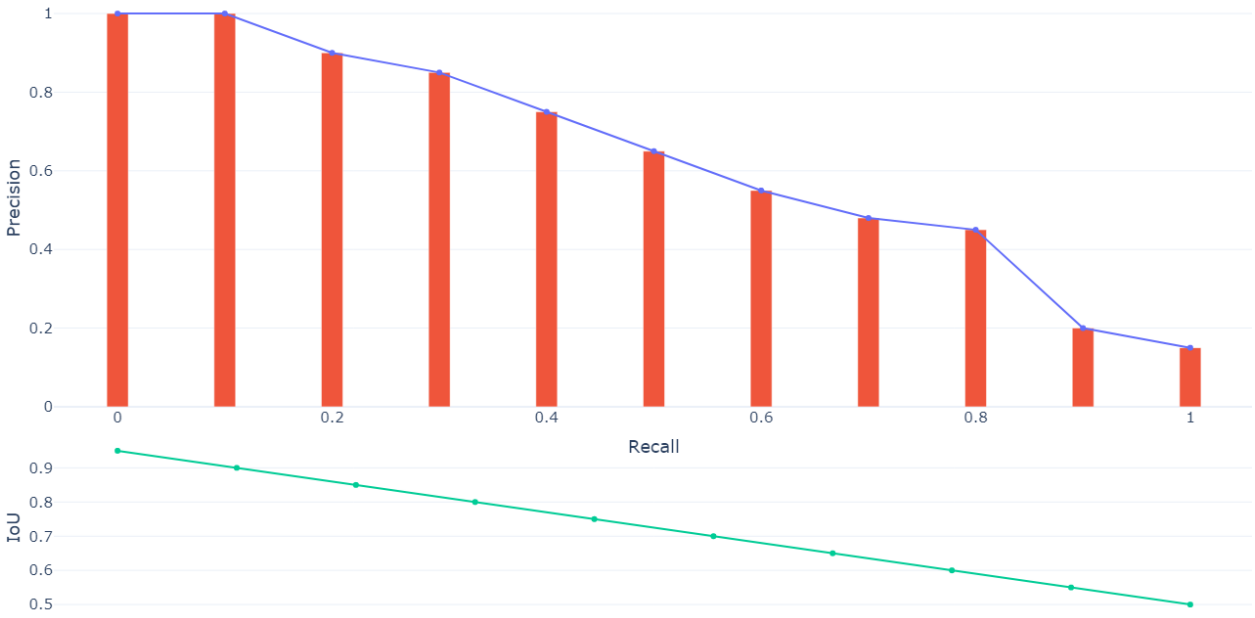


Abbildung 7 Precision-Recall Curve IoU .5:0.95

Um aus den AP der einzelnen Klassen den mAP zu berechnen, wird der Mittelwert aller Klassen berechnet. Um verschiedene Models miteinander zu vergleichen, muss die Validierung auf dem selben Datenset geschehen. Sehr prominent ist das COCO Dataset von Microsoft [4].

2.1.5 Voreingenommenheit von ML Models

Ein Model lernt nur Informationen, die im Datenset vorkommen. In jedem Datenset gibt es einen gewissen Bias. Wenn in einem Datenset mit Bildern von Bestäubern eine gewisse Art immer auf einer bestimmten Blume sitzt, lernt das Model diesen Zusammenhang. Neben dem Hintergrund der Objekte spielen auch Grösse und Schärfe eine Rolle. Die einfachste Methode, um die Voreingenommenheit zu minimieren ist das Überarbeiten und ergänzen des Datensets. Falls es nicht möglich ist, diversere Bilder zu erheben, kann mit Data Augmentation nachgeholfen werden.

2.1.6 Data Augmentation

Data Augmentation ist eine Technik zur künstlichen Vergrösserung eines Datensets. Damit ein Model zuverlässige Vorhersagen machen kann, braucht es genügend Trainingsdaten, die nicht immer vorhanden sind. Mit unterschiedlichen Methoden wird die Diversität des Datensets vergrössert, womit das Model robuster wird. Data Augmentation kann auf unterschiedliche Datenformate angewandt werden. Sehr häufig wird die aber an Bilddaten angewandt. Folgend werden einige Transformationen an einem Bild, welches in Abbildung 8 ersichtlich ist, gezeigt.

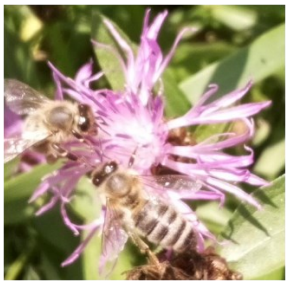


Abbildung 8 Data Augmentation: Originalbild

Flipping & Rotation

Beim Flipping werden die Bilder an einer Achse gespiegelt. Bei der Rotation werden die Bilder um einen bestimmten Winkel rotiert. Falls in einem Datenset ein Objekt immer in der selben Position vorhanden ist, wird das Model somit robuster für die Erkennung von Objekten, die in einer anderen Position vorhanden sind. Abbildung 9 zeigt Beispiele für Flipping und Rotation.

Abbildung 9 Data Augmentation: Flip and Rotate

Scale & Crop

Damit das Model abgeschnittene Objekte erkennen kann, wird Scale and Crop angewendet. Bestimmte Regionen aus Bildern werden skaliert und ausgeschnitten. Beispiele sind in Abbildung 10 visualisiert.

Abbildung 10 Data Augmentation: Scale and Crop

Translation

Bei der Translation werden Regionen aus einem Bild ausgeschnitten und räumlich versetzt. Wenn die Objekte im Datenset immer zentriert in den Bildern sind, wird das Model die Regionen am Rand weniger beachten. Durch Translation wird dieser Bias im Datenset reduziert. Abbildung 11 zeigt verschiedene Translationen des Bildes.

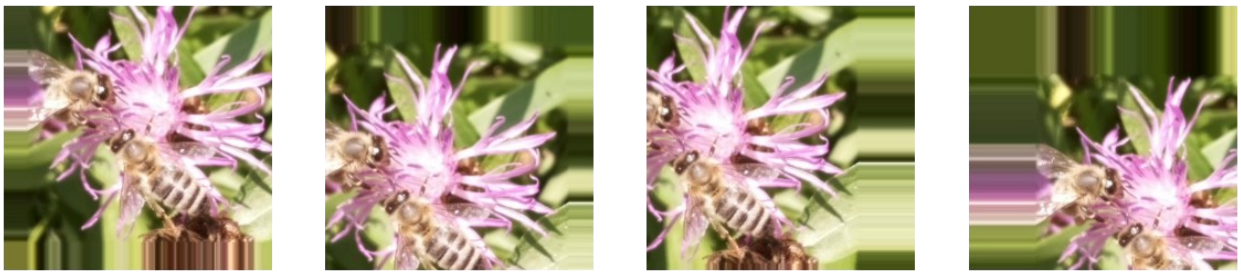


Abbildung 11 Data Augmentation: Translation

Noise

Bilder können Rauschen (Noise) beinhalten. Um auf Bilder mit Qualitätseinbussen vorbereitet zu sein, kann Noise künstlich hinzugefügt werden. Verschiedene Arten und Intensitäten von hinzugefügter Noise sind in Abbildung 12 ersichtlich.

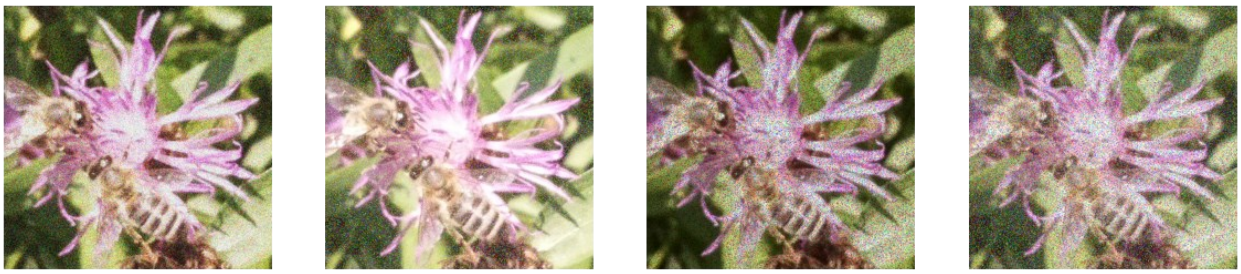


Abbildung 12 Data Augmentation: Noise

Helligkeit, Kontrast, Sättigung, Farbton

Oft sind die Bilder im Datenset unter optimalen Bedingungen aufgenommen worden. Wenn beim Einsatz des Models ein Objekt sehr stark beleuchtet wird oder durch einen automatischen Helligkeitsabgleich sehr dunkel wird, hat es Probleme mit der Erkennung. Durch die Generierung von Trainingsdaten mit modifizierten Helligkeiten, Kontrasten, Sättigungen und Farbtönen kann die Robustheit erhöht werden. Abbildung 13 zeigt Bilder mit möglichen Modifizierungen.

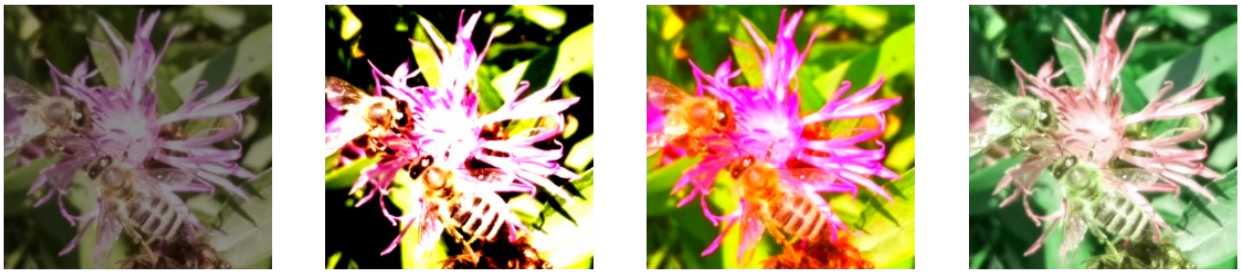


Abbildung 13 Data Augmentation: Brightness, Contrast, Saturation, Hue

Copy-Paste

Eine fortgeschrittene Methode der Data Augmentation ist Copy-Paste. Zu detektierende Objekte werden kopiert und in andere Bilder eingefügt. Somit ist es möglich, den Hintergrund Bias zu minimieren und die Voreingenommenheit des Models einzuschränken. Abbildung 14 zeigt die Anwendung der Copy-Paste Augmentation. Sie wird vermehrt bei komplexen Models eingesetzt, die auf grossen Bildern trainiert werden.

Abbildung 14 Data Augmentation: Copy Paste [5]

2.2 Convolutional Operations

Ein digitales Bild ist eine mehrdimensionale Matrix, worin pro Pixel und Farbkanal je einen Wert aufgeführt wird. Durch mathematische Operationen können die Werte der Matrix bearbeitet werden, um Merkmale hervorzuheben oder um Rauschen zu unterdrücken. Diese Operationen bilden die Grundlage für eine automatisierte Erkennung von Objekten auf einem Bild.

Für einfache Transformationen werden sogenannte Faltoperationen durchgeführt. Mit einer Matrize aus Gewichten, auch als Kernel oder Filter benannt, wird für jedes Pixel eines Bildes einen neuen Wert berechnet und ein neues Bild generiert. Abbildung 15 zeigt ein Beispiel eines Graustufenbildes, welches mit einem 3x3 Kernel geglättet wird. Der Kernel berechnet den Durchschnittswert der umliegenden Pixel. Der Kernel wird schrittweise, von oben Links nach unten Rechts über die Pixel im Originalbild gelegt. Für jede Position des Kernels werden die Werte des Bildes mit dem entsprechenden Wert im Kernel multipliziert und anschliessend addiert. So wird der Pixel (11,2) auf 124 gesetzt:

$$P(11,2) = (90 + 88 + 92 + 130 + 93 + 92 + 231 + 186 + 114) * \frac{1}{9} = 124$$

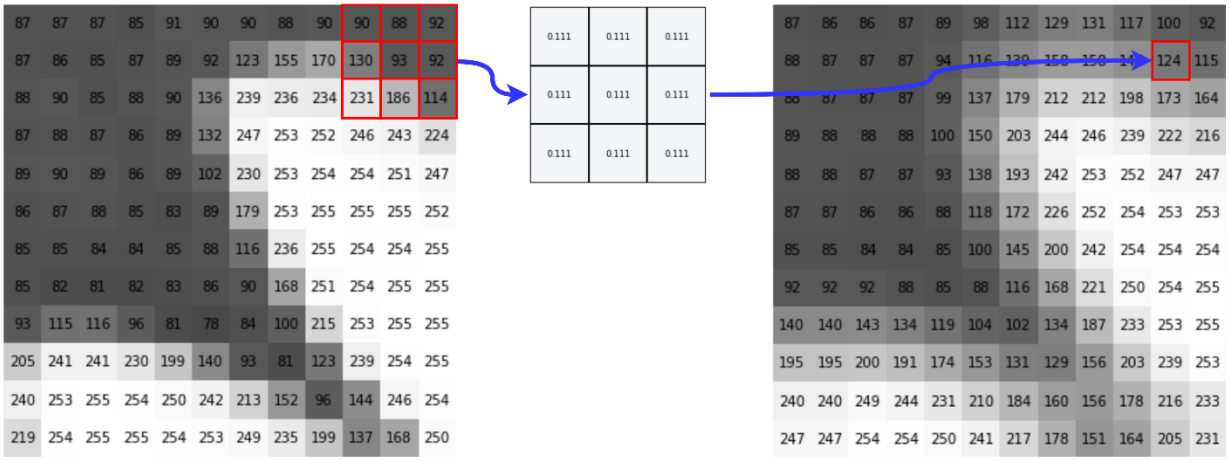


Abbildung 15 Glätten eines Bilder mit 3x3 Kernel

Für die Durchführung von Kerneloperationen sind neben der Form und Grösse des Kernels weitere Parameter zu spezifizieren.

Der Padding bezeichnet einen Pixelrahmen, welcher rund um das Bild ergänzt werden kann. Werden keine zusätzlichen Pixel ergänzt (Valid Padding), werden die Dimensionen des Ausgabebildes verringert, da die Pixel am Rand nicht berechnet werden können. Ein einfacher Ansatz für das Beibehalten ist das Hinzufügen eines Rahmens, bestehend aus Pixel mit dem Wert 0 (Same Padding). Abbildung 16 visualisiert die Auswirkungen auf die Ausgangsdimension.

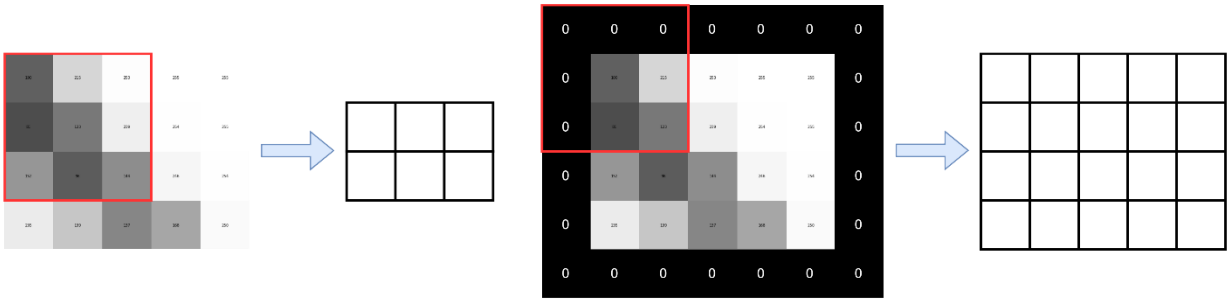


Abbildung 16 Kernel Operationen: Valid Padding (l), Same Padding (r)

Bei einer Transformation mit hinzugefügtem Padding ist auch eine Nearest Neighbor Interpolation möglich, womit die ergänzten Pixel auf den Wert der nächstgelegenen Pixel gesetzt werden.

Die Schrittweite (Stride) bestimmt die Anzahl Pixel, die der Kernel nach jedem berechneten Pixel in x oder y Richtung vorrückt. In den meisten Fällen liegt dieser Wert bei eins. Bei höheren Stride Werten werden die Dimensionen des Outputs entsprechend verringert. Abbildung 17 visualisiert das Vorrücken des Kernels mit einem Stride von eins (Rot) und mit einem Stride von zwei (Grün).

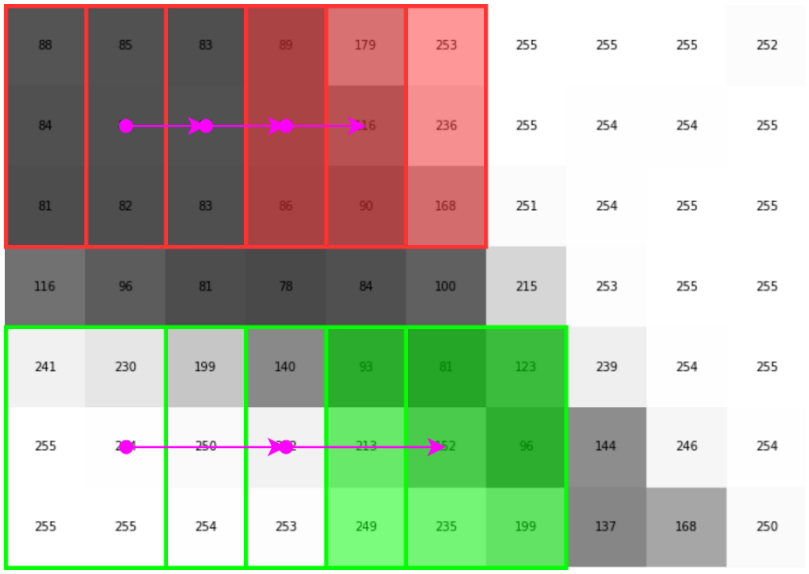


Abbildung 17 Kernel Operationen: Stride

Nachfolgend sind häufig eingesetzte Kernel Operationen demonstriert, die auf ein Graustufenbild einer Margarite in Abbildung 18 angewendet werden. Die Grösse der Kernel ist variabel. In den Beispielen wird jeweils ein 3x3 Kernel eingesetzt.

Abbildung 18 Graustufenbild einer Margarite

Um Bilder zu glätten, wird ein Blur Kernel eingesetzt, der auch im Beispiel in Abbildung 15 ersichtlich ist. Das Ziel der Glättung eines Bildes ist die Eliminierung von Rauschen, also das Entfernen von einzelnen störenden Pixeln, die vernachlässigbar sind. Das geglättete Bild ist in Abbildung 19 ersichtlich.

Blur

0.111	0.111	0.111
0.111	0.111	0.111
0.111	0.111	0.111

Abbildung 19 Normalized Blur Kernel

Ein Kernel zur Erhöhung der Bildschärfe ist in Abbildung 20 ersichtlich. Der Kernel gewichtet die horizontalen und die vertikalen Übergänge hoch, was in einer Hervorhebung der Kanten und Umrisse resultiert.

Sharpen

0	-1	0
-1	5	-1
0	-1	0

Abbildung 20 Sharpen Kernel

Mit dem Outline Kernel, der in Abbildung 21 ersichtlich ist, werden nur Regionen des Bildes, die sehr starke Übergänge haben, also Kanten und Umrisse, hervorgehoben. Gleichbleibende Flächen, wie etwa die Blütenblätter, rücken in den Hintergrund.

Outline

-1	-1	-1
-1	8	-1
-1	-1	-1

Abbildung 21 Outline Kernel

Der Laplace Kernel, welcher in Abbildung 22 ersichtlich ist, funktioniert sehr ähnlich wie der Outline Kernel. Er wird für die Detektion von horizontalen und vertikalen Kanten eingesetzt. In der Praxis ist er auch häufig mit negierten Werten anzutreffen.

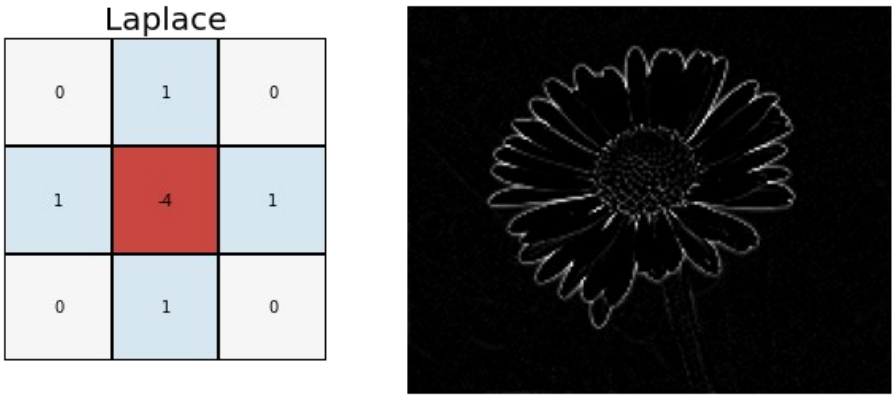


Abbildung 22 Laplace Kernel

Für die Erkennung von Unterschieden in benachbarten Pixelwerten in eine definierte Richtung werden Sobel Kernel eingesetzt. Mit dem Top Sobel Kernel, der in Abbildung 23 erkennbar ist, werden hell zu dunkel Übergänge von oben nach unten im Bild hervorgehoben. Übergänge von dunkel zu hell werden nicht beachtet. Abbildung 24 zeigt den Left Sobel Kernel, der Übergänge von hell zu dunkel von links nach rechts hervorhebt.

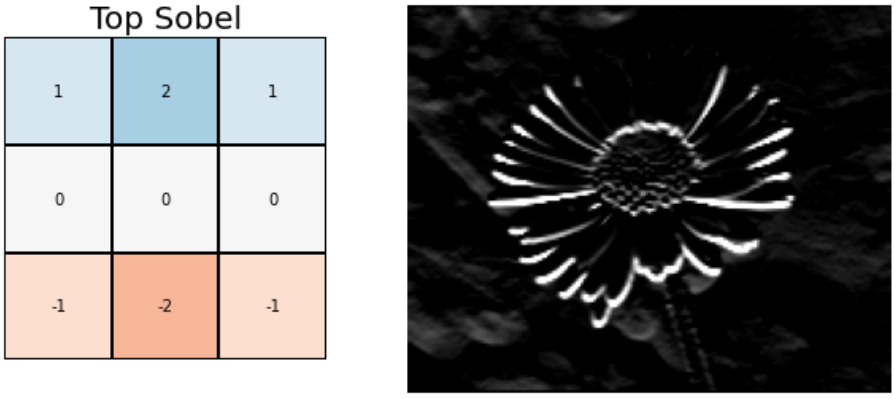


Abbildung 23 Top Sobel Kernel

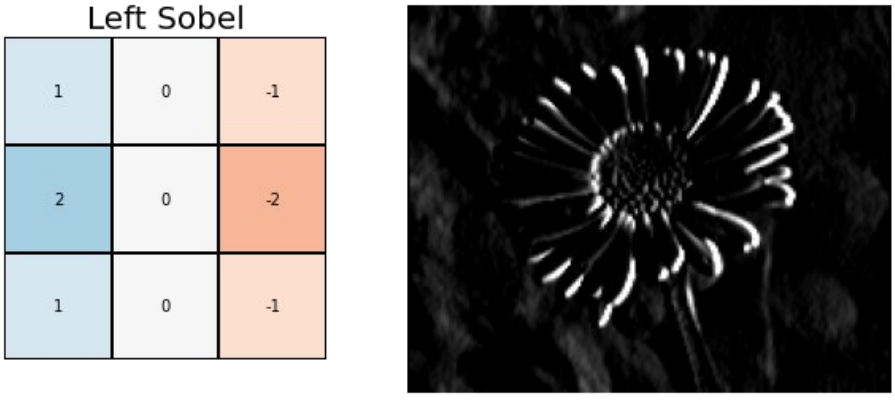


Abbildung 24 Left Sobel Kernel

Neben der Transformation von Bildern können Faltoperationen Merkmale aus Audiodateien extrahieren. Die bilden auch die Grundlage für die Erkennung von Geräuschen wie Rufe von Vögeln in Audioaufnahmen.

2.3 Convolutional Neural Network (CNN, ConvNet)

Ein CNN ist ein künstliches neuronales Netz, das auf das Detektieren von Mustern in Daten wie Bildern oder Audiodaten spezialisiert ist. Die Herausforderung dieser Daten liegt in der grossen Menge an Informationen. Ein RGB Bild mit einer Auflösung von 3280 auf 2464 Pixel besteht aus 24'245'760 einzelnen Werten. Die Aufgabe eines CNN ist es, die Eingangsdaten in eine Form zu reduzieren, die einfach zu bearbeiten ist und trotzdem alle relevanten Informationen beinhaltet.

Ein CNN besteht aus mehreren Schichten (Layer), in welchen Faltoperationen durchgeführt werden. Im Gegensatz zu herkömmlichen Pipelines der Bildverarbeitung wird jede Operation mehrfach mit unterschiedlichen Gewichten parallel berechnet. Die einzelnen Berechnungen in einer Schicht werden als Neuronen bezeichnet. Jedes Neuron verarbeitet die Summe aller Resultate der Neuronen aus der vorherigen Schicht.

Abbildung 25 zeigt den groben Aufbau eines minimalen CNN anhand eines Beispiels [6]. Die einzelnen Schritte werden in folgenden Unterkapiteln dokumentiert.

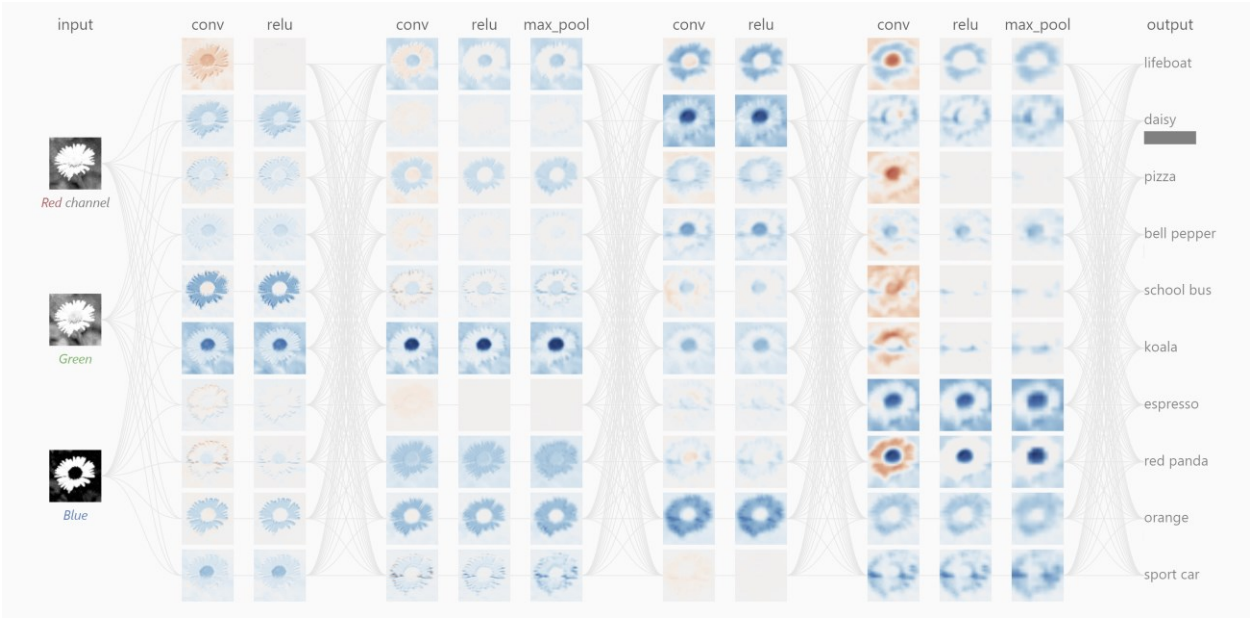


Abbildung 25 Aufbau CNN [6]

2.3.1 Input

Als Eingangsdaten für ein CNN für Image Classification sind Bilder. Die Form eines Bildes ist Breite x Höhe x Anzahl Channels. Bei klassischen RGB Aufnahmen sind drei Channels vorhanden, aus welchen die Farben der jeweiligen Pixeln zusammengesetzt ist. Abbildung 26 zeigt auf der linken Seite ein Farbbild und rechts davon die einzelnen Channels. In CNN werden die Channels unterteilt und separat behandelt [6].

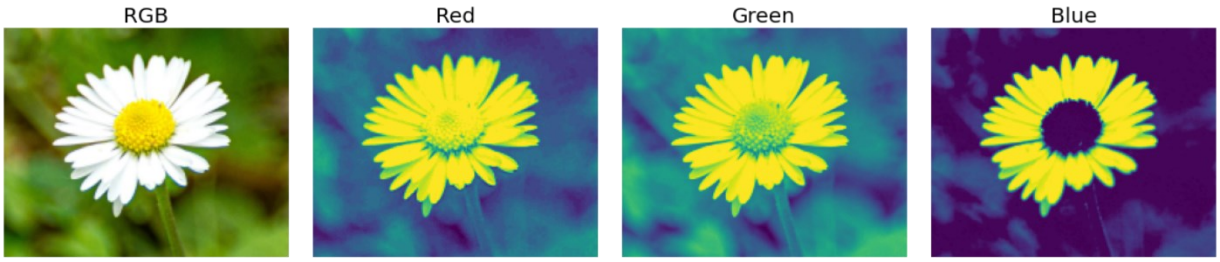


Abbildung 26 RGB Channels eines Bildes

2.3.2 Convolutional Layer (Conv)

Ein Convolutional Layer empfangt Inputdaten, transformiert sie und übergibt sie dem nächsten Layer. Die Transformation ist eine sogenannte Convolutional Operation, also eine Faltoperation, die im Kapitel Convolutional Operations dokumentiert sind. Das Ziel dieser Layer ist es, Formen, Kanten, Texturen und weitere Features zu erkennen. Jeder Kernel hat unterschiedliche Werte, die beim Trainingsvorgang des Netzwerks gelernt werden. Diese Werte werden als Gewichte (Weights) bezeichnet. Abbildung 27 zeigt die Funktion eines Conv Layers mit den gelernten Weights eines 3 x 3 Kernels mit Stride 1 und Valid Padding. Die Ausgangsdimensionen sind um jeweils zwei Pixel verkleinert.

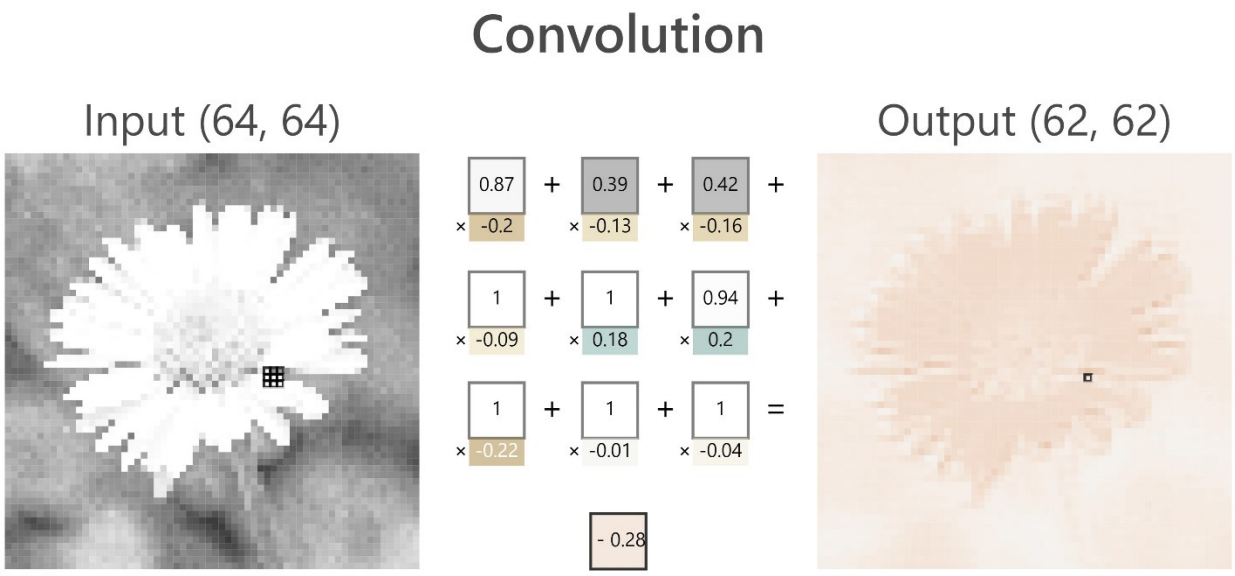


Abbildung 27 CNN - Convolutional Layer

Der erste Conv Layer verarbeitet die drei Farbkanäle als Input. Um die Inputs zusammenzuführen, werden die Resultate der jeweiligen Kernel Operationen addiert. Anschliessend wird ein Bias Wert addiert, der während dem Training gelernt wird. Der Bias Wert ist typischerweise eine Reelle Zahl. Der Vorhang des ersten Conv Layers des Beispielnetzwerks ist in Abbildung 28 visualisiert.

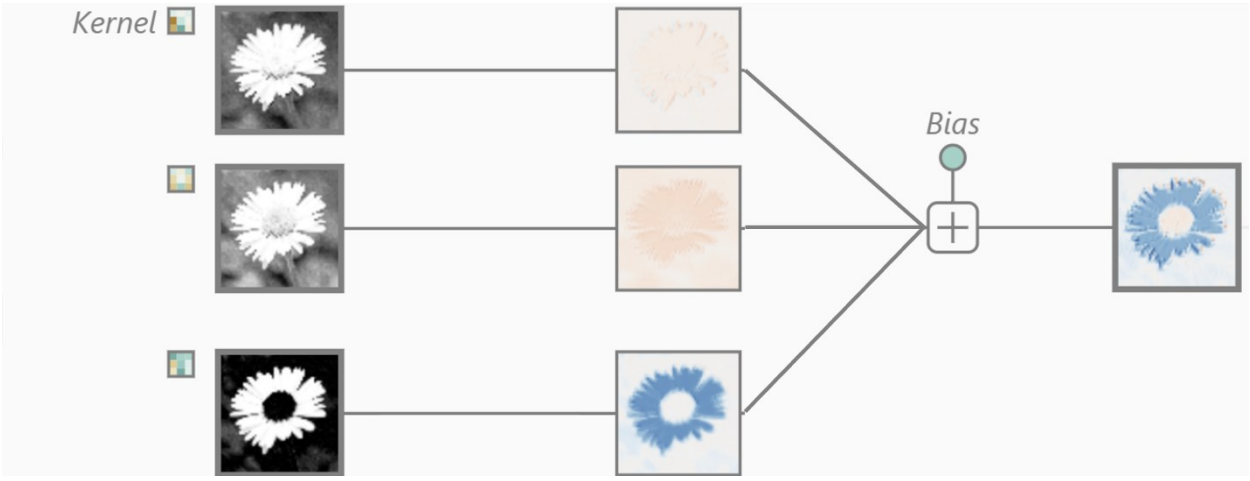


Abbildung 28 CNN: Zusammenführung von Inputs

2.3.3 Activation Layer

Um die Relevanz eines Neurons zu bestimmen, werden am Ende eines CNN oder zwischen einzelnen Layern Activation Layers eingesetzt. Sie filtern schwache und nicht relevante Features heraus, damit sich das Netzwerk «auf das wesentliche konzentriert». Das menschliche Hirn hat einen ähnlichen Mechanismus zum Priorisieren bzw. Weiterleiten von aufgenommenen Informationen. Wenn eine Sirene ertönt, werden die entsprechenden Neuronen aktiviert und man bemerkt es sofort. Normale Hintergrundgeräusche werden herausgefiltert, da sie nicht relevant sind.

Die einfachste Activation Function ist ReLU, Rectified Linear Unit. Alle negativen Werte werden auf 0 gesetzt, alle positiven werden beibehalten. Ein praktisches Beispiel ist in Abbildung 29 visualisiert. Die mathematische Funktion ist in Abbildung 30 ersichtlich.

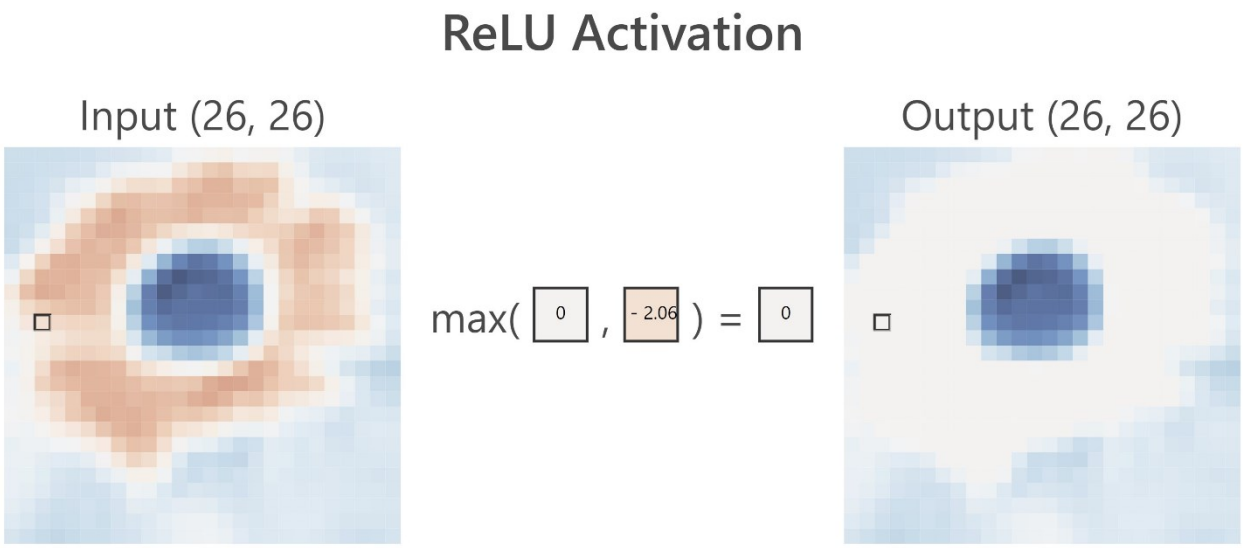


Abbildung 29 CNN: ReLU

Neben der Standard ReLU Funktion gibt es weitere Variationen, die negative Werte nicht komplett ignorieren. Abbildung 30 zeigt im mittleren Chart Leaky ReLU, wobei negative Werte linear transformiert werden. Rechts im Bild ist Shifted ReLU abgebildet. Die Grenze wird leicht unterhalb von 0 gebildet [7].

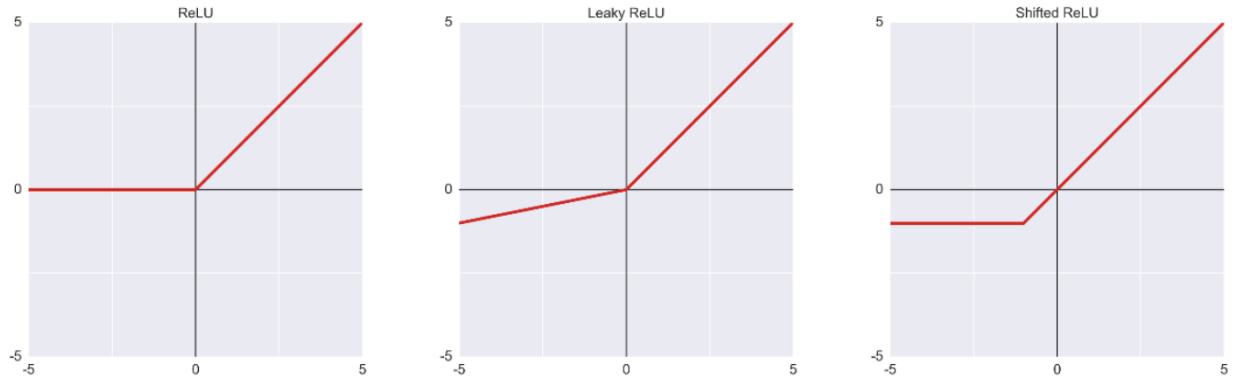


Abbildung 30 CNN: Activation, ReLU Variationen [7]

Weitere nichtlineare Aktivierungsfunktionen sind in Abbildung 31 visualisiert. Eine neben ReLU weit verbreitete Activation Funktion ist Sigmond, wobei die Ausgangswerte zwischen 0 und 1 liegen.

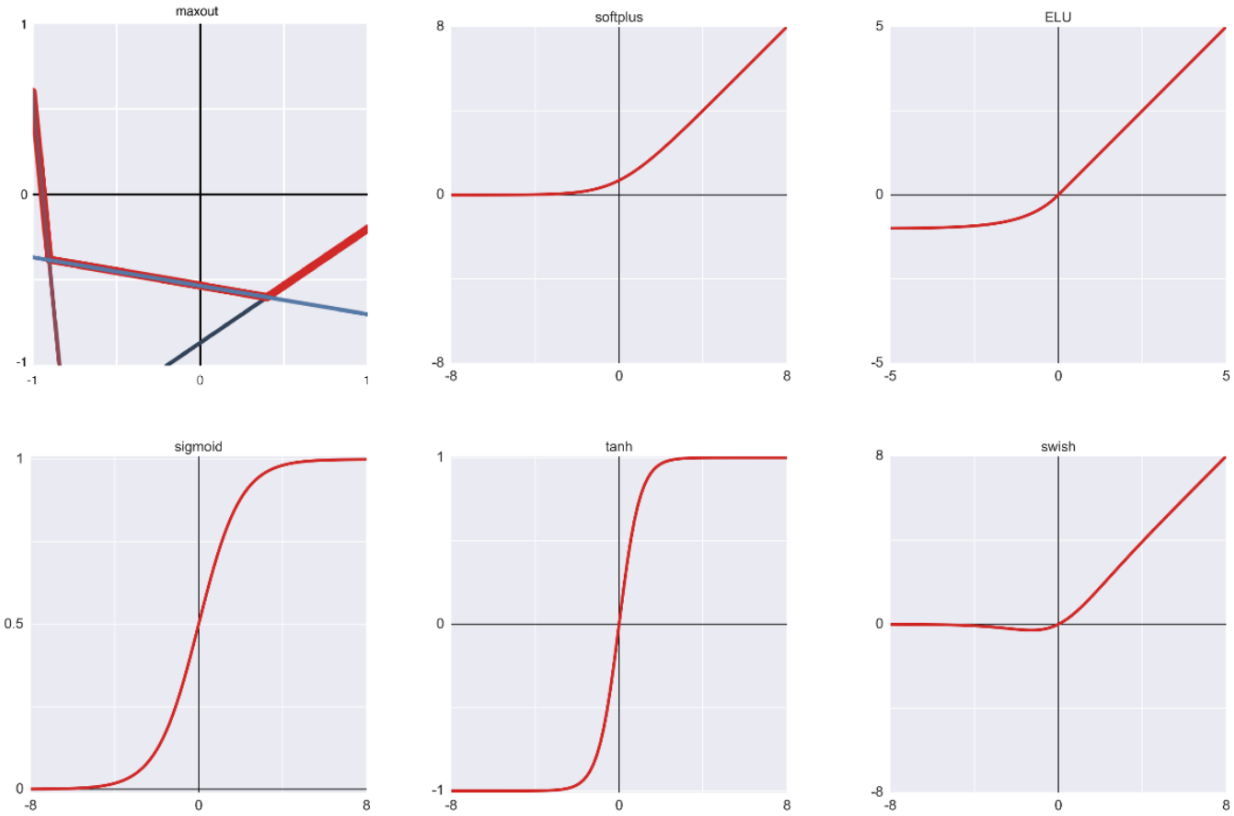


Abbildung 31 CNN: Aktivierungsfunktionen [7]

2.3.4 Pooling Layer

In Pooling Layern werden Überflüssige Informationen verworfen. Die mit Abstand am stärksten verbreitete Methode ist Max Pooling. Aus einem definierten Bereich wird jeweils nur das stärkste Feature, das Pixel mit dem höchsten Wert, behalten. Somit wird die Gesamtgrösse des Netzwerks klein gehalten und die Berechnungsdauer minimiert. Ähnlich wie bei Faltoperationen wird eine Kernelgrösse und die Stride gewählt. Mit einem 2x2 Kernel und einem Stride von zwei wird die Bildgrösse um 75 % verkleinert. Abbildung 32 zeigt die Anwendung von Max Pooling mit einem 2 x 2 Kernel an einem 20 x 20 Bild. Das Resultat hat Dimensionen von 10 x 10 Pixel.

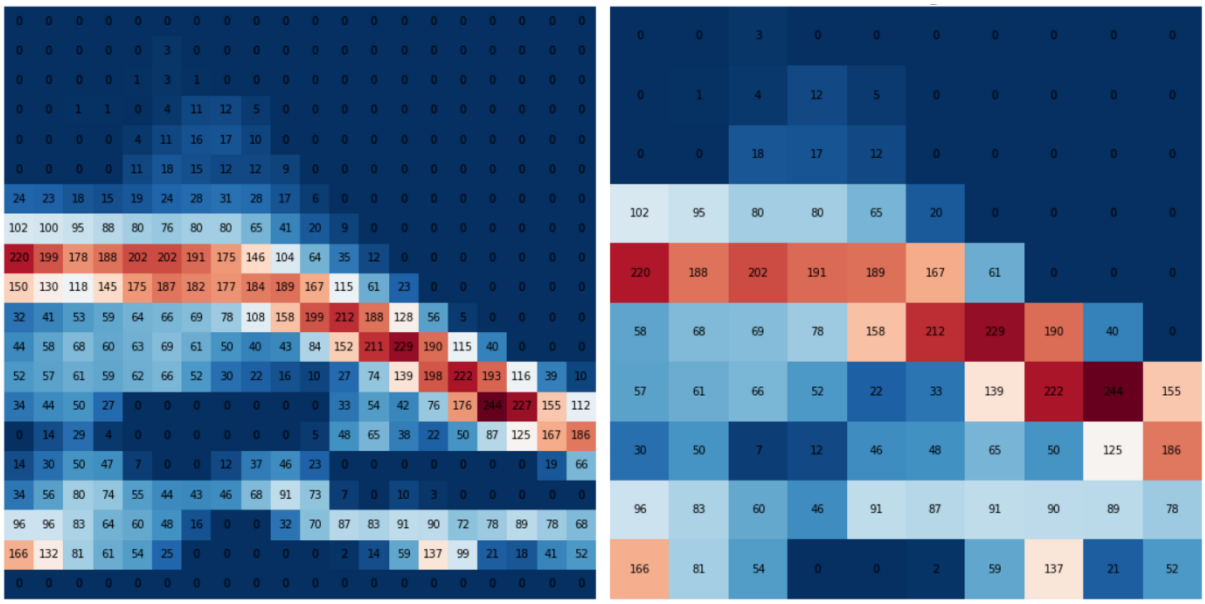


Abbildung 32 CNN: Max Pooling

Neben Max Pooling gibt es Average Pooling, wobei jeweils der Durchschnitt der Pixel im ausgewählten Bereich berechnet wird. In der Praxis hat sich Max Pooling jedoch als effektiver bewiesen [7].

2.3.5 Flatten Layer

Der Flatten Layer wandelt einen multidimensionalen Vektor in einen eindimensionalen Vektor, einen sogenannten Feature Vektor, um. Abbildung 33 zeigt die Anwendung eines Flatten Layers auf einen 2 x 2 x 2 Input. Der Output ist ein Vektor mit länge 8. Für jede Klasse wird jedes Element des Feature Vektors mit dem entsprechenden gelernten Weight multipliziert. Anschliessend wird die Summe der Produkte berechnet und ein Bias Wert addiert.

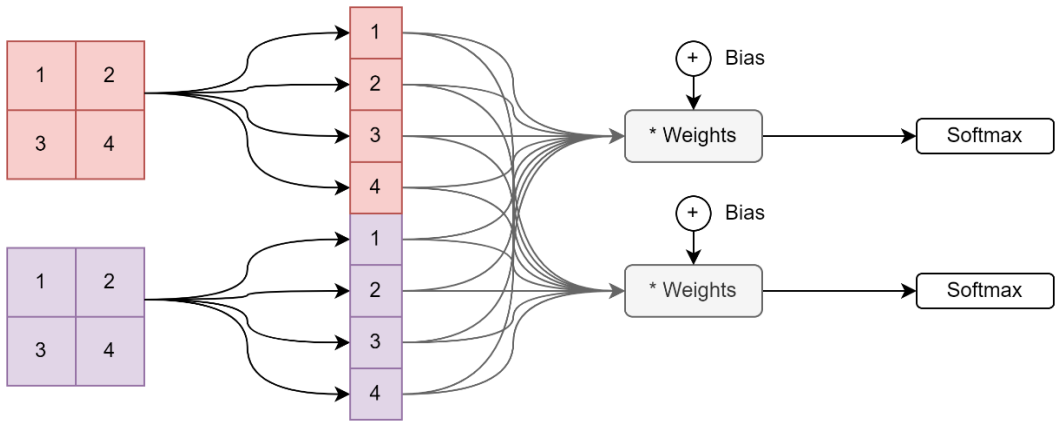


Abbildung 33 CNN: Flatten Layer

2.3.6 Softmax

Die Softmax Funktion hat die Schlüsselfunktion sicherzustellen, dass die Summe aller CNN Outputs 1 ergibt und somit eine Wahrscheinlichkeit pro Klasse berechnet werden kann. Folgend ist die Formel ersichtlich.

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_i \exp(x_j)}$$

Im Gegensatz zu einer Standardnormalisierung bezieht der Softmax neben den Proportionen der Werte auch die effektiven Werten mit ein. Abbildung 34 zeigt einen Vergleich von Standardnormalisierung und Softmax. Die Inputdaten sind ein Array, bestehend aus 9 Werten: [1, 2, 3, 4, 5, 6, 7, 8, 9]. Der grösste Wert, 9, liegt bei der Standardnormalisierung mit 0.2 nur ganz knapp vor der Probability des Werts 8. Softmax wertet diesen Unterschied so hoch, dass die Probability für den Wert 9 mehr als doppelt so hoch ist wie für den Wert 8. Somit kann mit der Softmax Funktion auch für kleine Unterschiede eine klare Unterscheidung berechnet werden.

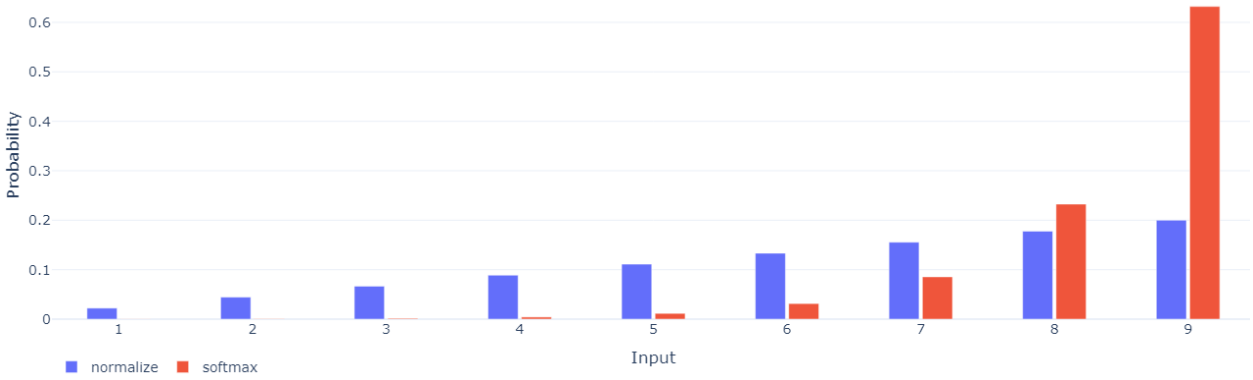


Abbildung 34 Vergleich zwischen Softmax und Standardnormalisierung

2.4 Deep Learning für die Auswertung von Bilddaten

Die Popularität von Deep Learning hat in den letzten Jahren stark zugenommen. Insbesondere für die Extrahierung von Informationen aus Bilddaten wurden grosse technologische Fortschritte gemacht. Die Auswertung von Bilddaten mit Deep Learning wird in Kategorien unterteilt, wovon die Wichtigsten nachfolgend dokumentiert sind.

2.4.1 Image Classification

Bei der Image Classification wird eine Aussage über den Inhalt eines Bildes gemacht, beispielsweise ob es einen Hund oder eine Katze zeigt. Der Output des Models sind die Wahrscheinlichkeiten pro Klasse. Abbildung 35 visualisiert den Vorgang.

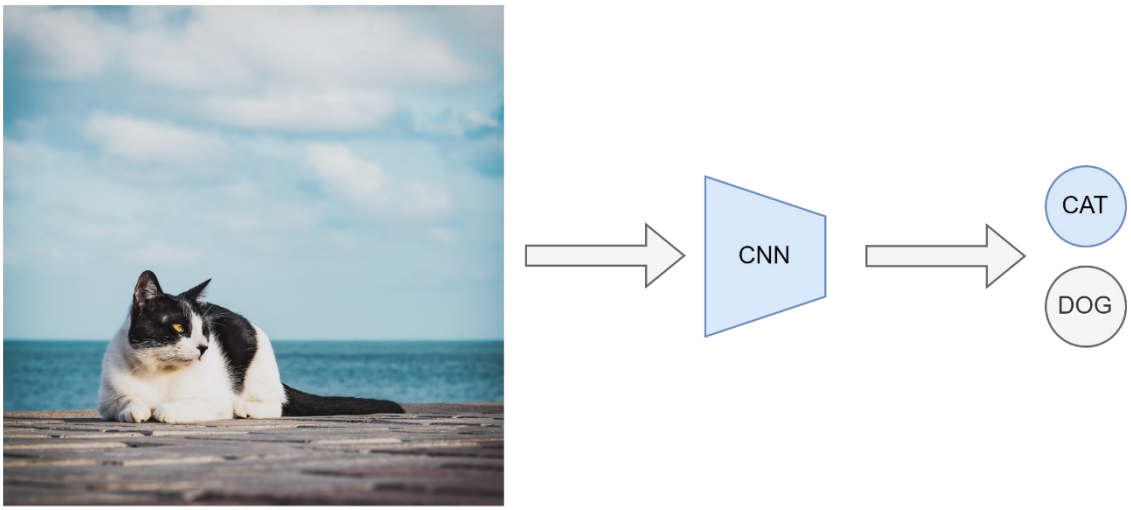


Abbildung 35 Image Classification [8]

2.4.2 Object Localization

Object Localization wird eingesetzt, um ein einzelnes, prominentes Objekt auf einem Bild zu lokalisiert. Der Output eines Object Localization Models ist eine Bounding Box, die aus mindestens vier Punkten besteht. Gängige Formate sind die Eckpunkte oben links (x1, y1) und unten rechts (x2, y2) oder der Mittelpunkt (x,y) mit Breite und Höhe der Objekts (w, h). Zusätzlich zu der Region beinhaltet das Resultat der Auswertung die Wahrscheinlichkeit, mit welcher das Objekt detektiert wurde. Abbildung 36 visualisiert die Funktionsweise.

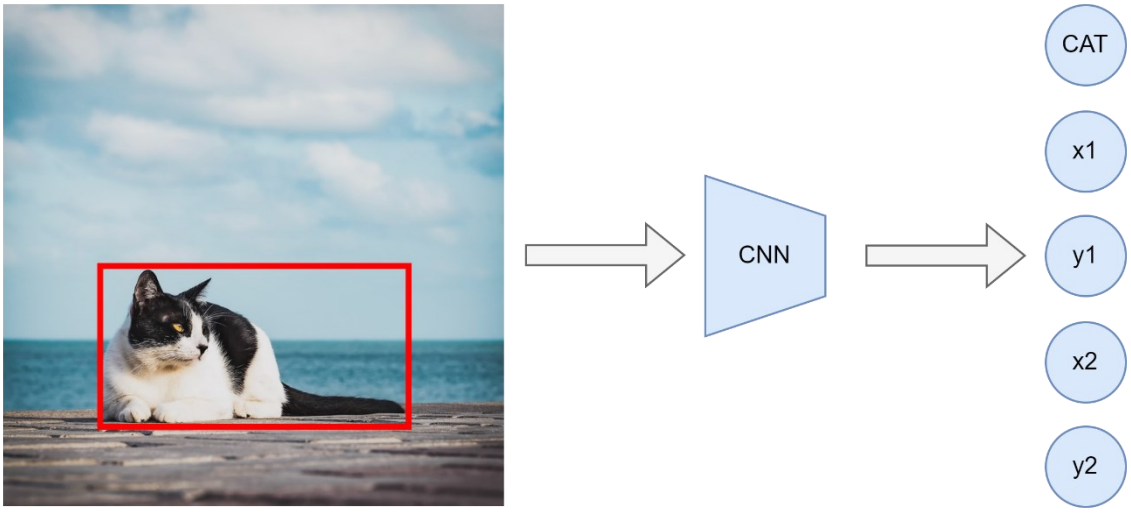


Abbildung 36 Object Localization [8]

2.4.3 Object Detection

Object Detection wird eingesetzt, um mehrere Objekte mit unterschiedlichen Klassen auf einem Bild zu erkennen. Pro erkanntes Objekt beinhaltet der Output die Klasse, die Wahrscheinlichkeit und eine Bounding Box der Position auf dem Bild. Abbildung 37 visualisiert die Schritte einer Objekterkennung. Im folgenden Unterkapitel wird das Thema Object Detection vertieft dokumentiert.

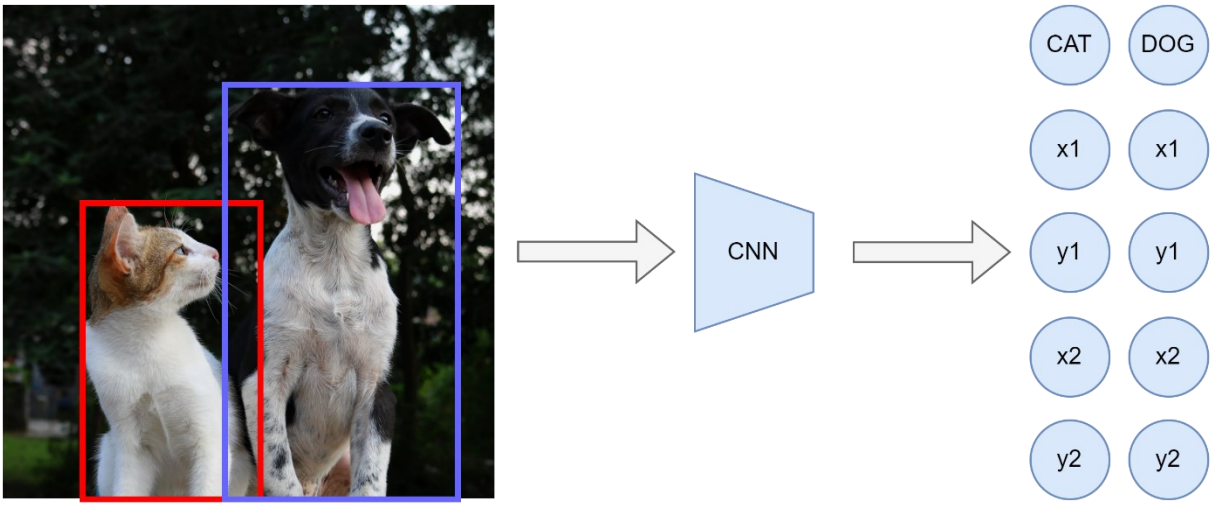


Abbildung 37 Object Detection [9]

2.5 Erkennung von Objekten auf Bildern

In den letzten 30 Jahren wurde sehr viel an der Herausforderung, Objekte auf einem Bild zu erkennen, geforscht. Mit vielen neuen Entwicklungen wurden bisherige Technologien in den Schatten gestellt. Folgend werden die wichtigsten Ansätze dokumentiert.

2.5.1 Sliding Windows

Einer der ersten Ansätze für die Erkennung von Objekten ist Sliding Windows. In kleinen Schritten werden quadratische Bereiche des Bildes ausgeschnitten, wie es in Abbildung 38 visualisiert ist. Mit einem CNN wird jeder ausgeschnittene Bereich analysiert, ob sich ein zu detektierendes Objekt darin befindet. Um Objekte mit unterschiedlichen Grössen zu detektieren, wird der Vorgang mit verschiedenen Window Dimensionen repetiert. Je mehr Durchläufe mit unterschiedlichen Window Grössen und je kleiner die Schrittweite, desto genauer wird das Resultat [10].

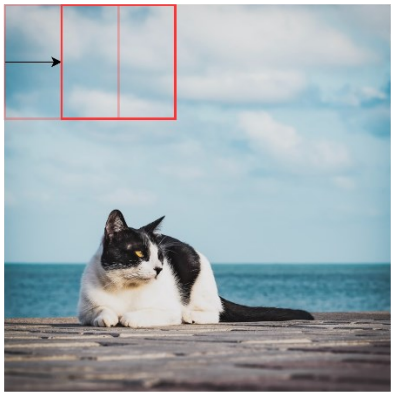


Abbildung 38 Sliding Windows [8]

Der Vorteil des Sliding Windows Ansatzes ist die sehr einfache Implementierung. Der grösste Nachteil ist der sehr hohe Berechnungsaufwand. Der Berechnungsaufwand kann auf Kosten der Genauigkeit reduziert werden, indem die Schrittweite vergrössert wird oder die Anzahl der unterschiedlichen Window Dimensionen verkleinert wird. Durch die quadratische Form der Ausschnitte wird das erkannte Objekt mit einer quadratischen Form markiert, auch wenn es rechteckig ist.

2.5.2 R-CNN

Ein Ansatz, um den hohen Berechnungsaufwand bei dem Sliding Windows Verfahren zu verringern, ist der Einsatz von Regional Based Networks, R-CNN. Mit einem Algorithmus, der im originalen Paper noch kein CNN war, werden rund 2000 mögliche Regionen markiert, in welchen sich möglicherweise ein Objekt befindet [11]. Durch die Vielzahl an Vorschlägen ist die Wahrscheinlichkeit sehr hoch, dass sich das Objekt markiert wird. Jeder Vorschlag wird in eine standardisierte Grösse transformiert und mit einem CNN klassifiziert. Abbildung 39 zeigt das Konzept von R-CNN.

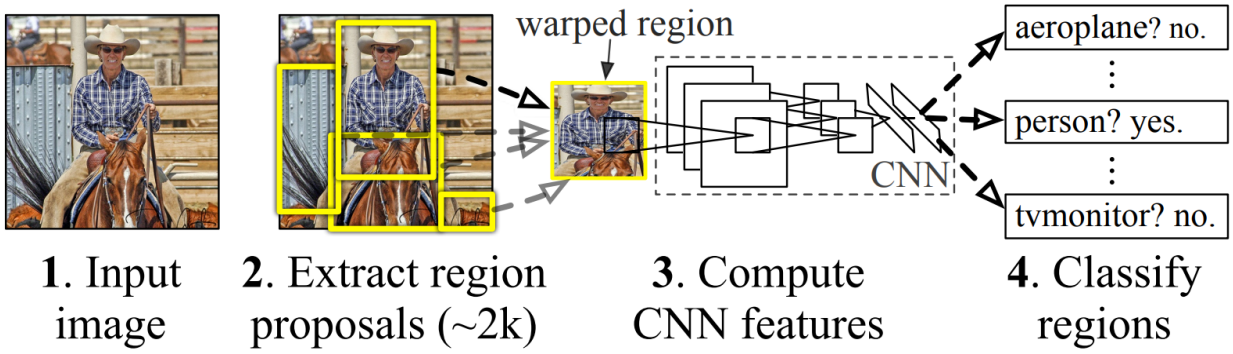


Abbildung 39 R-CNN Konzept [11]

Mit R-CNN beschränkt sich der Berechnungsaufwand auf die Klassifizierung der definierten Anzahl vorgeschlagener Bereiche. Trotzdem ist R-CNN sehr langsam und kann nicht in Echtzeitanwendungen eingesetzt werden. Die detektierten Objekte können mehrere Formen haben und müssen nicht quadratisch sein, was die Qualität der Resultate erhöht.

2.5.3 Fast(er) R-CNN

Um die Effizienz von R-CNN zu erhöhen, hat der Autor des R-CNN Papers [11] den Algorithmus weiterentwickelt und als Fast R-CNN publiziert [12]. Die hauptsächliche Anpassung ist der Ersatz des deterministischen Region Proposal Algorithmus durch ein CNN, wie es in Abbildung 40 visualisiert ist. Mögliche Regionen werden mit Selective Search auf einer Feature Map aus dem ersten CNN gesucht, womit die Anzahl der Vorschläge viel geringer ausfällt als bei R-CNN. Sowohl die Inferenzzeit als auch die Trainingszeit wurden um Faktor 10 verkleinert [13].

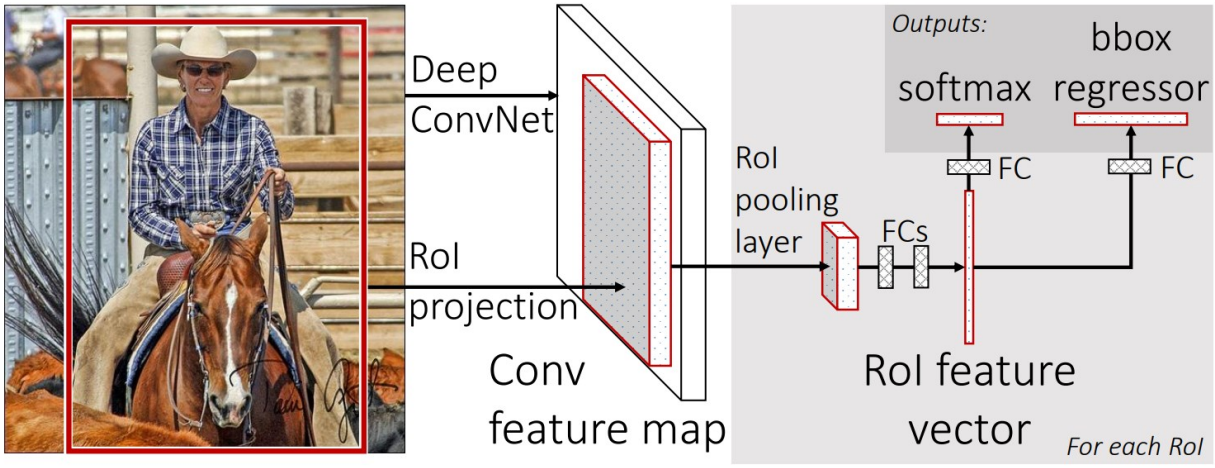


Abbildung 40 Fast R-CNN Konzept [12]

Der zeitaufwändigste Schritt bei Fast R-CNN ist die Suche nach möglichen Regionen mit Selective Search. In Faster R-CNN, der dritten publizierten R-CNN Version [14], wurde der Selective Search Schritt durch ein separates Netzwerk, ein Region Proposal Network (RPN) ersetzt. Durch diese Optimierung wurde die Geschwindigkeit gegenüber von Fast R-CNN noch einmal um den Faktor 10 erhöht [13].

2.5.4 YOLO (You Only Look Once)

Im Jahr 2015 wurde das Paper der ersten YOLO Version veröffentlicht [15]. Im Gegensatz zu den vorhandenen Methoden zur Objekterkennung wird nur ein einziges neuronales Netzwerk eingesetzt, welches die Boxen und Wahrscheinlichkeiten an einem gesamten Bild in einer Evaluation vorschlägt. Durch die Zusammenfassung aller Schritte in ein Netzwerk ist YOLO mit bis zu 150 FPS extrem schnell und somit für Echtzeitanwendungen geeignet.

Abbildung 41 zeigt das Konzept von YOLO. Über das Bild wird ein Raster der Grössen *S* x *S* gelegt. Aus den Features des gesamten Bildes werden Bounding Boxes vorgeschlagen, die nicht nach Klasse unterscheiden. Jede Bounding Box hat den Mittelpunkt in einer Zelle des Rasters. Somit werden für jede Zelle im Raster *B* Bounding Boxen der Form [x, y, w, h, confidence] und *C* Class Probabilities vorgeschlagen. Der Output hat folgende Form $S \times S \times (B * 5 + C)$. Mit einem 7 x 7 Raster mit vier Klassen und zwei Bounding Boxen ist die resultierende Form des Prediction Tensors $7 \times 7 \times 14$.

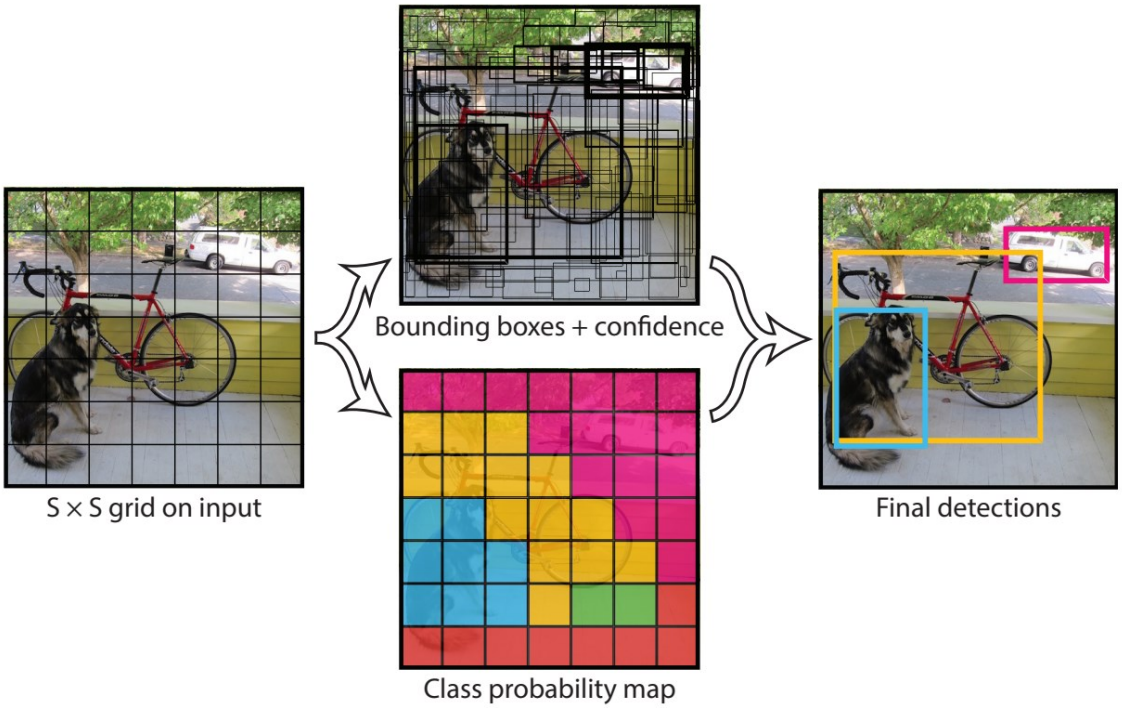


Abbildung 41 YOLO Object Detection Konzept [15]

Neben der sehr hohen Geschwindigkeit hat YOLO den Vorteil, generelle Repräsentationen von Objekten zu erkennen, da nicht wie bei R-CNN Variationen, das ganze Bild bekannt ist. Somit ist es auch möglich, Personen und Objekte auf abstrakten Gemälden zu erkennen, obwohl keine detaillierten Features sichtbar sind. Im Vergleich zu anderen Algorithmen macht YOLO mehr Lokalisierungsfehler und ist tendenziell etwas ungenauer. Die Wahrscheinlichkeit vom Erkennen von False Positives im Hintergrund ist bei YOLO wesentlich geringer als bei vergleichbaren Models [15].

2.5.4.1 YOLOv2

2016 wurde die zweite Version von YOLO publiziert [16]. Eine relevante Verbesserung ist die Einführung von Anchor Boxes. Anchor Boxen beschreiben die Form und Grösse der zu detektierenden Objekte. Vor dem Training werden die Bounding Boxen in den Trainingsdaten mit k-means Clustering in Gruppen mit verschiedenen Grössen und Formen unterteilt. Somit hat das Model mehr Informationen über die zu suchenden Objekte und hat eine erhöhte Genauigkeit. Eine weitere Verbesserung von YOLOv2 ist Multiscale Training. Damit das Model mit

unterschiedlichen Bilddimensionen arbeiten kann, werden die Bilder während des Trainings regelmässig in einem vordefinierten Bereich hoch- oder herunterskaliert [16].

2.5.4.2 YOLOv3

Die dritte Version von YOLO wurde im Jahr 2018 publiziert [17]. Die wesentlichste Erneuerung war der Ersatz des Backbones durch den komplexeren DarkNet-53 Backbone mit 106 Layern [18]. Das ermöglicht es, auf drei verschiedenen Skalierungsebenen Objekte zu erkennen, womit auch kleine Objekte detektierbar sind. Abbildung 42 zeigt die Architektur eines eingesetzten Feature Pyramid Network (FPN). In jeder Ebene werden relevante Regionen markiert und der Nächsten weitergegeben. Trotz des komplexeren Backbones war YOLOv3 rund drei mal so schnell wie die anderen Models mit vergleichbarer Genauigkeit.

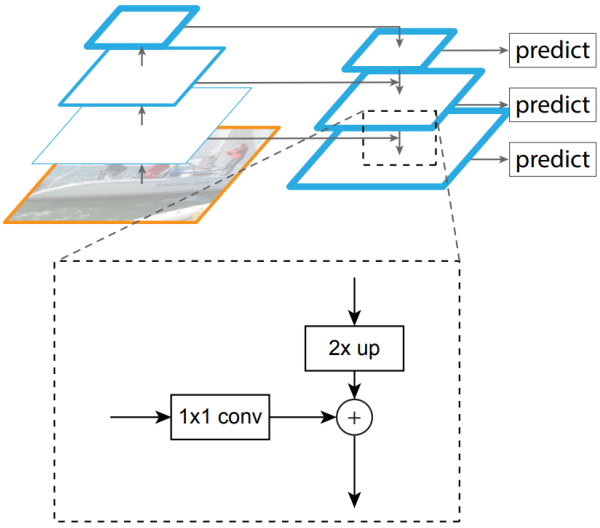


Abbildung 42 Feature Pyramid Network (FPN) [19]

2.5.4.3 YOLOv4

YOLOv3 war die letzte Version, die von Joseph Redmond entwickelt wurde. Im Jahr 2020 wurde eine optimierte Version von YOLOv3 als YOLOv4 publiziert [20]. Die Optimierungen werden im Paper in zwei Kategorien unterteilt.

Die Erste Kategorie umschliesst Technologien, mit welchen die Performance des Models erhöht wird, ohne negative Auswirkungen bei der Inferenz zu erhalten. Entwickelt wurde die automatische Data Augmentation beim Training. Neben trivialen Methoden wurde CutOut eingesetzt, wobei rechteckige Bereiche der Bilder entfernt werden, damit die Objekte auch nur mit einem Teil der Informationen erlernbar sind [21]. CutMix kopiert Bereiche von Bildern in andere Bilder, um weniger Abhängigkeiten vom Hintergrund zu generieren [22]. Mit der Mosaik Methode werden mehrere Bilder aneinandergefügt und als ganzes trainiert, um das Model möglichst robust zu gestalten. Weiter wurden Dropout Methoden zur Regularisierung eingeführt. Während dem Training werden einige gelernte Abhängigkeiten zufällig gelöscht, um Overfitting vorzubeugen.

Die zweite Kategorie bezieht sich auf Technologien zur Erhöhung der Genauigkeit auf Kosten der Effizienz. Darunter fallen die Einführung von Spatial Attention Modules (SAM), wobei räumliche Informationen der Features in den Lernprozess miteinbezogen werden [23]. Das Post Processing wurde, im Gegensatz zu den Vorgänger Versionen, direkt im Model implementiert. Die NMS wird anhand des Distance-IoU Loss [24] berechnet, welcher neben den Überlappungen der Boxen auch die Distanz der Mittelpunkte analysiert. Falls zwei Objekte überlappen, die Mittelpunkte aber eine grosse Distanz haben, werden sie beibehalten.

2.5.4.4 YOLOv5

Die aktuellste YOLO Version ist das Open Source Projekt YOLOv5. Es ist umstritten, ob der Name gerechtfertigt ist, da es sich grundsätzlich um eine PyTorch [25] Implementation von YOLOv3 handelt [26]. Aktuell ist noch kein Paper zu YOLOv5 verfasst worden. PyTorch minimiert den Aufwand für das Training und das Testen von Models. Die automatische Data Augmentation wurde in der Version 5 noch einmal erweitert, womit noch das Model robuster wird. Durch kleinere Anpassungen der Architektur wurden die Inferenzkosten gesenkt, wobei mit YOLOv5 genau und sehr schnell ist. Die enorme Popularität erreichte YOLOv5 aber durch die sehr hohe Benutzerfreundlichkeit und gute Dokumentation.

2.5.5 SSD (Single Shot Detector)

Das Single Shot Detector Model wurde im Jahr 2015, kurz nach dem ersten YOLO Model, publiziert [27]. Das Model besteht aus zwei Komponenten, dem Backbone Model und dem SSD Head. Als Backbone Netzwerk wird ein auf dem ImageNet Datenset [28] vortrainiertes VGG-16 Model [29] ohne den finalen Layer zur Klassifikation eingesetzt, welches leicht modifiziert wurde. Der SSD Head besteht aus einem oder mehreren Convolutional Layern, die zum Backbone hinzugefügt werden. Die Outputs des SSD Heads sind Bounding Boxen, Klassen und Wahrscheinlichkeiten der erkannten Objekte. Der Head funktioniert sehr ähnlich wie YOLO, es wird ein Raster über das Bild gelegt, worin jede Zelle für die sich darin befindenden Objekte verantwortlich ist. Im Gegensatz zu der ersten YOLO Version werden bei SSD mehrere Skalierungen des Bildes analysiert, womit kleinere Objekte besser erkannt werden. Abbildung 43 vergleicht die Architekturen von SSD und YOLO. Es ist erkennbar, dass das SSD Netzwerk mehr Bestandteile hat, womit eine höhere Genauigkeit erreicht wurde.

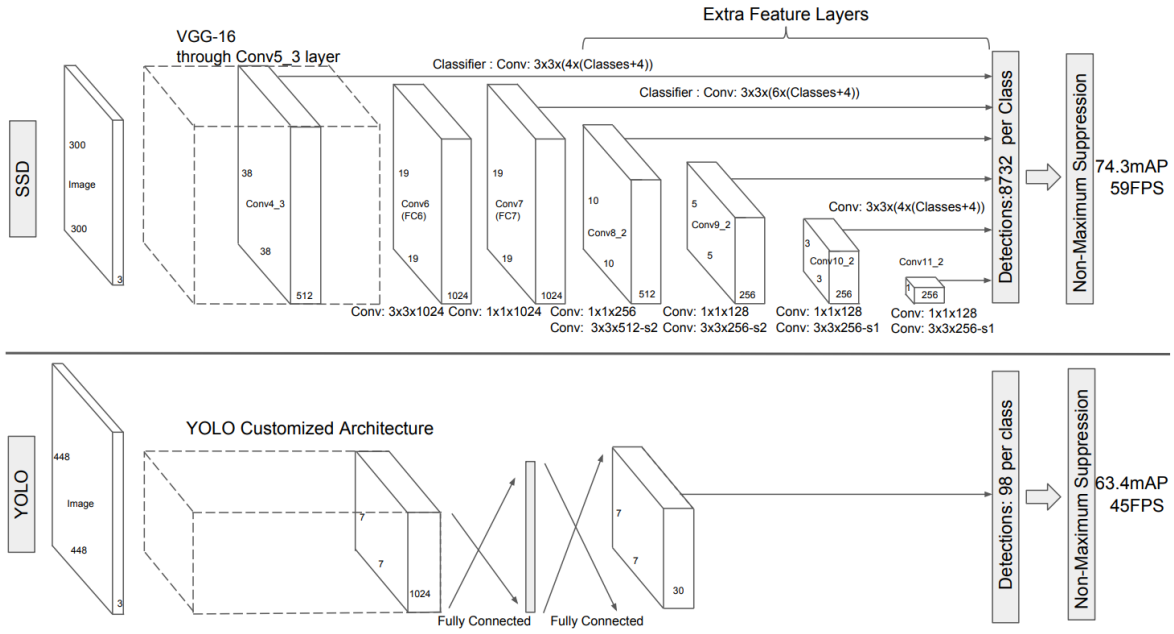


Abbildung 43 Vergleich Architekturen SSD und YOLO [27]

2.6 Vor- und Nachverarbeitung von Daten

Auszuwertende Daten müssen vor der Inferenz manipuliert werden, um den Input Parametern des Models zu entsprechen. Nachfolgend werden die Vor- und Nachverarbeitungsschritte beschrieben.

2.6.1 Preprocessing

Die zu analysierenden Bilder müssen vor der Inferenz vorverarbeitet werden. Dazu gehört das Konvertieren des Datentyps und die Anpassung der Dimensionen. Bei einem Quantisierten INT8 Model werden die Bilder von Float32 in INT8 konvertiert. Die meisten Models arbeiten mit quadratischen Bildern.

Für die Konvertierung von einem rechteckigen in ein quadratisches Bild gibt es verschiedene Ansätze, die in Abbildung 44 visualisiert sind. Der einfachste Ansatz ist die Skalierung der x- und y Achse um unterschiedliche Faktoren, um möglichst viele Pixel des Originalbildes zu behalten. Der Nachteil liegt in einer Verzerrung des Bildes. Falls das Model mit verzerrten Bildern trainiert wurde, wird es die Objekte trotzdem gut erkennen. Wenn die Aspect Ratio nicht verändert werden soll, wird die Letterbox Transformation angewandt. Die längere Achse wird auf die vorgegebene Dimension verkleinert, die kürzere Achse wird um den selben Faktor angepasst. Der Rahmen wird mit einer neutralen Farbe ergänzt. Es ist möglich, einen Stride hinzuzufügen, falls das Model bei den Conv Layern keinen Stride ergänzt.

Abbildung 44 Transformation eines rechteckigen in ein quadratisches Bild

Beim Vergrössern oder Verkleinern eines Bildes können unterschiedliche Resampling Algorithmen angewandt werden, um den Qualitätsverlust zu minimieren. Abbildung 45 zeigt gezoomte Ausschnitte aus verkleinerten Bildern mit unterschiedlichen Algorithmen. Eine Transformation mit geringerem Qualitätsverlust benötigt generell mehr Rechenaufwand.

Abbildung 45 Bildqualität nach Resize mit verschiedenen Resampling Algorithmen

Um die Unterschiede der Performance aufzuzeigen, wurde ein Bild mehrmals mit unterschiedlichen Algorithmen verkleinert und jeweils die Durchschnittszeit berechnet. Die Resultate sind in Abbildung 46 visualisiert. Speziell auf Geräten mit beschränkter Berechnungsleistung kann die Zeit der Auswertung durch die Wahl eines schnelleren Algorithmus stark verringert werden.

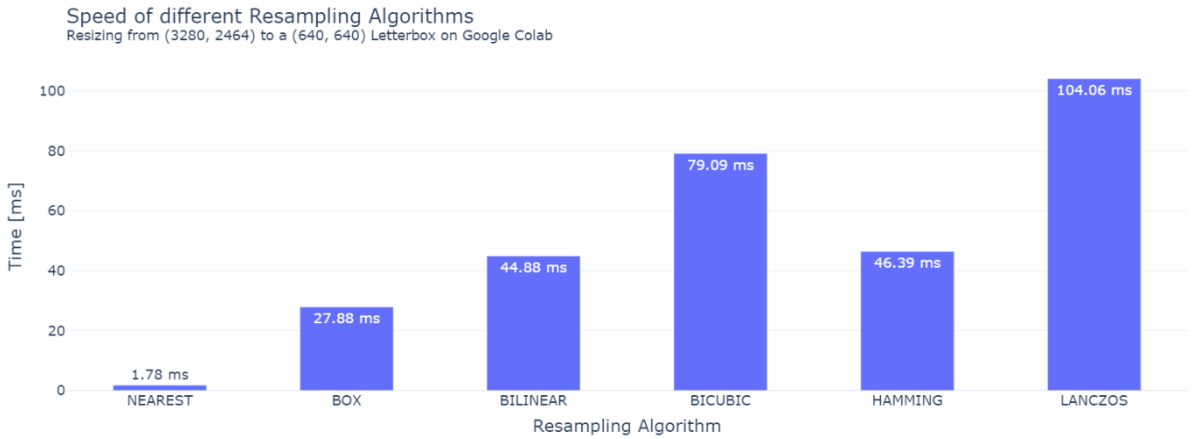


Abbildung 46 Geschwindigkeiten verschiedener Resampling Algorithmen

2.6.2 Postprocessing

Das Postprocessing beinhaltet das Skalieren der Bounding Boxen um den Faktor, der beim Preprocessing angewandt wurde. Die erkannten Objekte müssen gefiltert werden, damit irrelevante Resultate unterdrückt werden. Für die Filterung wird NMS eingesetzt, die im folgenden Unterkapitel beschrieben wird.

2.6.2.1 Non Maximum Suppression (NMS)

Eine Objekterkennungspipeline enthält am Schluss eine Komponente zur Erzeugung von Vorschlägen für die Klassifizierung. Ein Vorschlag besteht aus einer Bounding Box und der Wahrscheinlichkeit, dass sich darin ein zu detektierendes Objekt befindet. Die Anzahl der Vorschläge variiert je nach Model, im Falle eines TFLite Yolov5s Models sind es 25200 [30]. Abbildung 47 zeigt eine gefilterte Ansicht von mehreren vorgeschlagenen Regionen in einem Bild, in welchen sich ein Bestäuber befinden könnte.

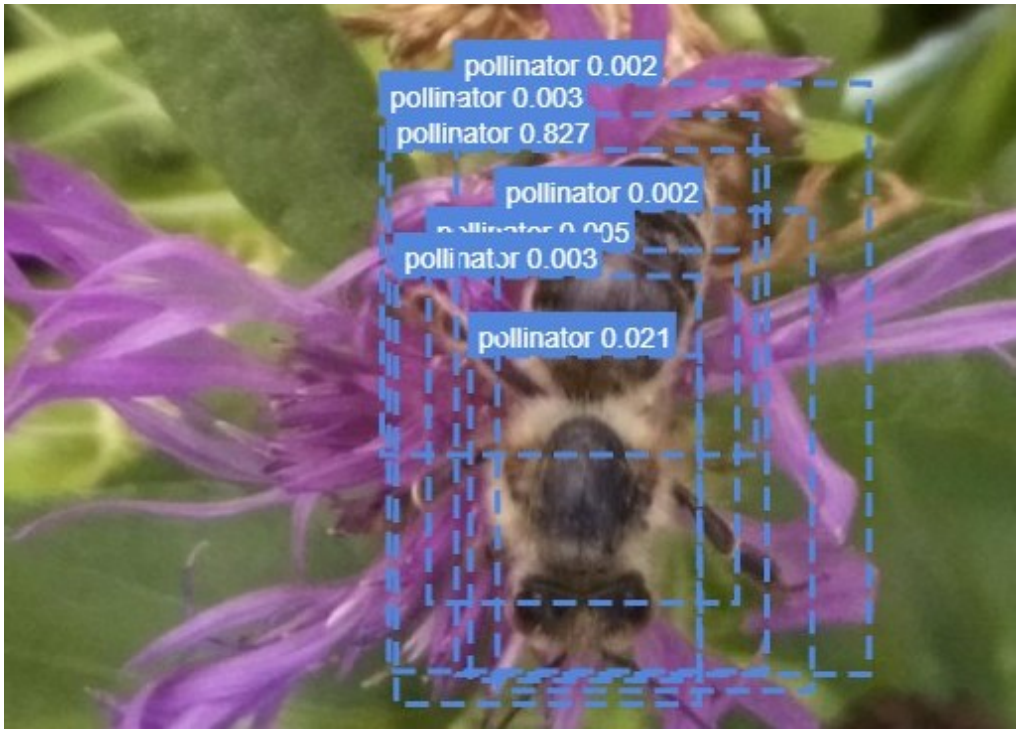


Abbildung 47 Bounding Boxen Yolov5 Training

Der NMS Algorithmus wird dafür eingesetzt, die Vorschläge zu filtern und nur noch eine Bounding Box pro Objekt mit der entsprechenden Confidence beizubehalten. Inputs des Algorithmus sind die Vorschläge des Models und einen IoU Threshold. Der Ablauf von NMS ist wie folgt:

1. Der Vorschlag mit der höchsten Confidence wird ausgewählt und von der Input Liste in die Output Liste verschoben.
2. Für jeden Vorschlag in der Input Liste wird der IoU mit dem ausgewählten Vorschlag berechnet. Ist er über dem IoU Threshold, wird der entsprechende Vorschlag aus der Input Liste entfernt, um zu verhindern, dass ein Objekt doppelt markiert wird.
3. Die Schritte 1 und 2 werden wiederholt, bis sich kein Vorschlag mehr in der Input Liste befindet.

Eine Herausforderung für NMS sind Gruppierungen von Objekten. Wenn das erste Objekt mit einer hohen Confidence erkannt wird, werden im Worst Case alle umliegenden Objekte ignoriert, weil die Bounding Boxen zu stark mit dem ersten Objekt überlappen. Ein einfacher und effizienter Weg, dieser Problematik vorzubeugen ist Soft-NMS [31].

Anstatt Vorschläge mit grossen IoU Werten und hohen Confidence Werten komplett zu ignorieren, werden ihre Confidence Werte proportional zum IoU verkleinert. Abbildung 48 zeigt den Vergleich zwischen NMS (l) und Soft-NMS (r). Durch die starke Überlappung der Zebras wird das Hintere bei der Verwendung von NMS ignoriert, auch wenn ein Vorschlag mit hoher Confidence vorhanden ist. Soft-NMS erkennt es, jedoch mit einer verringerten Confidence.

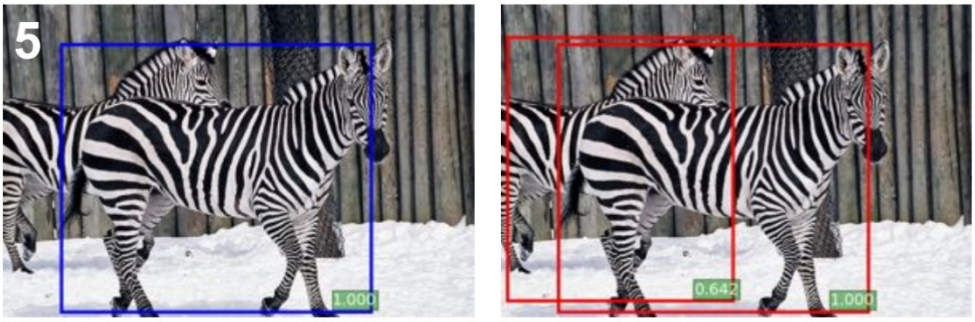


Abbildung 48 NMS vs. Soft-NMS [31]

Die PyTorch Implementation von YOLOv5 hat die NMS bereits in der Auswertungspipeline implementiert. Bei der Konvertierung in ein TFLite Model ist NMS standardmässig nicht dabei und muss implementiert werden. Im Rahmen einer Export Competition von Ultralytics [32] wurde eine TFLite Implementation für Raspberry Pi SBC entwickelt, die Implementierungen von Pre- und Postprocessing inklusive NMS beinhaltet [33].

2.7 ML Hardware

Das Training von ML Models wie auch die Inferenz kann auf den Hardwarekomponenten CPU, GPU oder TPU durchgeführt werden. Eine Analogie der drei Komponenten ist das Drucken eines Dokuments. Eine CPU druckt Buchstabe um Buchstabe, während eine GPU ganze Wörter auf einmal druckt. Die TPU druckt das ganze Dokument in einem Schritt, jedoch mit einer etwas geringeren Auflösung. Die Eigenschaften, Vor- und Nachteile der Komponente werden folgend dokumentiert.

2.7.1 CPU

Jeder Rechner verfügt über eine Central Processing Unit. CPU sind darauf ausgerichtet, eine möglichst kurze Latenzzeit zu erhalten und haben eine sehr schnelle Memory Access Time. Die Berechnungen, die sie in einem Taktzyklus durchführen können sind limitiert, was die Zeit für die Inferenz und vor allem für das Training von Models relativ hoch ausfallen lässt.

2.7.2 GPU

Graphics Processing Units werden oft als «Herz des Deep Learnings» bezeichnet, da sie parallele Berechnungen in einer Vielzahl von Threads berechnen. GPUs verfügen über einen dedizierten Arbeitsspeicher, auf den sie sehr schnell zugreifen können. Das Bottleneck befindet sich in der Übertragung der Daten aus dem Memory zu der GPU. Die Inferenz auf einer GPU ist wesentlich schneller als auf einer herkömmlichen CPU, dafür resultiert auch ein höherer Energieverbrauch. Gemäss den Benchmarks von YOLOv5 kann die Inferenzzeit der Models damit um Faktor sieben verringert werden.

2.7.3 TPU

Tensor Processing Units wurden von Google spezifisch für den Einsatz von Deep Learning entwickelt und ermöglichen schnellere Inferenzen als GPUs. TPUs verarbeiten quantisierte Models, die nur aus bestimmten Datentypen bestehen und sind somit kleiner als Float32 Models. Der Kern einer TPU ist die Matrix Multiply Unit (MXU), ein grosses Systolisches Array, womit ein limitiertes Set von hardwareseitig implementierten Matrixoperationen ohne Memory Access möglich sind. In der ersten Version der TPU waren somit 256'000 Operationen pro Takt möglich [34]. Die Quantisierung der Models hat eine geringere Genauigkeit zur Folge.

Coral, eine Plattform von Google für die lokale Anwendung von AI, entwickelt TPU Accelerator für den Einsatz im Edge Computing Bereich. Neben Single Board Computer mit integrierten TPUs produzieren sie auch externe TPU Einheiten, die über USB an einem Rechner angeschlossen werden. Der Kostenpunkt beträgt knapp 60 USD [35]. Abbildung 49 zeigt die Schritte für das Deployment mit TPU eines Deep Learning Models. Das Model muss nach dem Training quantisiert und im TFLITE Format exportiert werden. Nach dem Export wird es mit dem EdgeTPU Compiler [36] kompiliert und kann mit dem Accelerator eingesetzt werden.

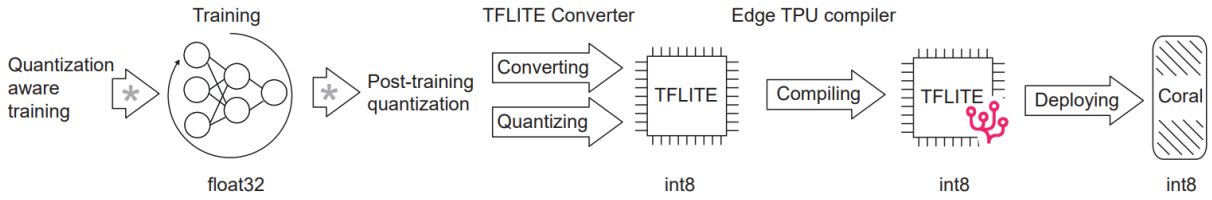


Abbildung 49 Deployment eines DL Models mit Edge TPU [37]

3 Datenexploration

Im Rahmen des Forschungsprojekts Mitwelten wurden knapp 1.5 Millionen Bilder von Blumen aufgenommen. Die Bilder waren auf fünf Festplatten zu 2 TB verteilt. Die Dateinamen der Bilder bestehen aus der ID der Kamera und einem UTC Timestamp im ISO 8601 Format. Die Bildformate sind JPG. Ein Bild, welches am 19. Juli 2021 um 11:48:07 (Lokalzeit) von der Kamera mit der ID 1996-0324 aufgenommen wurde, wird unter folgendem Namen gespeichert:

1996-0324_2021-07-19T09-48-07Z.jpg

Die zeitliche Verteilung der Aufnahmen wird in Abbildung 50 visualisiert. An den ersten Aufnahmetagen wurden mögliche Capture Intervalle getestet, womit überdurchschnittlich viele Bilder produziert wurden. Aufgrund von defekten Kameras wurden im Juni und im Juli weniger Bilder aufgenommen. Mitte August wurden die Aufnahmen an den ersten Standorten beendet.

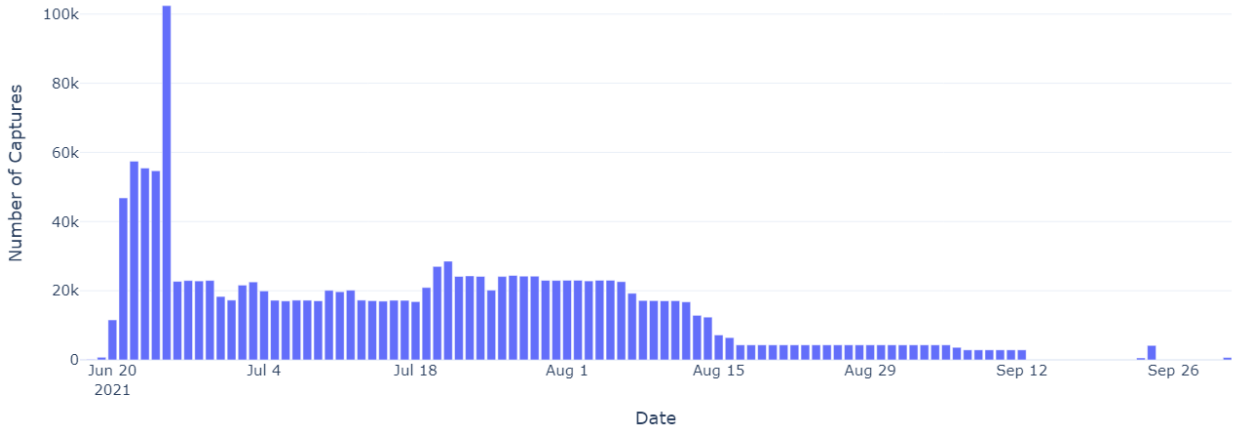


Abbildung 50 Aufnahmen pro Tag

Die Aufnahmen stammen von 18 Kameras, die auf fünf Standorte aufgeteilten wurden. Die Zusammensetzung der Herkunft ist in Abbildung 51 visualisiert. Dabei ist ersichtlich, dass von drei Kameras durch Defekte weniger Bilder vorhanden sind.

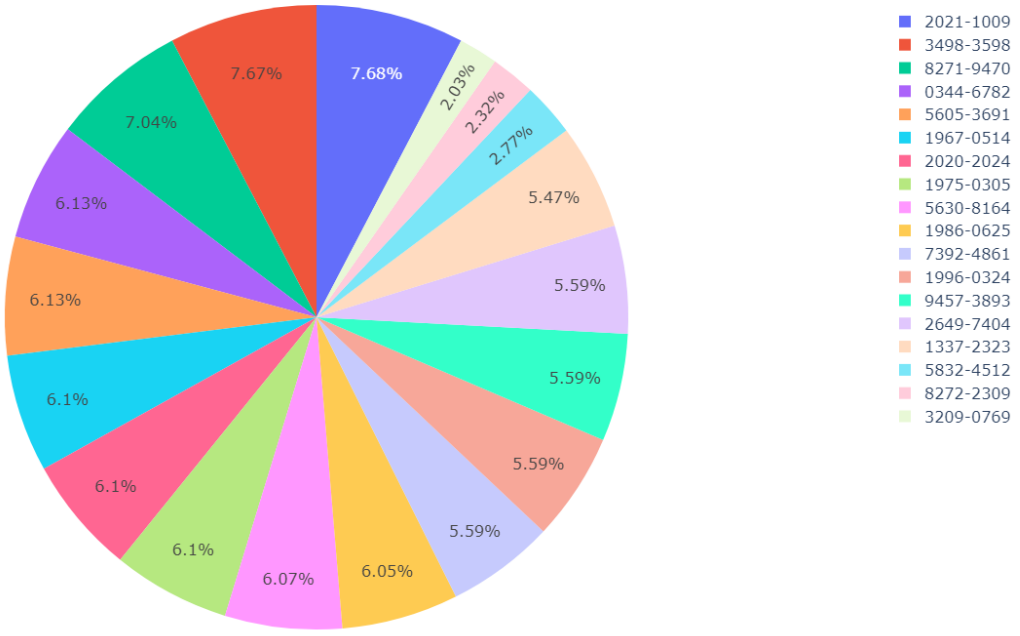


Abbildung 51 Aufnahmen pro Kamera

3.1 Image Exploration Tool

Die Qualität der erhobenen Bilder variiert stark. Unterschiede zeigen sich in der Distanz zu den Blumen, in der Bildschärfe und in der Auflösung. Bis zum 20. Juni wurden die Bilder mit einer Auflösung von 1920x1080 erfasst, später mit 3280x2464. In sehr vielen Aufnahmen sind keine blühenden Blumen vorhanden. Um Zeiträume der Aufnahmen mit bzw. ohne sichtbare Blumen zu unterscheiden, wurde das Image-Exploration Tool entwickelt. Der Code ist auf GitHub im Repository P8-Tools [38] abgelegt.

Von allen Daten werden die Namen, das im Namen enthaltene Aufnahmedatum und die Kamera ID in eine SQLite [39] Datenbank geschrieben. Die Datenbank besteht aus einer Table. Die Felder sind in Tabelle 1 abgebildet.

Feld	Beschreibung
filename	Der Dateiname des Bildes.
path	Der relative Pfad des Bildes.
node_id	Die ID der Kamera, die das Bild aufgenommen hat.
date	Das Aufnahmedatum.
flower	Die Information, ob Blumen auf dem Bild ersichtlich sind.
pollinator	Die Information, ob Bestäuber auf dem Bild ersichtlich sind.
available	Die Information, ob die Datei auf einem zur Zeit angeschlossenen Laufwerk verfügbar ist.

Tabelle 1 SQLite Table Image Exploration Tool

Die Bilder sind auf fünf Festplatten verteilt, die nacheinander eingelesen und in die Datenbank eingetragen wurden. Um zu wissen, welche Bilder momentan zugänglich sind, wurde das Feld «available» eingeführt. Bei einem Wechsel der Festplatte wird das Feld in der Datenbank aktualisiert. Die Felder «flower» und «pollinator» sind für Metainformationen zu den Bildern vorgesehen.

Für das Hinzufügen von Metainformationen wurde mit dem Python Framework Dash [40] ein User Interface in Form eines Dashboards entwickelt. Das primäre Ziel des Interfaces ist es, die Bilder ohne sichtbare Blumen zu markieren, um die Datenmenge für die nächsten Schritte zu minimieren. In der Hauptansicht des Dashboards, welche in Abbildung 52 ersichtlich ist, geschieht das Markieren durch Betätigen der entsprechenden Schaltfläche oder durch Drücken der Tasten «y» für «Blume(n) Sichtbar» und «x» für «Keine Blume(n)» sichtbar. Die Bilder können in zufälliger Reihenfolge oder nach dem Aufnahmedatum klassifiziert werden. Angezeigt werden nur die Bilder, welche sich auf einer momentan angeschlossenen Festplatte befinden. Der Bereich kann durch die Filter auf der rechten Seite zusätzlich nach Kamera ID oder Aufnahmezeitraum eingeschränkt werden. Für Aufnahmen, die nicht eindeutig klassifizierbar sind, gibt es die Option, sie als «Not Sure» zu klassifizieren.

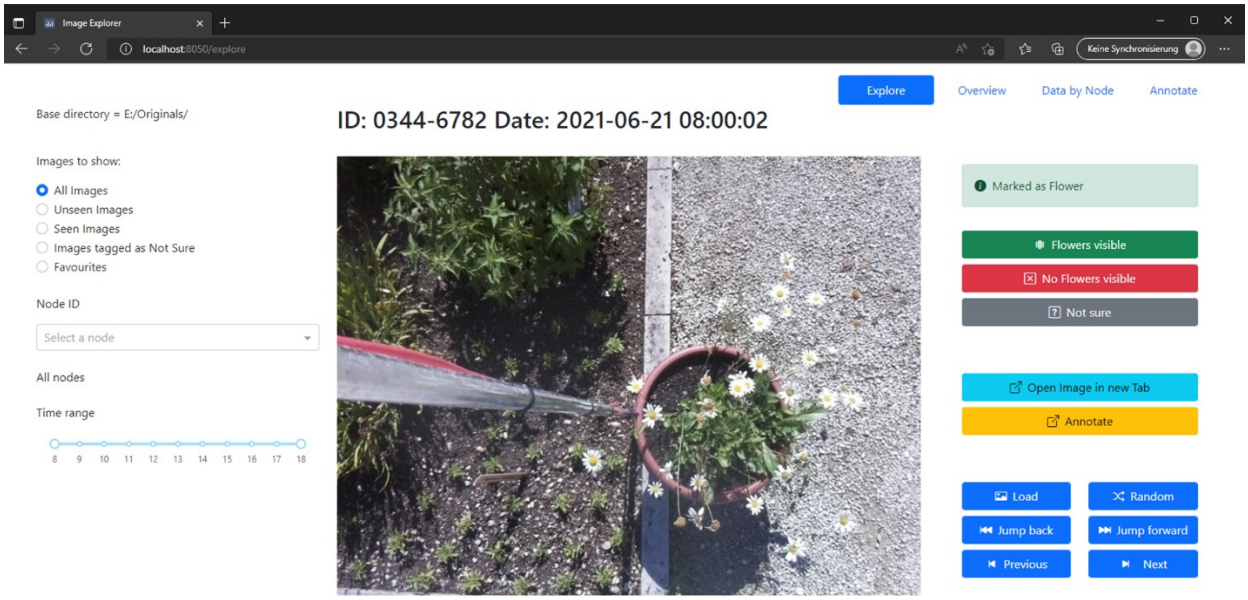


Abbildung 52 User Interface Image Exploration Tool

Knapp 2700 Bilder wurden mit dem Tool klassifiziert. Zeitabschnitte, die für das Projekt brauchbar sind, werden aus den Informationen hochgerechnet. Pro Kamera wird eine Detailansicht generiert. Abbildung 53 zeigt die Ansicht der Kamera 1967-0514 mit 71 klassifizierten Bildern. Anhand der berechneten Zeiträume wurden die brauchbaren Bilder von allen Kameras auf eine 5TB Harddisk kopiert. Rund 50 % aller Aufnahmen sind brauchbar, der Rest wurde nicht weiter verwendet.

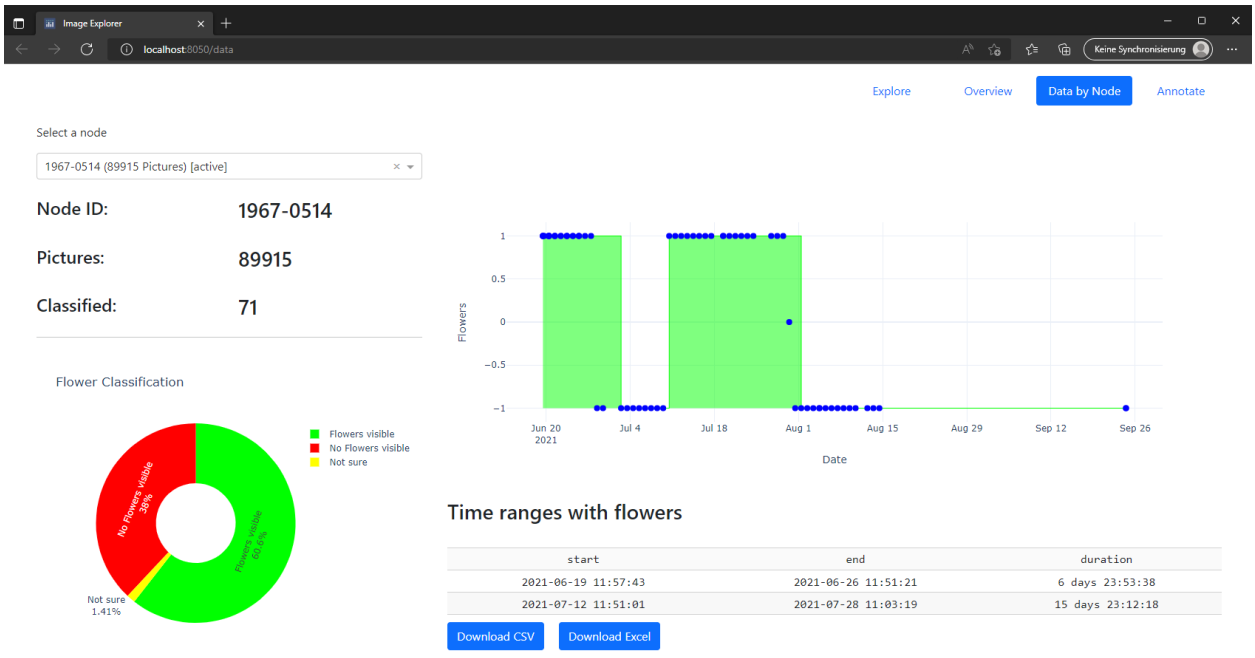


Abbildung 53 Image Exploration Tool Detail Ansicht

Abbildung 54 visualisiert die klassifizierten Bilder. Die grünen Marker signalisieren brauchbare Bilder, die grauen Marker sind Bilder ohne sichtbare Blumen. Die besten Zeiträume über alle Aufnahmen betrachtet, liegen um den 25. Juni und um den 25. Juli. In den Bereichen ohne Marker wurden keine Bilder erfasst.

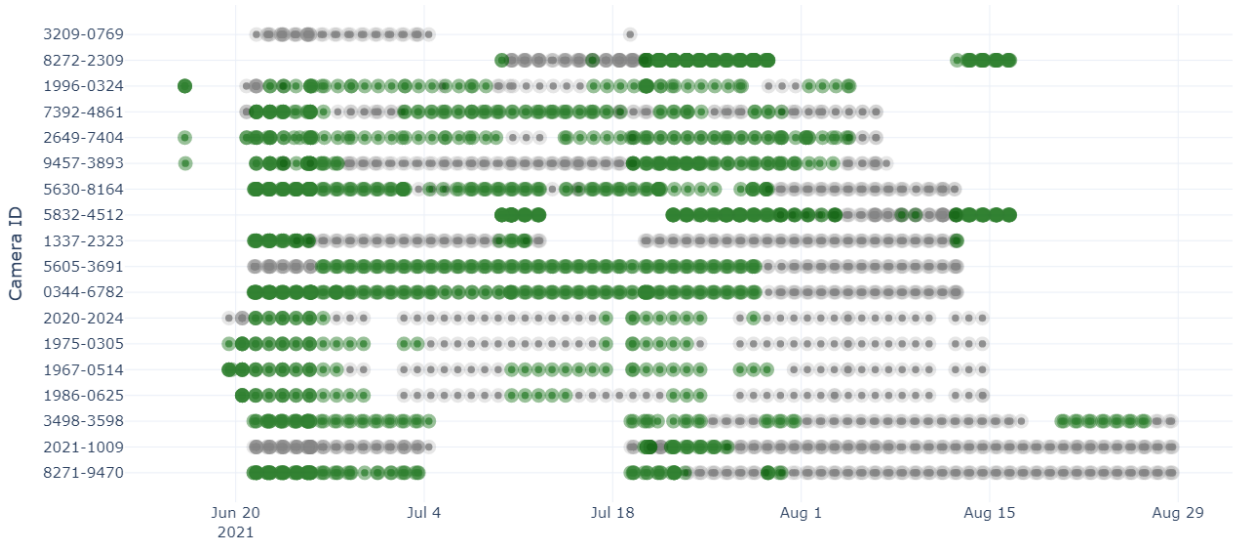


Abbildung 54 Bilder mit Blumen nach Kameras und Aufnahmedatum

3.2 Annotieren der Bilder

Um mehr Informationen über den Inhalt der Bilder zu erhalten, wurde das Image Exploration Tool um die entsprechende Funktionalität erweitert. Eine Annotation besteht aus einer Bounding Box und einer Klasse und werden in einer SQLite DB gespeichert. Für den Export als Textdateien wurde ein Skript geschrieben, welches auch die entsprechenden Bilder in den Zielordner kopiert.

Für das Annotieren von Bildausschnitten mit Bestäubern oder von Blumen, die nicht in der Datenbank des Image-Exploration-Tools sind, wurde der Standalone-Annotator, ein alleinstehendes Tool, entwickelt. Bilder im Input Ordner werden eingelesen und in die Label Queue geschrieben, die nach und nach abgearbeitet wird. Ein Screenshot der Applikation ist in Abbildung 55 ersichtlich. Die Labels werden im YOLOv5-kompatiblen Textformat gespeichert.

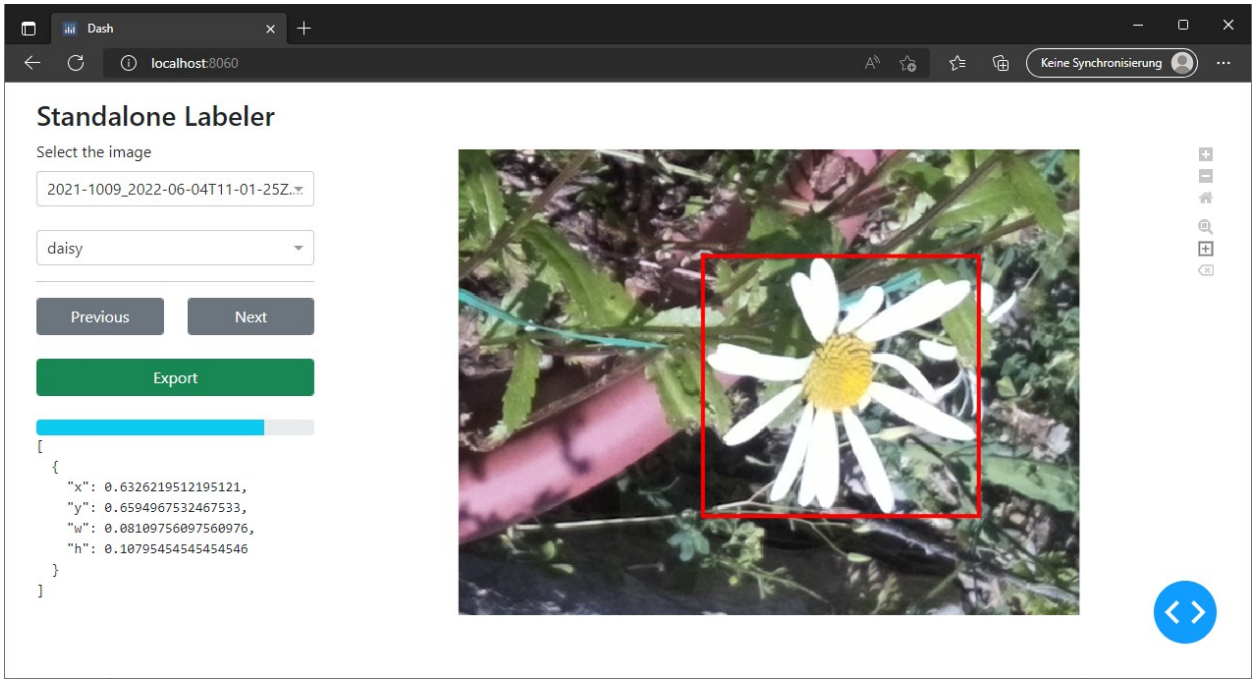


Abbildung 55 Screenshot Standalone-Annotator

4 Auswertung von Bilddaten

Auf Bildern, die von dem Mitwelten Kamerasystems aufgenommen werden, sollen Bestäuber detektiert werden. Mit den Resultaten der Auswertung sollen Aussagen über Indikatoren der Biodiversität von Bestäubern machbar sein. Die Hauptanforderung ist das Erkennen von Bestäubern auf den Blumen, womit deren Aktivität in Relation mit dem Standort, Aufnahmedatum- und Zeit erforscht werden kann. Weiter soll erforscht werden, ob es möglich ist, verschiedene Klassen von Bestäubern zu unterscheiden. Die Auswertung dient sekundär zur Reduzierung der Datenmenge, da anstelle der Originalbilder nur Ausschnitte mit relevanten Informationen gespeichert werden.

Der State-of-the-Art Ansatz zum Erkennen von Objekten auf Bildern sind Deep Learning Models. Methoden der klassischen Bildverarbeitung zur Erkennung von Blumen auf Bildern wurden im Rahmen des Mitwelten Forschungsprojekt analysiert. Die Ergebnisse zeigen eine geringe Robustheit gegen Einflüsse wie Hintergrund, Beleuchtung oder Schärfe. Für die Entwicklung der Auswertung wurde der Deep Learning Ansatz verfolgt.

Die Mediangrösse eines Bestäubers beträgt gut 70x70 Pixel auf einem Bild mit einer Auflösung von 3280x2464 und sind somit sehr klein. Ein Objekterkennungsmodel auf der Originalgrösse zu trainieren ist sehr ineffizient und setzt leistungsstarke, für Deep Learning Anwendungen optimierte Hardware voraus. Die Bestäuber befinden sich in den Aufnahmen der ersten Fallstudie bis auf wenige Ausnahmen auf den Blumen.

Im Durchschnitt machen die Blumen gut 14 % der Bildfläche aus, die restlichen 86 % sind für das Erkennen der Bestäuber irrelevant. Blumen besitzen die Eigenschaft, sich optisch von der Umgebung abzuheben und somit die Aufmerksamkeit der Bestäuber auf sich zu ziehen. Die Erkennung von Blumen ist durch die Merkmale mit einem leichten Objekterkennungsmodel realisierbar.

Damit die Bestäuber für die Analyse eine genügend hohe Auflösung haben, wurde eine Two-Stage Pipeline für die Auswertung entwickelt. Abbildung 56 zeigt die zwei Schritte für das Erkennen von Bestäubern. Auf einer verkleinerten Kopie des Originalbildes detektiert ein Model Blumen. Die Bereiche mit gefundenen Blumen werden anschliessend aus dem Originalbild ausgeschnitten. In einem zweiten Schritt sucht ein weiteres Model jeden Ausschnitt nach Bestäuber ab und schneidet sie aus.

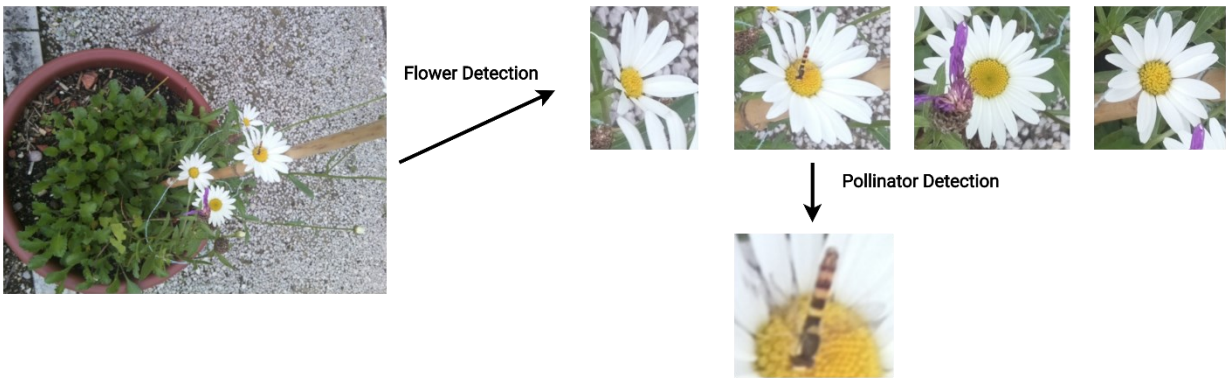


Abbildung 56 Pipeline für die Auswertung der Bilddaten

Die Resultate der Auswertung beinhalten somit neben Informationen über die Bestäuber auch Informationen über die Anzahl und Arten von vorhandenen Blumen. Als Alternative könnte der zweite Schritt durch einen Image Classification Algorithmus ausgetauscht werden. Ein Nachteil davon ist, dass die Ausschnitte als gesamtes klassifiziert werden. Mehrere Bestäuber würden nicht erkannt und die Gefahr, die Entscheidung aufgrund der Blume und nicht des Bestäubers zu treffen, wäre viel höher. Zudem wird mehr Speicherplatz benötigt, um die grösseren Ausschnitte zu speichern. In folgenden Unterkapiteln wird die Entwicklung der Auswertung dokumentiert.

4.1 Datensets

Für die Erkennung von Blumen und Bestäubern wurde jeweils ein Datenset aus den Aufnahmen der ersten Fallstudie erstellt. Für das Annotieren der Blumen wurde das Image Exploration Tool eingesetzt. Aus zufällig gewählten Bildern wurde ein erstes Flower Datenset, bestehend aus gut 100 Bildern und einer Klasse, zusammengestellt und ein YOLOv5 Model trainiert. Trotz der geringen Anzahl Trainingsdaten erkannte das Model fast alle Blumen auf den Testbildern und konnte somit verwendet werden, um Ausschnitte aus Bildern für das Pollinator Datenset zu generieren. In folgenden Unterkapiteln werden die letzten Versionen der Datensets beschrieben.

4.1.1 Flower Dataset

Das Datenset besteht aus 501 Bildern mit 1653 annotierten Blumen. 364 Bilder stammen von der ersten Fallstudie des Mitwelten Projekts. Die Qualitäten der Aufnahmen variieren sehr stark. Bei Aufnahmen der Wilden Möhre war die Kamera bei allen Bildern sehr nah am Objekt, was in einem starken Bias der ersten Model Versionen resultierte. Grosse, helle Objekte, wie etwa Schilder wurden mit einem hohen Score als Wilde Möhre erkannt. Da im Rahmen des Mitwelten Projekts keine diversere Bilder der Wilden Möhre erfasst wurden, wurde das Datenset mit Bildern aus der iNaturalist [41] Datenbank ergänzt. Das Citizen-Science Projekt iNaturalist verfolgt das Ziel, die Biodiversität anhand von Beobachtungen der Community zu erforschen.

Insgesamt 137 Bilder der Wilden Möhre, die in der Schweiz aufgenommen wurden und qualitativ hochwertig sind, wurden heruntergeladen, um eine bessere Generalisierung der Models zu ermöglichen. Zum Schutz des Urheberrechts werden die heruntergeladenen Bilder nicht im publizierten Datenset vorhanden sein. Alle Fotos von iNaturalist wurden nach dem Herunterladen von mehreren vortrainierten YOLOv5 Models auf das Vorhandensein von Menschen, inklusive Hände oder Füsse, getestet. Die entsprechenden Dateien wurden gelöscht. Das annotieren der Bilder geschah manuell, wobei jedes Bild noch einmal betrachtet und auf das Vorhandensein von Personen kontrolliert wurde.

Für die Zusammenstellung des Datensets wurden in einem ersten Schritt zufällige Bilder aus allen Aufnahmen des Mitwelten Projekts ausgewählt und annotiert. In einem zweiten Schritt wurden gezielt Aufnahmen von noch nicht vorhandenen Kameras und Zeiten gewählt, um die gesamten Aufnahmen gut zu repräsentieren. Abbildung 57 zeigt die Herkunft und Aufnahmezeitpunkte der Bilder im Datenset. Die Grösse der Punkte dient als Indikator der Anzahl Blumen, die sich auf einem Bild befinden. Alle Aufnahmedaten der Bilder von iNaturalist sind in der Grafik in einem Zeitpunkt zusammengefasst.

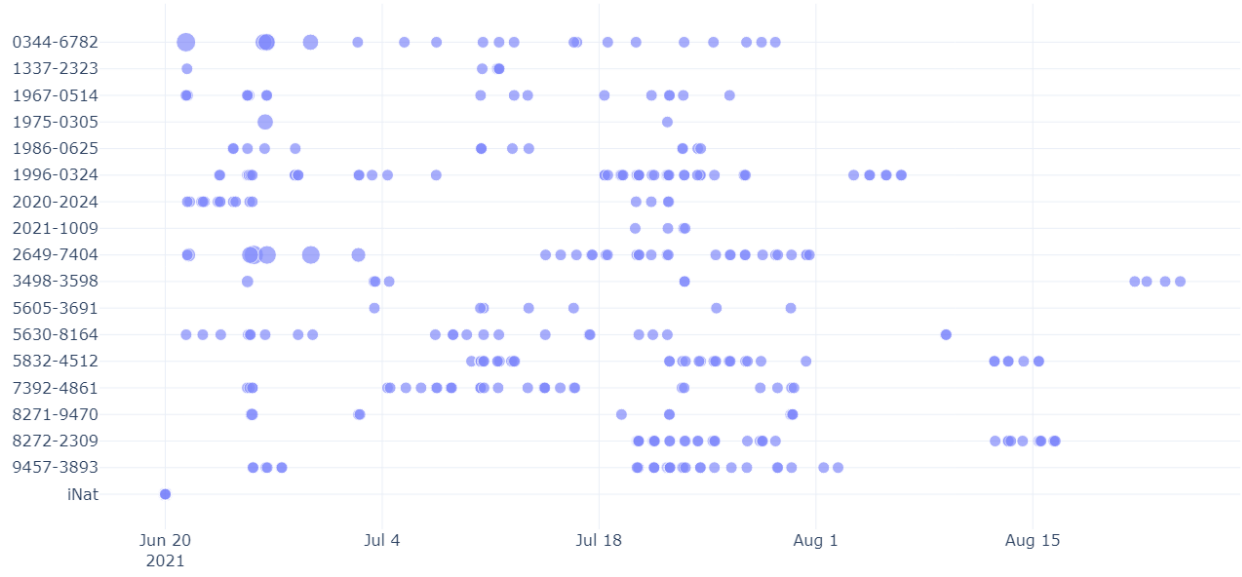


Abbildung 57 Flower Dataset: Herkunft und Aufnahmedatum

Mit 811 Annotierten Margariten ist sie im Datenset am stärksten Vertreten und macht knapp 50 % aller Blumen aus. Die restlichen 50% teilen sich die Wilde Möhre und die Flockenblume. Die Aufteilung ist in Abbildung 58 visualisiert.

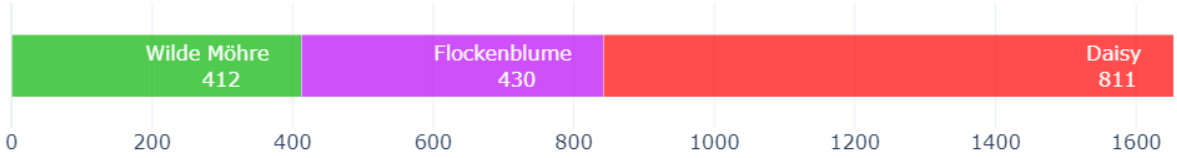


Abbildung 58 Flower Dataset: Anzahl Blumen nach Art

Das Datenset beinhaltet keine Background Bilder, da die Blumen einen relativ kleinen Anteil der Bildflächen ausmachen.

Die Grössen der Blumen im Datenset sind sehr unterschiedlich, wie Abbildung 59 zeigt. Die Mediangrösse der Margariten liegt bei ca. 100 Pixel, die der Flockenblumen bei ca. 215 Pixel und die der Wilden Möhre bei 500 Pixel. Diese Informationen sind relevant für die Wahl der Bildgrösse des Models. Wenn die Bilder um Faktor 10 auf eine Weite von 320 Pixel verkleinert werden, ist die Hälfte der Margariten kleiner als 10 Pixel, womit die Erkennung schwieriger wird.

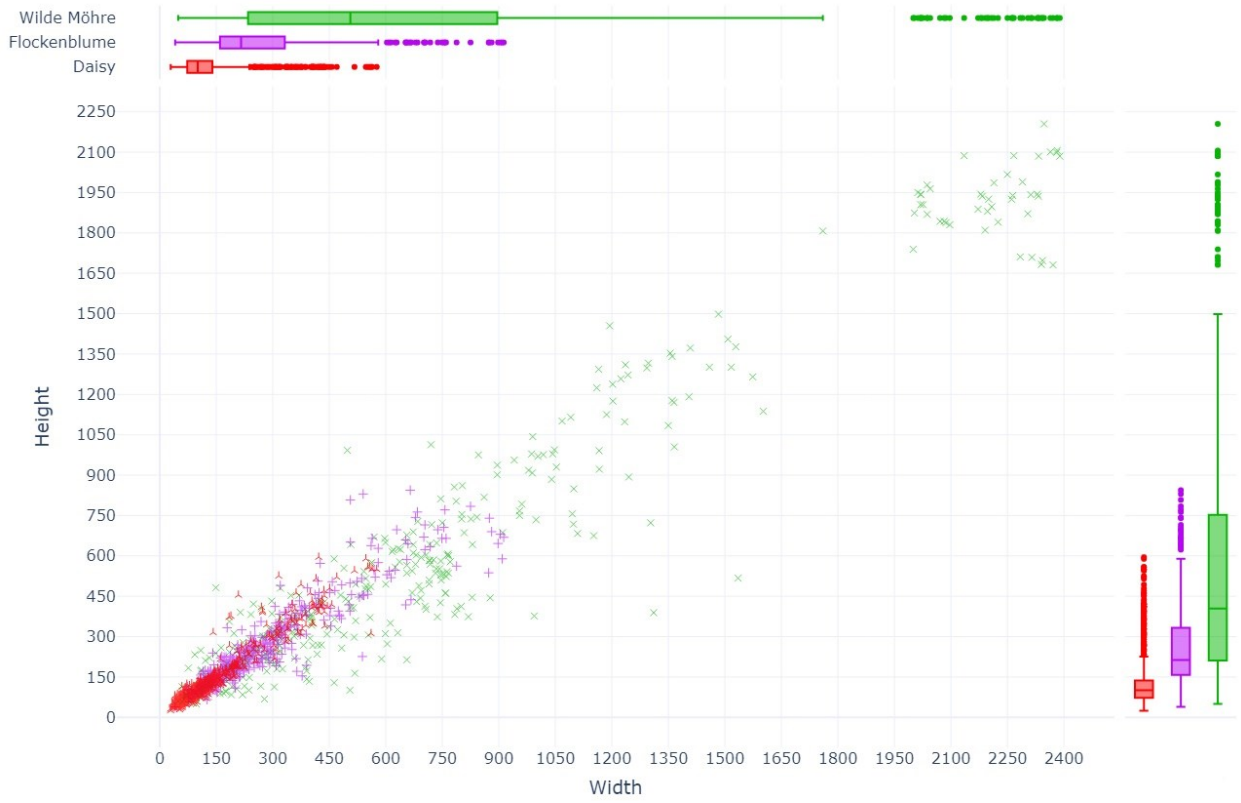


Abbildung 59 Flower Dataset: Grösse der Blumen nach Art

Abbildung 60 zeigt die Örtliche Verteilung der Blumen in den Bildern. Die Grösse der Marker in der Visualisierung zeigt die Grösse der Blumen im Bild. Es ist erkennbar, dass die Margariten eher in der rechten Hälfte vorhanden sind, während die Wilde Möhre vorwiegend zentral positioniert ist. Die Wilden Möhren, die sich nicht in der Mitte befinden, stammen aus den ergänzenden Bildern von iNaturalist.

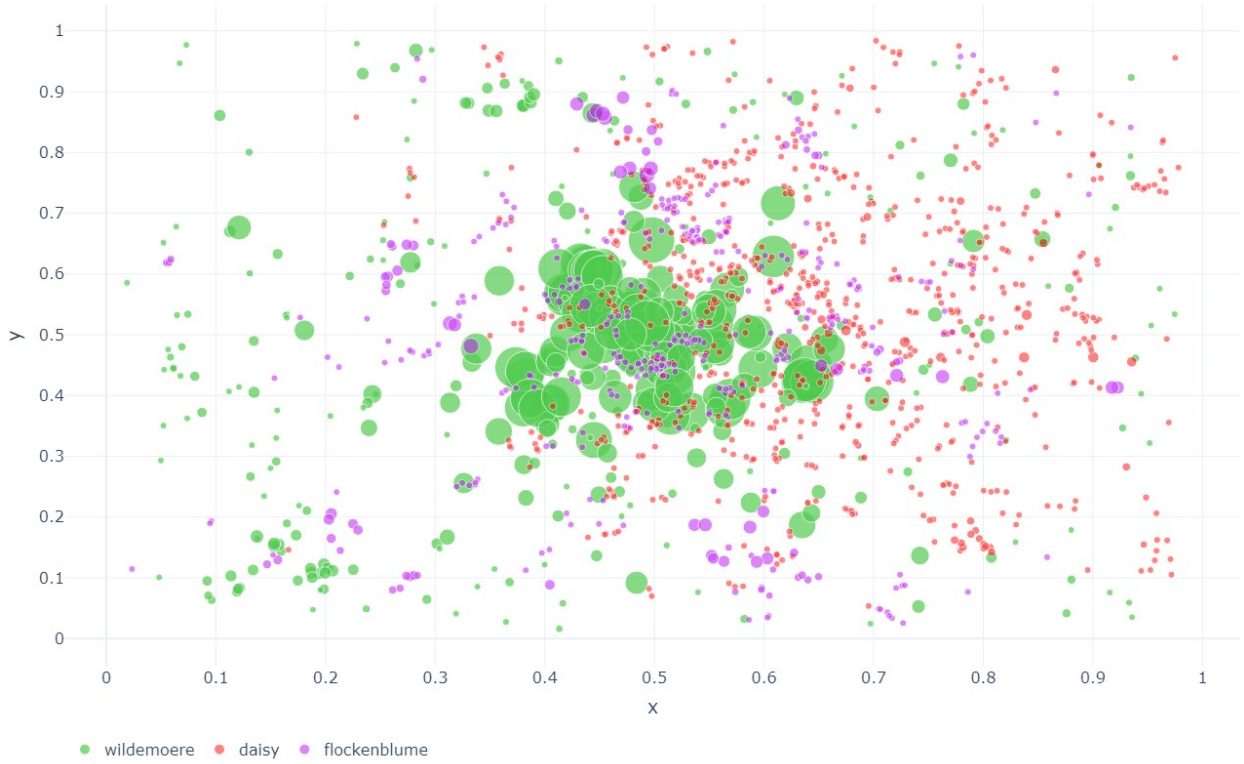


Abbildung 60 Flower Dataset: Verteilung und Grösse der Blumen

Das Datenset wurde auf Kaggle unter dem Namen *mitwelten-flower-dataset* [42] veröffentlicht.

4.1.2 Pollinator Dataset

Das Datenset besteht aus 3’679 Bildern. Auf 2'687 Bildern ist mindestens ein Bestäuber vorhanden. Auf den Aufnahmen sind vermehrt Knospen oder sonstige Teile von Pflanzen ersichtlich, die über sehr ähnliche optische Eigenschaften wie Bestäuber verfügen. Zwei Beispiele sind in Abbildung 61 ersichtlich. Um dem Model zu lernen, was kein Bestäuber ist, wurden 992 Bilder, auf welchen in ersten Testdurchläufen fälschlicherweise Bestäuber erkannt wurden, als Hintergrundbilder hinzugefügt.

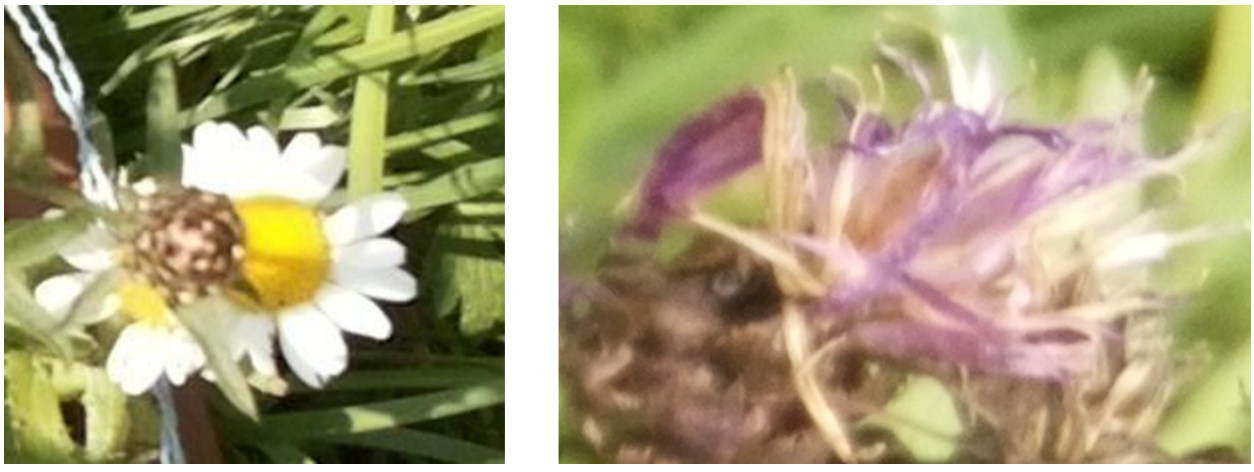


Abbildung 61 Pollinator Dataset: Background Images

Wie bei dem Datenset der Blumen wurde darauf geachtet, dass möglichst viele verschiedene Standorte im Datenset vorhanden sind. Die Blumen waren zu bestimmten Zeiträumen nicht mehr am blühen, was die Vorkommnisse der Bestäuber eingeschränkt hat. Abbildung 62 zeigt die Herkunft und den Aufnahmetag der Bilder. Die Anzahl der Aufnahmen pro Tag wird mit der Grösse

der Marker visualisiert. Die Grünen Dreiecke sind Bilder mit Bestäubern, die grauen Kreise sind Hintergrundbilder.

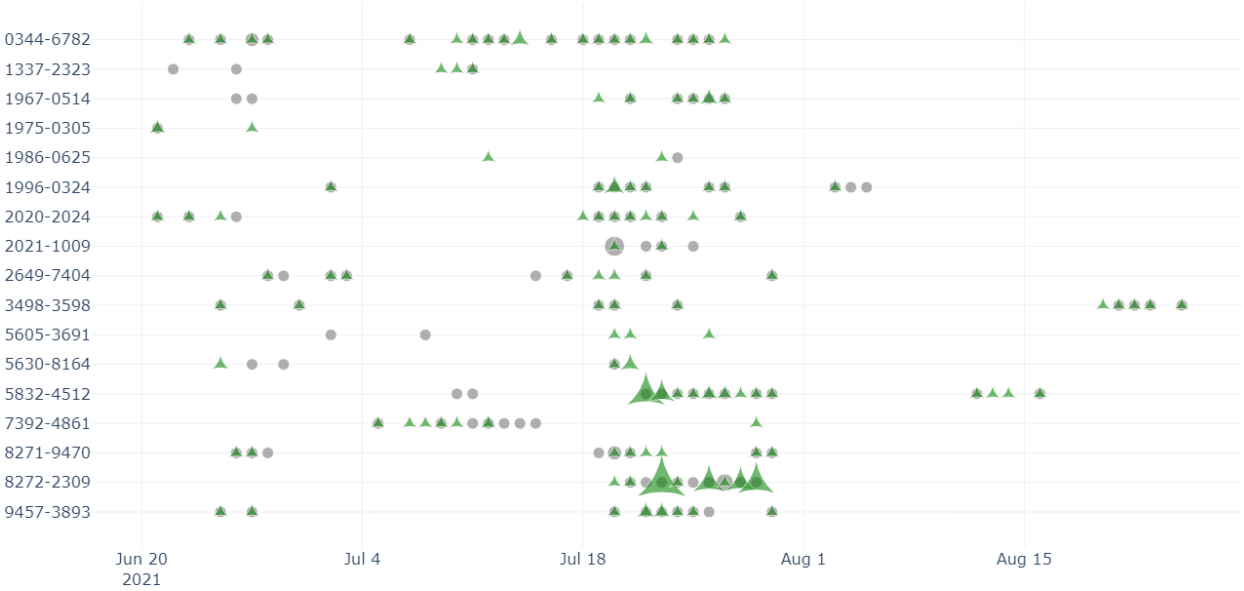


Abbildung 62 Pollinator Dataset: Herkunft und Aufnahmetag inkl. Background Bilder (grau)

Alle Bilder im Datenset wurden von einem Flower Model aus den Originalbildern ausgeschnitten. Die ersten Versionen des Datensets bestanden aus einigen hundert Bildern und einer einzigen Klasse für die Bestäuber. Mit jeder Version des Datensets wurden verschiedene Models trainiert, die eingesetzt wurden, um das Datenset zu erweitern. Mit dem Flower Model wurden Blumen aus zufälligen Bildern ausgeschnitten, worauf das Pollinator Model mögliche Bestäuber detektiert hat. In einem halbautomatisierten Prozess wurden die Vorschläge direkt ins Datenset übernommen, angepasst und übernommen oder als Hintergrundbild eingesetzt. Die Bilder mit den annotierten Bestäubern wurden von einem Biologen in gleichmässige Klassen der Morphospezies kategorisiert. Es resultierten die fünf Kategorien Honigbiene, Wildbiene, Hummel, Schwebfliege und Fliege mit ausreichend Beobachtungen für das Trainieren eines Models.

Für jede Kategorie ist in Abbildung 63 ein Bild aus dem Datenset ersichtlich.

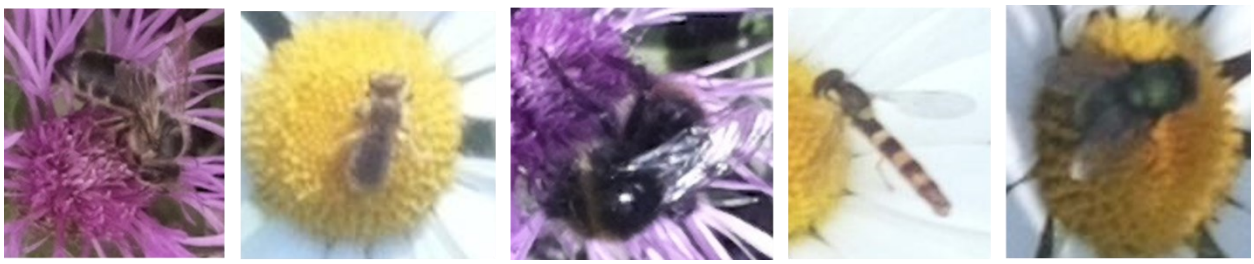


Abbildung 63 Pollinator Dataset: Morphospezies (Honigbiene, Wildbiene, Hummel, Schwebfliege, Fliege)

Die taxonomischen Einordnungen der definierten Klassen sind in Abbildung 64 visualisiert. Aus hierarchischer Sicht befinden sich die Klassen nicht auf der selben Ebene. Die Kategorie Wildbienen umschliesst alle Arten der Überfamilie Apoidea, mit Ausnahme der Gattungen Apis und Bombus. Die Kategorie Fliegen beinhaltet bis auf die Familie Syrphidae alle Spezies der Unterordnung Brachycera.

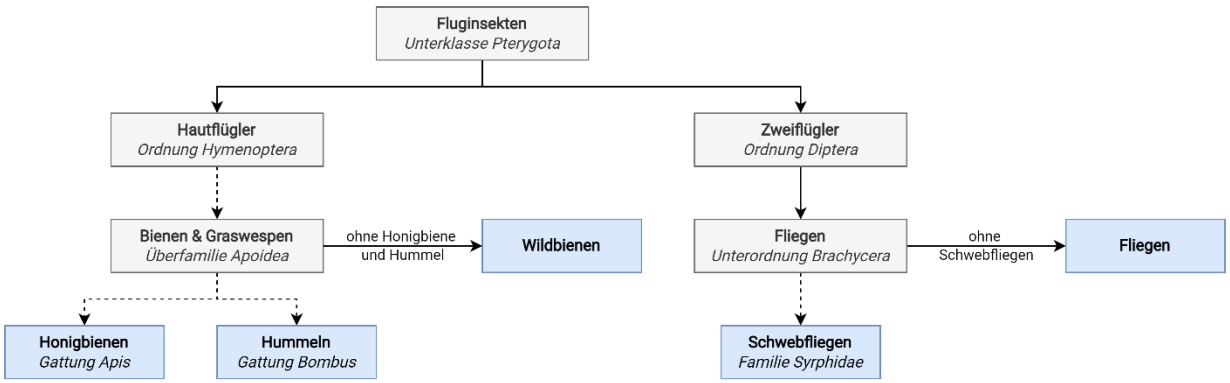


Abbildung 64 Pollinator Dataset: Taxonomie der Bestäuber [43]

Abbildung 65 zeigt die Aufteilung der Bestäuber im Datenset nach Art. Die Grössen der Klassen sind nicht gleichmässig. Wildbienen und Honigbienen verfügen über sehr ähnliche optische Merkmale, was eine Unterscheidung schwierig macht. Schwebfliegen, Fliegen und Hummeln sind optisch besser voneinander unterscheidbar, womit die geringere Anzahl Vorkommnisse das Model nicht schwerwiegend beeinträchtigt.

Abbildung 65 Pollinator Dataset: Anzahl Bestäuber nach Art

Die Grösse der Bestäuber variiert nach Klasse. Die Verteilung der Grössen nach Klasse ist in Abbildung 66 visualisiert.

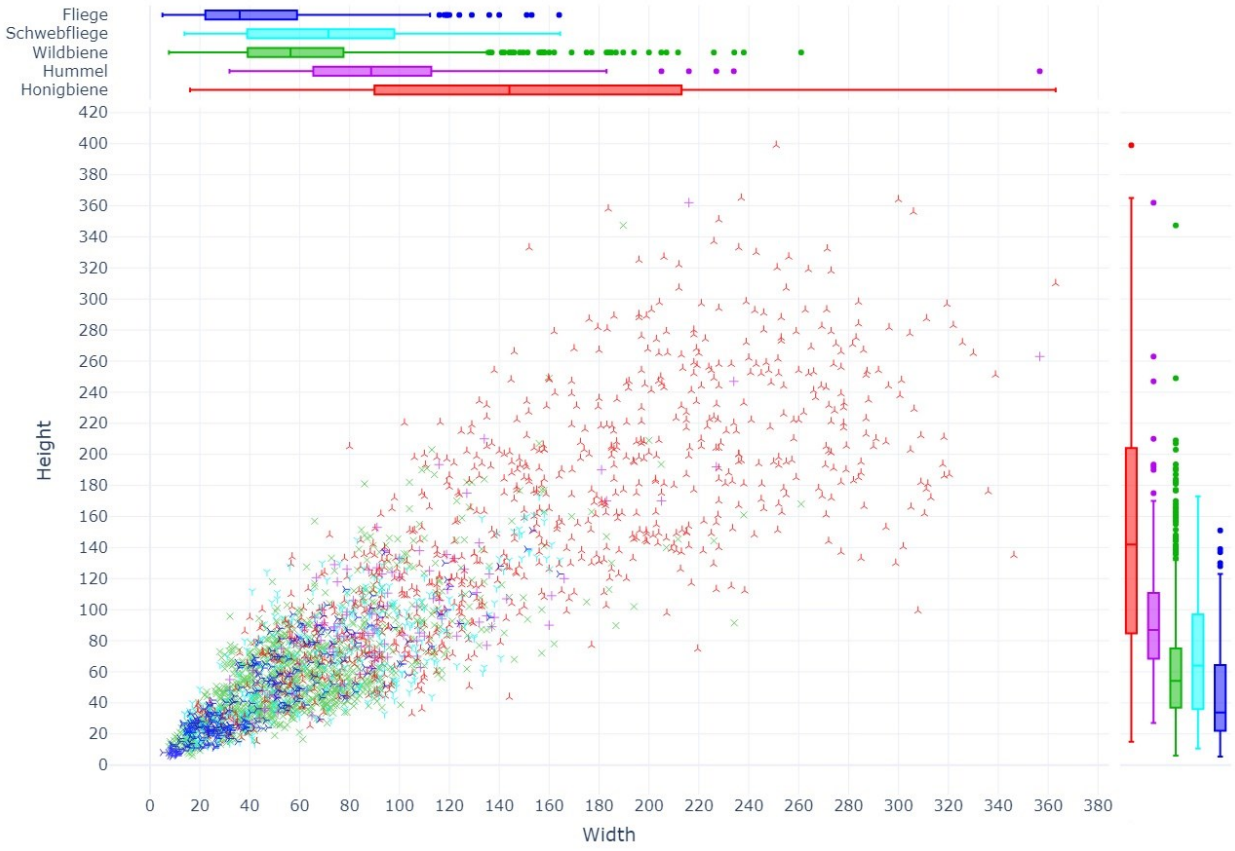


Abbildung 66 Pollinator Dataset: Grösse der Bestäuber nach Art

Auffällig ist, dass die Medianbreite der Honigbiene mit 144 Pixeln wesentlich grösser als die 89 Pixel der Hummel ist. Die Grössen der Bestäuber auf den Bildern sind also nicht proportional zu der realen Grösse. Der Grund für die unterschiedlichen Grössen liegt bei dem Erfassen der Bilder. Die Honigbienen wurden sehr oft in Aufnahmen von Flockenblumen aus kurzer Distanz erfasst. Die Wildbienen und die Fliegen wurden häufig auf Margariten aus einer grösseren Distanz fotografiert.

Die Positionen der Bestäuber auf den Bildern sind in Abbildung 67 visualisiert. Der Durchmesser der Marker ist proportional zu den Dimensionen der Bestäuber. Fliegen und Schwebfliegen befinden sich vermehrt in der Mitte der Bilder, die Honigbienen sind verteilter. Während die Fliegen oft zentral auf den Margariten aufgenommen wurden, gibt es viele Aufnahmen von Honigbienen, die seitlich an einer Flockenblume hängen.

Abbildung 67 Pollinator Dataset: Verteilung und Grösse der Bestäuber

Der grösste Teil der Bestäuber befindet sich auf Margariten oder Flockenblumen, die restlichen auf der Wilden Möhre. Einige Bilder der Wilden Möhre sind mit durchschnittlichen Abmessungen von rund 2000 auf 2000 Pixel viel grösser als die restlichen Bilder. Für ein Model, das mit einer Bildgrösse von 640 Pixel arbeitet, ist es praktisch unmöglich, kleine Bestäuber auf diesen Bildern korrekt zu erkennen. Knapp 2 % der Bilder wurden verworfen, da sie mehr als doppelt so gross wie die restlichen 98 % sind. Als maximale Grösse wurde 1100 auf 1100 Pixel festgelegt. Für die Entwicklung eines Models zur Auswertung von Ausschnitten der Wilden Möhre sind zum aktuellen Zeitpunkt nicht genügend Daten vorhanden. Mit Aufnahmen aus den Fallstudien 2 und 3 des Mitwelten Projekts kann das Datenset erweitert und ein Model mit einer höheren Bildgrösse trainiert werden.

Die Bilder im Datenset haben eine Mediangrösse von rund 320 x 320 Pixel. Ein Bestäuber macht im Durchschnitt 24 % von der Bildfläche aus. Für das Training eines Models muss die zu verarbeitende Bildgrösse festgelegt werden. Um möglichst wenig Informationen zu verlieren ist es naheliegend, diese Grösse möglichst hoch festzulegen. Mit der Bildgrösse des Models steigt die Grösse und die Rechenleistung für die Inferenz. Die Bildgrösse der Pre-Trained YOLOv5 Models liegt bei 640 Pixeln. Die Grösse muss ein Vielfaches von 32 sein. Bei einer Bildgrösse 640

müssen fast alle Bilder im Datenset vergrössert werden. Abbildung 68 zeigt die Grössen der Bestäuber im Originalbild und bei auf die Dimensionen 320, 480 und 640 Pixel angepassten Bildern. Bei jeder Grösse werden die kleinsten Bilder vergrössert, womit auch die kleinsten Bestäuber grösser werden. Bei einer Bildbreite von 320 Pixel werden jedoch auch rund 50 % der Bilder verkleinert, was zum Verlust von signifikanten Merkmalen führen kann. Verschiedene Models wurden unter Anwendung unterschiedlicher Bildgrössen trainiert. Die Auswirkungen auf die Genauigkeit und die Geschwindigkeit werden in den nächsten Kapiteln beschrieben.

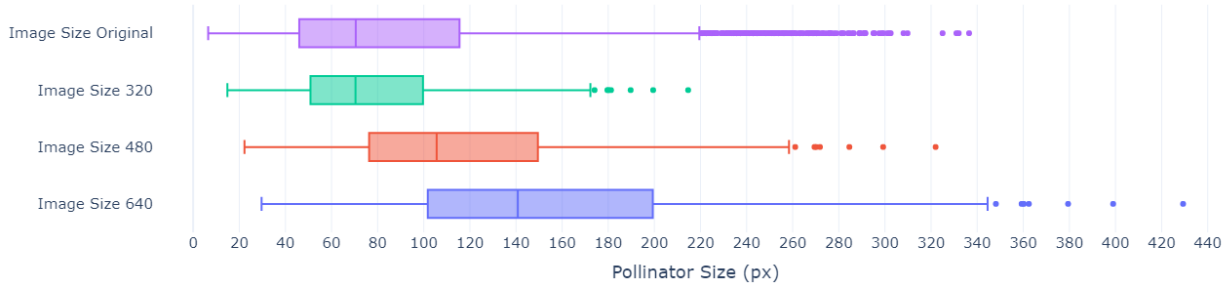


Abbildung 68 Pollinator Dataset: Grössen der Bestäuber nach Resizing

Das Datenset wurde auf Kaggle unter dem Namen *mitwelten-pollinator-dataset* [44] veröffentlicht.

4.2 Wahl des Models

Für die Wahl des Frameworks und Modeltyps wurden verschiedene Optionen analysiert. Die Wahl des gewählten Frameworks basiert auf den folgend genannten Anforderungen.

4.2.1 Anforderungen

Die Hauptanforderung an das Model ist eine möglichst zuverlässige Erkennung der Objekte, damit biologisch relevante Aussagen mit dem System machbar sind. Weiter soll die Inferenz auf Plattformen mit beschränkten Rechenleistungen durchführbar sein, was eine möglichst kleine Modelgrösse voraussetzt. Die Abhängigkeiten für das Deployment sollen minimal ausfallen, um einen Nachbau des Systems zu vereinfachen. Der Trainingsvorgang soll einfach reproduzierbar sein, damit weitere Klassen ohne grossen Aufwand trainiert werden können. Eine optionale Anforderung ist die Möglichkeit, Hardware Beschleuniger wie GPU und TPU bei der Inferenz einzusetzen.

4.2.2 EfficientDet-Lite

EfficientDet [45] ist ein von Google entwickeltes System zur Objekterkennung. Es gilt als sehr schnell, ist skalierbar und genau. Es basiert auf einem gewichteten bidirektionalen Feature Pyramid Network (BiFPN), womit Abhängigkeiten von Features auf verschiedenen Skalierungsebenen gut erkannt werden. EfficientDet Lite Models wurden speziell für Mobile Anwendungen entwickelt. Es gibt fünf Lite Versionen. Die Parameter der Versionen sind in Tabelle 2 ersichtlich.

Version	Bildgrösse	Model Grösse	mAP COCO 2017
Lite0	320x320	18.13MB	27.5
Lite1	384x384	21.85MB	32.6
Lite2	448x448	25.6MB	36.3
Lite3	512x512	36.91MB	39.9
Lite4	640x640	61.49MB	44.5

Tabelle 2 EfficientDet-Lite Models Parameter [46]

Die Bildgrösse des Models ist abhängig von der Modelgrösse, was bedeutet, dass es nicht möglich ist, eine kleine Version wie EfficientDetLite0 oder EfficientDetLite1 mit einer Eingangsgrösse von 640 Pixeln zu trainieren. Die Kompilierung für den Einsatz eines TPU Accelerators ist mit den Versionen 0 bis 2 möglich, bei den höheren Versionen werden mehrere TPUs vorausgesetzt.

Unterschiedliche Grössen wurden mit dem Flower Datenset trainiert. Das beste Ergebnis wurde mit der Version Lite3 erzielt. Nach 300 Trainingsepochen auf einer Tesla P100 GPU in drei Stunden wurde eine Average Precision von 0.64 erreicht. Die Inferenzzeiten sind jedoch sehr hoch. Auf dem Raspberry Pi 3B+ dauerte eine Auswertung knapp fünf Sekunden, was durch unabhängige Benchmarks bestätigt wurde [47]. Die kleineren Versionen produzierten keine brauchbaren Resultate. An allen Resultaten war auffällig, dass die Klassen oft nicht korrekt bestimmt wurden. Eine mögliche Ursache für diese Ungenauigkeit kann an dem globalen NMS liegen.

Der Aufwand für das Training eines Models ist relativ gering. Die Konfigurationsmöglichkeiten sind beschränkt. Das TensorFlow Modul *tflite_model_maker* [48] vereinfacht den Prozess für das Trainieren eines Models mit eigenen Daten stark. Ein bereitgestelltes Tutorial [49] in Form eines Jupyter Notebooks kann modifiziert werden, um ein Model zu erstellen. Die Annotationen müssen als CSV Datei oder im Pascal VOC XML Format [50] vorliegen. Nach der Auswahl einer Model Spezifikation kann das Training mit einem Befehl gestartet werden.

Durch die sehr lange Inferenzzeit und die Ungenauigkeit bei der Unterscheidung zwischen den Klassen erfüllten EfficientDet Lite Models die Anforderungen an das Model nicht. Für eine Auswertung im Backend mit einer GPU könnten grössere EfficientDet Versionen trainiert werden, um die Tauglichkeit zu prüfen.

4.2.3 TensorFlow 2 Object Detection API

Mit der TensorFlow 2 Object Detection API kann eine Vielzahl von Models zur Objekterkennung trainiert werden. In einem Tutorial [51] werden alle nötigen Schritte dazu erklärt. Die 40 verfügbaren vortrainierten Models sind im TensorFlow Detection Model Zoo [52] aufgelistet.

Der Aufwand für das Trainieren eines Models ist sehr hoch. Die Annotationen müssen im Pascal VOC XML Format vorliegen. Für Trainings- und Test Set wird aus den Bildern und den Annotationen jeweils eine binäre *tfrecord* Datei generiert. In einer weiteren Datei muss für jede Kategorie eine ID und ein Name definiert werden. Das gewünschte Model wird anschliessend von dem Model Zoo heruntergeladen und extrahiert. Neben dem Model Checkpoint enthalten die heruntergeladenen Dateien eine Konfigurationsdatei. In der Konfigurationsdatei wird das gesamte Model inklusive Hyperparameter abgebildet. Abhängig vom Model müssen sehr viele Anpassungen getätigt werden, damit das Training funktioniert. Nach dem Starten des Trainings muss parallel dazu die Validierung mit dem Test Set gestartet werden. Mit der Applikation TensorBoard [53] kann der Fortschritt beobachtet werden.

Mehrere unterschiedliche Models wurden mit dem Flower- und Pollinator Datenset trainiert und verglichen. Die besten Ergebnisse wurden mit einem SSD-MobileNet-v2 [27] Model mit Bildgrösse 640x640 Pixel erzielt. Verwendet wurde die erste Version des Flower Datensets, worin alle Blumen in einer Klasse zusammengefasst sind. Nach 20'000 Epochen erreichte die Average Precision den Wert von 0.655.

Für den Export des Models als TFLite Datei sind mehrere Schritte nötig, da die Metadaten nicht automatisch hinzugefügt werden. Die Inferenzgeschwindigkeit auf dem Raspberry Pi betrug rund zwei Sekunden.

Der grösste Nachteil der Models der TensorFlow Object Detection API liegt in dem hohen Aufwand und der Komplexität des Trainings. Dadurch wird die Erweiterung des Models um neue Klassen oder mit einem grösseren Datenset massiv erschwert.

4.2.4 YOLOv5

YOLOv5 [30] basiert auf dem PyTorch Framework und gilt als sehr benutzerfreundlich. Es gibt 10 Grössen von vortrainierten Models. Die Parameter der für das Projekt relevanten Versionen sind in Tabelle 3 dokumentiert.

Version	Bildgrösse in Pixel	Model Grösse	mAP COCO 2017
YOLOv5n	640x640	3.87MB	28.0
YOLOv5s	640x640	14.1MB	37.4
YOLOv5m	640x640	40.8MB	45.4
YOLOv5l	640x640	89.3MB	49.0
YOLOv5n6	1280x1280	6.86MB	36.0
YOLOv5s6	1280x1280	24.8MB	44.8

Tabelle 3 YOLOv5 Model Versionen [30]

Das Trainieren eines YOLOv5 Models ist sehr einfach und schnell. Die Annotationen werden in Textdateien gespeichert. Pro Bild wird in einer gleichnamigen Textdatei jedes Objekt in einer Zeile, bestehend aus Klasse, Mittelpunkt , Breite und Höhe, definiert. In einer YAML Datei wird der Pfad zu den Trainings- und Testdaten und die Namen der vorhandenen Klassen definiert. Für das Starten des Trainings muss neben dem YAML File die Model Version ausgewählt werden. Nach jeder Epoche wird das Ergebnis validiert. Der Fortschritt kann mit TensorBoard oder mit dem Logging Tool Weights & Biases [54] verfolgt werden. Die gesamte Architektur des Models und die Hyperparameter werden in YAML Files gespeichert und sind gut dokumentiert. Wahrend des Trainings werden die Resultate der Validierung nach jeder Epoche mit den restlichen verglichen und ein Snapshot des Models aus der besten Epoche gespeichert. Zu jedem Zeitpunkt sind zwei Model Files, best.pt und last.pt, verfügbar. Wenn das Model nach einer definierten Anzahl Epochen nicht mehr verbessert wird, stoppt das Training automatisch.

Im Vergleich zu TFLite-Model-Maker oder der TensorFlow Object Detection API ist der Prozess des Entwickelns eines Models signifikant einfacher. Die Trainingszeit ist im Vergleich um ein Vielfaches kürzer. Die Models lassen sich in alle gängigen Formate exportieren. Ein Export für die Inferenz auf einem Edge TPU Accelerator wird auch unterstützt.

Ein Model der Grösse Small wurde mit einem Pollinator Datenset trainiert, in welchem alle Bestäuber in einer Klasse aufgeführt sind. Nach knapp 200 Epochen wurde eine Average Precision von 0.69 erreicht. Durch die eingebaute Data Augmentation wurde sehr schnell eine hohe Genauigkeit erreicht. Die Inferenzzeit auf dem Raspberry Pi 4 betrug knapp zwei Sekunden bei einem Model der Grösse Small.

YOLOv5 erfüllt alle Anforderungen und hebt sich hauptsächlich in der Benutzerfreundlichkeit von EfficientDet mit TFLite-Model-Maker und der TensorFlow Object Detection API ab. Eine Erweiterung des Models um neue Klassen oder ein grösseres Datenset ist mit YOLOv5 ohne grossen Aufwand realisierbar.

4.3 Entwicklung der Models

Die Entwicklung der Models wurde iterativ durchgeführt. Jede neue Version der Datensets wurde mit verschiedenen Modelgrössen trainiert und mit den vorherigen Versionen verglichen.

4.3.1 Training

Die Trainings wurden auf Google Colab [55] und Kaggle [56] durchgeführt, da die bereitgestellten Tesla P100 GPU mit 16GB GPU Memory den Trainingsprozess stark beschleunigen. Damit längere Trainings möglich sind, wurde ein Upgrade auf Google Colab Pro durchgeführt. Alle Experimente wurden auf der Logging Plattform Weights & Biases festgehalten, um sie untereinander zu vergleichen. Als Hauptindikator für die Bewertung der Genauigkeit wurde der mAP.5:.95 gewählt, da er über alle Klassen hinweg die Precision und auch der Recall berücksichtigt. Für die Pollinator Models wurde zusätzlich der Class Loss berücksichtigt, um die Bestäuber möglichst präzise zu kategorisieren. Im Flower Datenset wurden nur Blumen annotiert, die blühen. Eine klare Grenze zwischen «guten» Blumen und verwelkten Blumen gibt es nicht. Die Grenzwertigen Blumen, welche im Datenset nicht annotiert wurden, erkennt das Model zu einem gewissen Prozentsatz trotzdem als Blume, was den mAP Wert der Klasse Daisy beschränkt.

Das Datenset wurde im Verhältnis 80:20 in Trainings- und Test Set unterteilt. Die Zusammensetzung der Sets wurde zufällig generiert. Auf ein Validierungsset wurde verzichtet, um das Training mit möglichst vielen Daten durchzuführen.

Das Jupyter Notebook, womit die Models trainiert wurden, ist auf GitHub im Repository P8-Tools [38] abgelegt. Die meisten Models wurden auf Google Colab trainiert. Bei den ersten Versionen der Models ist vermehrt die Situation aufgetreten, dass die Verbindung zum Interface von Colab unterbrochen wurde und alle Daten des aktuellen Trainings verloren gingen, da ein erneuter Zugriff während einer laufenden Sitzung nicht möglich war. Um diese Situation zu verhindern wurde eine automatische Synchronisierung des Ausgabeordners mit einem Google Drive Ordner implementiert. Die Synchronisierung wird alle 30 Sekunden ausgelöst und hat zuverlässig funktioniert.

4.3.2 Hyperparameter Evolving

Für das Training von Models gibt es, je nach Typ, unterschiedliche variable Hyperparameter. Bei YOLOv5 sind sie standardmässig auf das COCO Datenset [4] optimiert. Da die Flower- und Pollinator Datensets über andere Charakteristiken verfügen, wurden die besten Hyperparameter für den Trainingsvorgang mittels Evolving bestimmt. Mehrere Trainings mit jeweils 10 Epochen wurden mit verschiedenen Kombinationen von angepassten Hyperparametern durchgeführt und evaluiert. Um die Ergebnisse zu vergleichen, muss eine Formel für die Berechnung der Fitness des Models bestimmt werden. Die Parameter mAP.5, mAP.5:.95, Precision, Recall, Object Loss, Class Loss und Box Loss werden gewichtet und addiert bzw. subtrahiert. Die Gewichtungen müssen dem Zweck der Anwendung angepasst werden. Abbildung 69 visualisiert die Metriken und eine daraus berechnete Fitness.

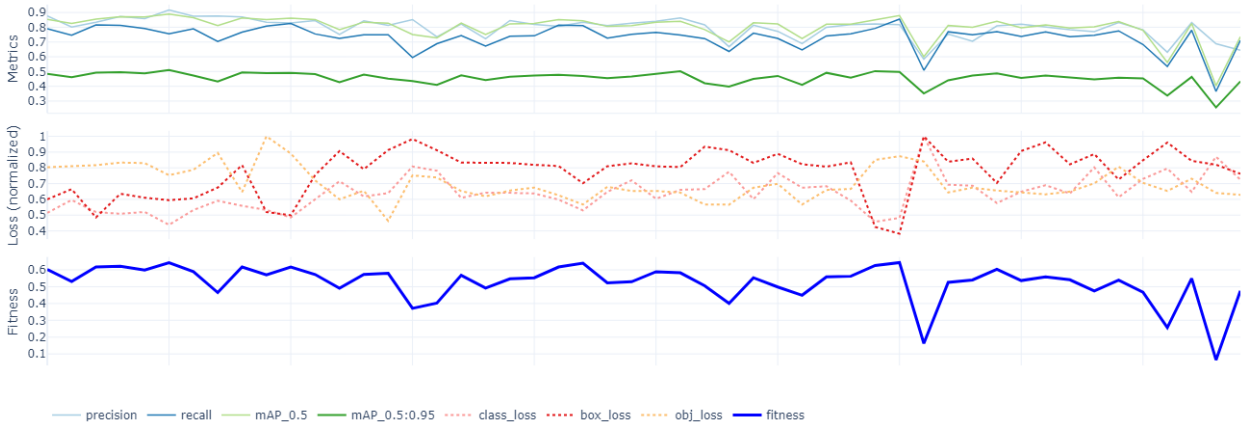


Abbildung 69 Hyperparameter Evolving: Metriken zur Berechnung der Fitness

Die Funktion zur Berechnung der Fitness des Pollinator Models ist in Codeausschnitt 1 ersichtlich. Der Parameter mAP.5:.95, welcher die Genauigkeit des Models gut repräsentiert, ist am höchsten gewichtet. Für eine gute Unterscheidung der Klassen ist ein niedriger Class Loss wichtig, der mit 0.3 gewichtet wird.

```
def fitness(map_05, map_05_95, precision, recall,
            class_loss, box_loss, obj_loss):
    weight_map_05 = 0.1
    weight_map_05_95 = 0.9
    weight_precision = 0.1
    weight_recall = 0.1
    weight_class_loss = 0.3
    weight_box_loss = 0
    weight_obj_loss = 0
    fitness = (
        weight_map_05 * map_05
        + weight_map_05_95 * map_05_95
        + weight_precision * precision
        + weight_recall * recall
        - weight_class_loss * class_loss
        - weight_box_loss * box_loss
        - weight_obj_loss * obj_loss
    )
    return fitness
```

Codeausschnitt 1 Berechnung der Fitness

Abbildung 70 visualisiert Auswirkungen der unterschiedlichen Hyperparameter auf die Fitness des Models. Die vertikale Achse zeigt die Fitness, die horizontale Achse die Werte der Parameter. Anhäufungen von Punkten werden mit einer höheren Farbintensität visualisiert. Die Werte der besten Epoche sind mit einem Fadenkreuz markiert.

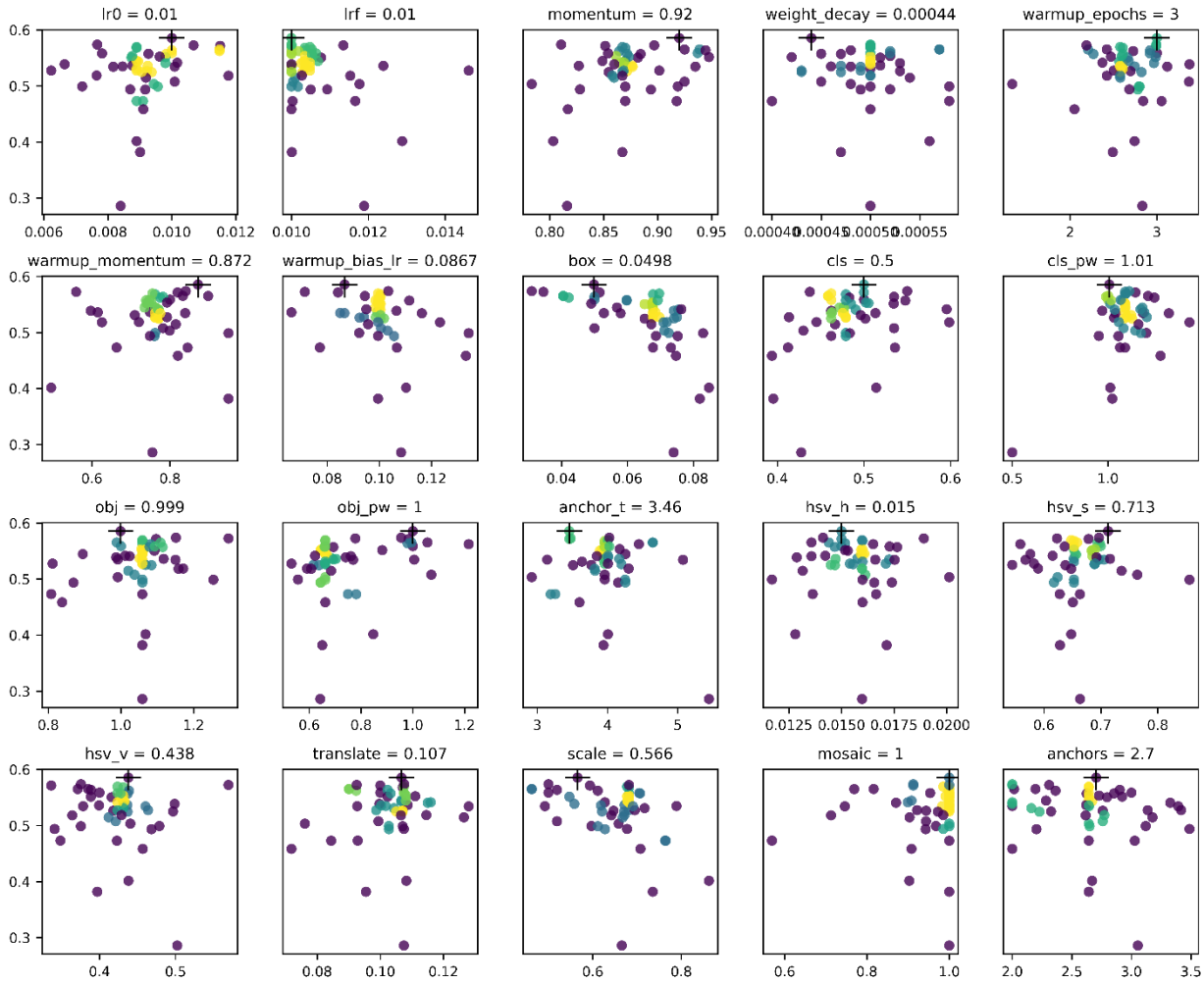


Abbildung 70 Hyperparameter Evolving Yolov5m Pollinator

Das Hyperparameter Evolving ist sehr zeitaufwändig. Die daraus resultierenden Hyperparameter unterscheiden sich nur minimal von den Standardwerten. Der Wert der gefundenen optimalen Learning Rate ist nach 10 Epochen nicht sehr aussagekräftig. Das Endresultat wurde durch das Evolving in manchen Fällen leicht verbessert. Oft hat die Modifizierung der Hyperparameter dazu geführt, dass die Genauigkeit in den ersten Epochen stark anstieg, danach aber stagnierte. Abbildung 71 zeigt die mAP Werte von zwei Flower Models der selben Grösse mit den Standardparametern und «optimierten» Parametern.

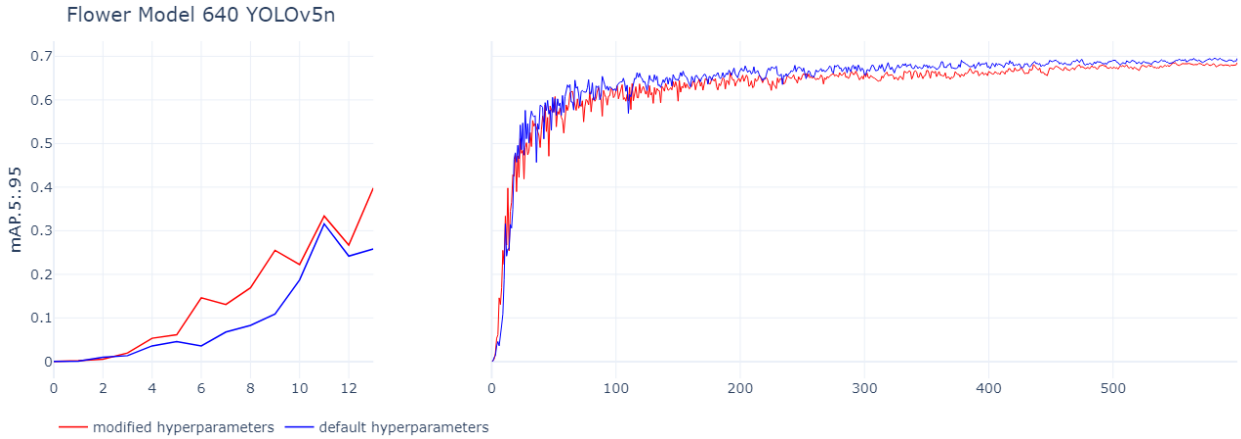


Abbildung 71 Vergleich Standard Hyperparameter und modifizierte Hyperparameter

In der rechten Visualisierung ist erkennbar, dass das Model mit den modifizierten Hyperparametern schneller einen höheren mAP Wert erreicht. Auf den gesamten Trainingsvorgang betrachtet schneiden die Standardwerte besser ab.

4.3.3 Quantisierung

Als Quantisierung von Models bezeichnet man eine Konvertierung der Parameter in kleinere Datentypen, um die Modelgrösse zu verkleinern und höhere Geschwindigkeiten zu erreichen. Der Nachteil einer Quantisierung ist die Abnahme der Genauigkeit [57]. Für gewisse Interpreter und Hardwarebeschleuniger wie der Coral EdgeTPU Accelerator wird eine Quantisierung vorausgesetzt, da nur bestimmte Datentypen unterstützt werden [58].

4.3.4 Resultate Flower Models

Insgesamt 25 Flower Models mit unterschiedlichen Bildgrössen, Modelgrössen, Hyperparameter und sonstigen Optionen wurden trainiert. Total wurden knapp 34 Stunden Berechnungszeit mit GPU für das Training der Models beansprucht. Die fünf besten Flower Models werden in Abbildung 72 verglichen. Der höchste mAP.5:.95 Wert hat ein Model der Grösse n6 mit Bildgrösse 1280 erreicht, am kleinsten ist er bei einem Model der Grösse n mit Bildgrösse 480. Die Grösse der Model Files liegen zwischen 3.8 und 14.5 MB. Das Ziel des Flower Models ist es, möglichst alle Blumen auf einem Bild zu detektieren. Die Ausschnitte der Blumen werden vom Pollinator Model weiter ausgewertet, womit das Auftreten von False Positives das Endresultat nur in der Verarbeitungszeit beeinflusst. Das Model mit dem höchsten Recall Wert hat eine Bildgrösse von 640 und eine Model Version der Grösse n.

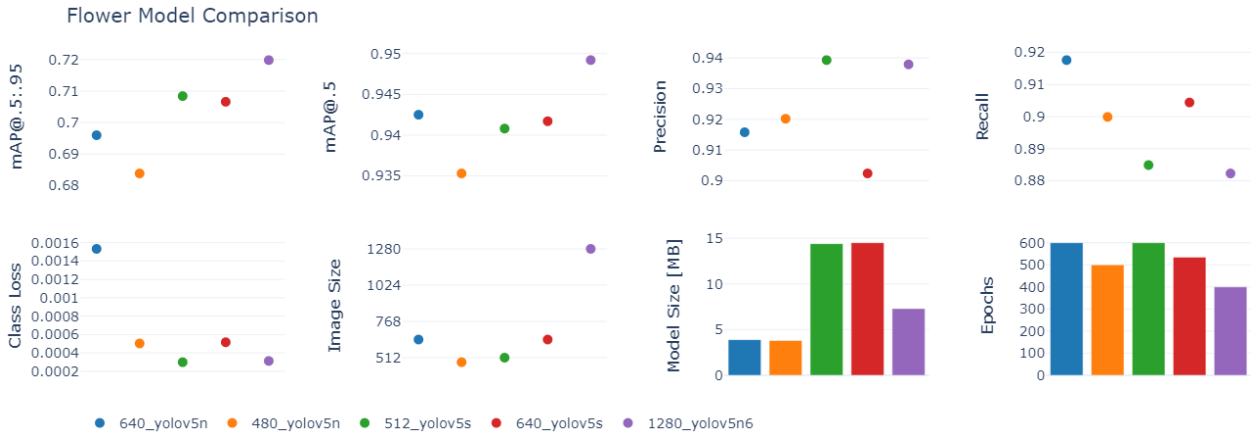


Abbildung 72 Vergleich Flower Models

Die Models wurden an zufälligen Bildern der ersten Fallstudie getestet und haben gute Resultate erbracht. Bis auf wenige Ausnahmen wurden alle blühenden Blumen mit der korrekten Klasse erkannt. Verblühte Blumen wurden teilweise als Blumen erkannt, wenn gewisse optische Merkmale einer blühenden Blume vorhanden waren. Ähnliche Blumen aus der Umgebung, die nicht trainiert wurden, detektierten die Models grösstenteils nicht. Die Models erfüllen den Zweck, möglichst viele der Blumen auf einem Bild mit der korrekten Klasse zu erkennen.

Die Performance der Models wurde auf einem Raspberry Pi 4 verglichen, indem die Models am Test Set validiert wurden. Die Validierung wurde jeweils für eine nicht quantisierte PyTorch Version, eine quantisierte INT8 Version und eine INT8 quantisierte EdgeTPU Version analysiert. Die Ergebnisse der Models 1280_n6, 512_s und 640_n sind in Abbildung 73 visualisiert. Die Genauigkeiten der nicht quantisierten Models stimmen mit denen der Validierung nach dem Training überein. Die kürzeste Inferenzzeit, mit und ohne Quantisierung, hat das Model der Grösse n. Auf dem Raspberry Pi 4 hat eine INT8 Quantisierung der Models die Inferenzzeit in jedem Fall minimal vergrössert. Die Inferenz auf der TPU hat die Zeiten um mindestens 60% verringert.

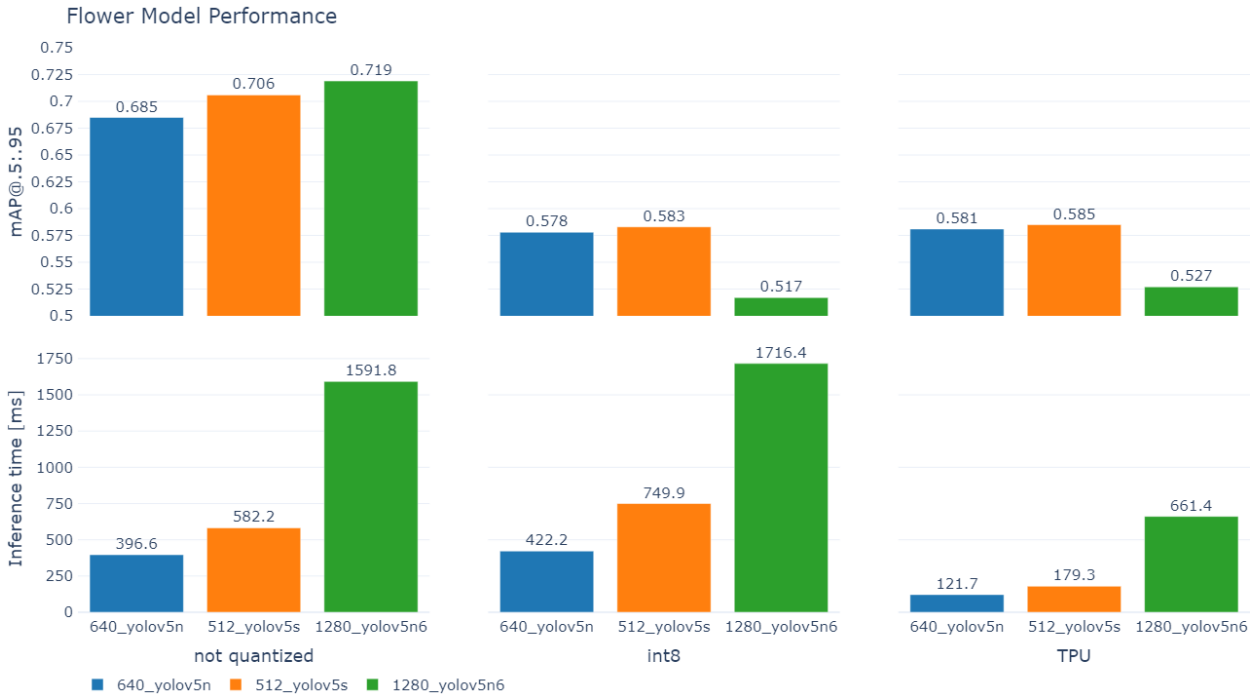


Abbildung 73 Genauigkeit und Inferenzzeiten Flower Models Raspberry Pi 4

Auffällig ist, dass die Genauigkeit bei quantisierten Models stark abnimmt. Das Model mit Bildgrösse 1280, welches ohne Quantisierung die höchste Genauigkeit erreicht, büsst den mAP.5:.95 um 0.2 ein. Eine genauere Analyse von den Ergebnissen der Validierung haben gezeigt, dass die Margariten sehr schlecht detektiert werden und rund 50 % der detektierten Margariten False Positives sind. Somit sind TPU Models für die Erkennung von Blumen nicht brauchbar.

4.3.5 Resultate Pollinator Models

43 verschiedene Versionen von Pollinator Models wurden in einer Gesamtzeit von gut 160 Stunden trainiert. Abbildung 74 vergleicht die besten Models, die jeweils fünf Klassen von Bestäubern unterscheiden können. Den höchsten mAP.5:.95 Wert von 0.6613 hat ein Model der Grösse m mit Bildgrösse 640 erreicht. Das Model wurde mit evaluierten Hyperparametern trainiert. Minimal ungenauer ist ein Model der Grösse s mit Bildgrösse 480. Für die Erkennung der Bestäuber ist eine hohe Precision wichtig, um False Positives möglichst klein zu halten. Für die Metriken Precision und Recall hat ein Model der Grösse m mit Bildgrösse 416 am besten abgeschnitten. Das grösste Model, der Grösse l mit einer Model File Grösse von 354 MB hat, über alle Metriken betrachtet, eine mittlere Genauigkeit und einen sehr hohen Class Loss.

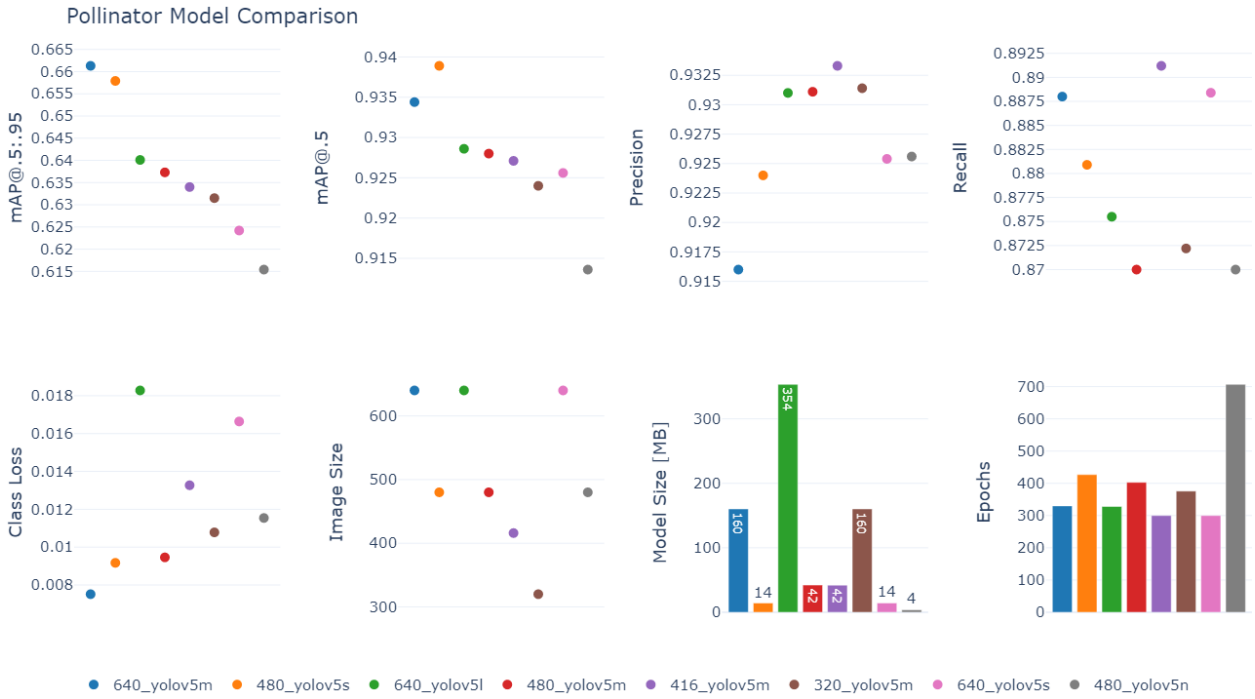


Abbildung 74 Pollinator Models Vergleich

Es ist wichtig, die Ergebnisse der Validierung nicht als einzige Quelle der Wahrheit zu betrachten, da das Test Set nur ein kleiner Ausschnitt der Daten ist, der immer einen Bias aufweist und die gesamten Daten eventuell nicht umfänglich repräsentiert. Die Models wurden an zufälligen Bildern der ersten Fallstudie ausgetestet, um mögliche Anfälligkeiten zu erkennen.

Für das Flower Model ist die Unterscheidung zwischen den einzelnen Klassen aufgrund der optischen Merkmale sehr einfach. Um Schwächen der Models zu erforschen, wurden über 60'000 Bilder ausgewertet und die Resultate analysiert. Die Klassifizierung von Bestäubern gestaltet sich, bedingt durch die Wahl der Klassen, schwieriger. Die Klasse Wildbiene umschliesst, mit Ausnahme der Honigbiene und der Hummel, die gesamte Überfamilie Apoidea. Die Bestäuber der Klasse Wildbiene verfügen über sehr unterschiedliche optische Merkmale, was eine Klassifizierung erschwert und False Positives zur Folge hat. Gelegentlich wurden Honigbienen oder Aufnahmen von Schwebfliegen aus kurzer Entfernung als Wildbienen erkannt. Mit einem niedrigen Confidence Threshold werden vermehrt Bestäuber ähnliche Objekte als Bestäuber erkannt, die auch von Menschen nicht einfach von Bestäubern zu unterscheiden sind. Da diese Objekte auf einer Vielzahl von aufeinanderfolgenden Aufnahmen ersichtlich sind, stechen diese Entdeckungen bei der Analyse der Resultate heraus. Die Hummeln im Datenset sind vorwiegend dunkel. Es wurde beobachtet, dass Knospen von Flockenblumen als Hummeln erkannt wurden.

4.4 Analyse der Models

Während dem Training lernt ein Model Features von Objekten. Mit dem Class Activation Mapping (CAM) Verfahren [59] können diskriminante Regionen durch die Berechnung einer Class Activation Map gefunden werden. Für jedes Pixel im Bild wird die Relevanz in der letzten Faltungsschicht berechnet. Abbildung 75 zeigt eine Class Activation Map des Flower Models an einem Bild zwei Flockenblumen. Die grösste Relevanz ist klar in den Regionen, wo sich die Blumen befinden. CAM kann hilfreich sein, um herauszufinden, welche Features das Model lernt, um Objekte zu erkennen. Auffällig ist, dass die Knospe einer Flockenblume auch als potentiell relevant erkannt wurde. Daraus lässt sich schliessen, dass das Model nicht nur die violetten Blätter der Blumen, sondern auch die grünen Blätter unterhalb der Blume betrachtet.

Abbildung 75 Class Activation Map Flockenblume

Mit CAM wird nur der letzte Layer berechnet. Grad-CAM [60] ist eine Erweiterung von CAM unter Einbeziehung von Gradienteninformationen. Für jeden Layer im Model wird der Einfluss von einzelnen Features auf den letzten Layer bestimmt. Abbildung 76 zeigt eine Grad-CAM der zweiten Schicht eines Flower Models. Das Model führt simple Faltoperationen durch, die für das Endergebnis relevant sind. Auf der linken Seite werden die Spitzen an den Blättern der Blumen erkannt. Auf der rechten Seite wird ein Bottom Sobel durchgeführt, womit die Mittelpunkte hervorgehoben werden. Durch die Kombination von allen relevanten Regionen werden vom Model die Endergebnisse berechnet.

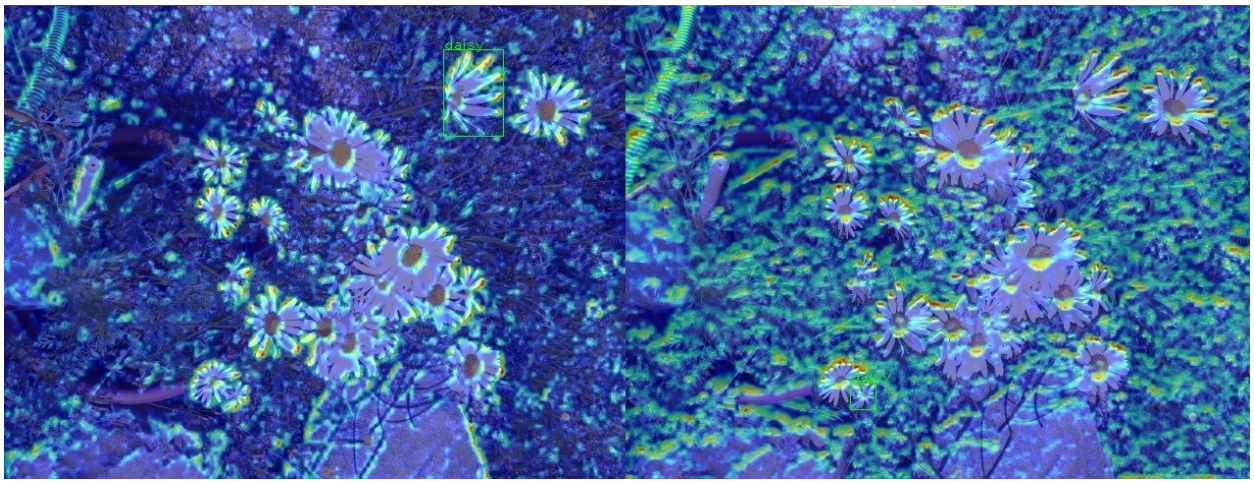


Abbildung 76 Grad-CAM Beispiel Margariten

Ein Layer in der Mitte eines Pollinator Models mit einer trainierten Klasse ist in Abbildung 77 ersichtlich. In der linken Visualisierung wird der Thorax der ersten Biene und der Hinterleib der zweiten Biene erkannt. Auf der rechten Seite wird ein Bein der ersten Biene und die Kante des

Flügels der zweiten Biene erkannt. Spezifische Features wie Beine, Texturen und Farbe werden von dem mehrklassigen Model für die Unterscheidung zwischen Klassen eingesetzt.

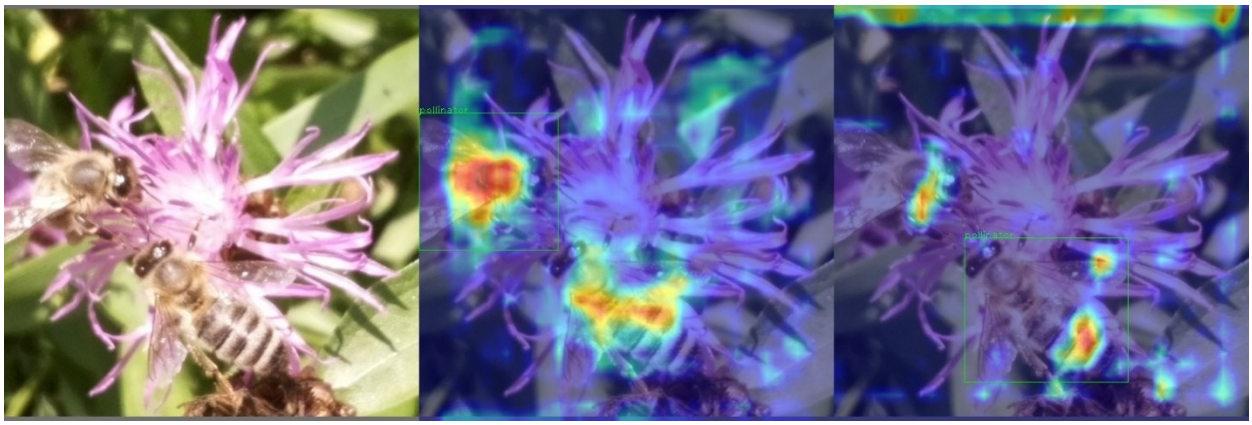


Abbildung 77 Grad-CAM Beispiel Bestäuber

Für die Berechnungen der CAM und GradCAM Layer wurde YOLO-V5 GRADCAM [61] eingesetzt.

4.5 Inferenz

Als Inferenz wird die Durchführung der Detektion von Objekten auf einem Bild mit einem Machine Learning Model bezeichnet.

4.5.1 Konfiguration

Für die Inferenz können verschiedene Parameter konfiguriert werden. Für die Durchführung werden das Model File, auch als Weights File benannt, und die zu verarbeitende Billdgrösse benötigt. Abhängig vom Format sind die Klassen mit Namen oder als ID in dem Metadaten enthalten. Für die Inferenz wurde jeweils ein Parameter für die Namen der Klassen implementiert, um die Benennung nicht vom Model File abhängig zu machen.

Der Confidence Threshold definiert eine Grenze des Scores, ab welchem ein Objekt als Detektion gilt. Wird er höher gewählt, beinhalten die Resultate weniger False Positives und mehr False Negatives. Es ist ein Trade-Off zwischen dem Erkennen von möglichst vielen potentiellen Objekten und einer möglichst genauen Erkennung.

Der IoU Threshold definiert den Grenzwert der Überlagerung, ab welchem zwei Objekte als unterschiedliche Objekte erkannt werden. Je höher der IoU Threshold gewählt wird, desto grösser ist die Chance, dass ein Objekt mehrfach erkannt wird. Wenn der IoU Threshold sehr klein gewählt wird, kann es sein, dass ein Objekt, das nahe bei einem anderen liegt, nicht erkannt wird.

Mit der maximalen Anzahl von Detektionen wird der Output limitiert. Bei TFLite Models liegt dieser Wert standardmässig bei 25. Für die Erkennung der Bestäuber kann der Wert kleiner gewählt werden.

Der Parameter Margin beschreibt einen Rahmen, der den Objekten beim Ausschneiden hinzugefügt wird. Auf den Bildern mit Flockenblumen befinden sich die Bestäuber oftmals nicht komplett auf der Blume. Wenn das Flower Detection Model die Blume bündig ausschneidet, wird der Bestäuber abgeschnitten. Durch das Hinzufügen eines Rahmens wird die Bounding Box um die definierte Anzahl Pixel vergrössert. Falls der Margin zu gross gewählt wird, kann es vorkommen, dass ein Bestäuber auf mehreren Ausschnitten von unterschiedlichen Blumen erkannt wird.

Bei einer Inferenz mit einem YOLOv5 Model in PyTorch können die auszuwertenden Bilder mit Test-Time Augmentation [62] künstlich vermehrt werden, um möglicherweise mehr Objekte zu erkennen. Die Inferenzzeit wird dadurch wesentlich erhöht.

Beim NMS wird jedem Objekt die Klasse mit dem höchsten Score zugewiesen und der Rest wird unterdrückt. Die Multi-Label Option ermöglicht es, den NMS pro Klasse anzuwenden und ein Objekt mehrmals als unterschiedliche Klasse zu erkennen. Für das Mapping von mehreren Resultaten auf ein Objekt wurde eine Funktion entwickelt, die im folgenden Unterkapitel beschrieben wird.

4.5.2 Multi Class Detections

Beim NMS wird für jedes Objekt genau eine Klasse vorgeschlagen. Falls ein Bestäuber mit einer Wahrscheinlichkeit von 0.85 als Honigbiene erkannt wird, zu 0.84 aber eine Wildbiene sein könnte, wird diese Information nicht preisgegeben. Durch die Multi-Label Option in der PyTorch Implementation von YOLOv5 ist es möglich, mehrere überlappende Bounding Boxen mit unterschiedlichen Klassen zu erkennen. Um festzustellen, ob das Model für das selbe Objekt mehrere Klassen vorschlägt, wurde eine Funktion zur Überprüfung entwickelt. Zwischen allen Bounding Boxen wird der Grad der Überlappung anhand des IoU berechnet. Es entsteht ein direktionaler Graph. Ein Beispiel mit sechs Boxen ist in Abbildung 78 visualisiert.

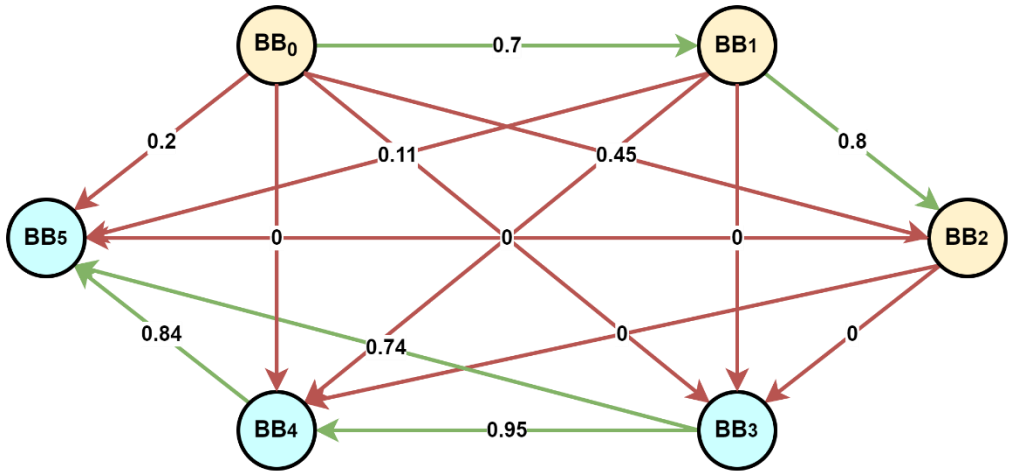


Abbildung 78 Beispiel Multilabel IoU Abhängigkeiten

Für die Bestimmung, ob zwei Bounding Boxen das selbe Objekt markieren, wird ein IoU Threshold festgelegt. Im Beispiel in Abbildung 78 liegt er bei 0.65. Alle Verbindungen, die über dem Threshold liegen, sind grün markiert, die restlichen rot. BB_0 hat eine grüne Verbindung zu BB_1, BB_1 hat eine grüne Verbindung zu BB_2. Die ersten drei Bounding Boxen zeigen also auf das selbe Objekt im Bild. BB_3 hat eine grüne Verbindung zu BB_4 und zu BB_5, BB_4 hat eine zu BB_5, womit klar ist, dass alle drei das selbe Objekt markieren. Im Bild befinden sich also zwei Objekte mit jeweils drei möglichen Klassen.

5 Automatisierung des Auswerteverfahrens

Das im Rahmen des Forschungsprojekts Mitwelten entwickelte Bilderfassungssystem ist in Abbildung 79 visualisiert. Als Kameras werden über Ethernet mit PoE angeschlossene Raspberry Pi 3 B+ mit Kameramodulen eingesetzt. Mit einem HTTP Request kann eine Aufnahme ausgelöst und das Bild heruntergeladen werden. Kontrolliert wird das System vom AP Gateway, welcher von einem Raspberry Pi 4 mit angeschlossener 2TB Harddisk kontrolliert wird. Der AP Gateway löst in regelmässigen Intervallen Aufnahmen aus und speichert diese auf der Disk. Ein automatischer Upload der Daten in das Backend ist in Entwicklung. In folgenden Abschnitten wird für die Raspberry Pi in den Kameras der Begriff Kamera und für das Raspberry Pi im AP Gateway den Begriff AP Gateway verwendet.

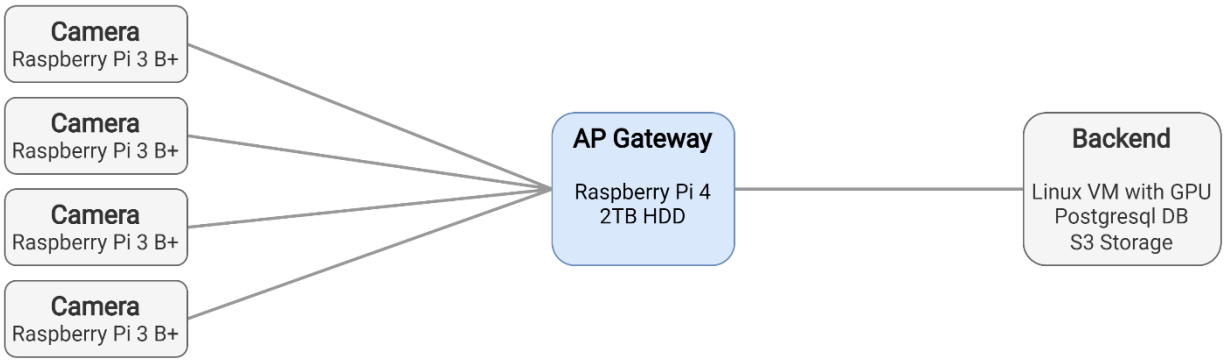


Abbildung 79 Diagramm bestehendes Bilderfassungssystem Mitwelten

Es wurde beschlossen, dass keine Daten auf den Kameras gespeichert werden. Dieser Beschluss basiert auf der einfacheren Handhabung des Datenschutzes und der Verkürzung der Lebensdauer einer SD Karte bei Schreibvorgängen.

Auf den Kameras und dem Raspberry Pi im AP Gateway läuft das 32 Bit Raspberry Pi OS Lite (Buster) [63]. Für die Bereitstellung der Aufnahmen läuft auf den Kameras MJPG-Streamer [64]. Die Automatisierung der Auswertung soll, sofern es möglich ist, mit der bestehenden Infrastruktur kompatibel sein.

PyTorch unterstützt aktuell nur 64 Bit Raspberry Pi OS [65]. Eine Implementierung auf 32 Bit Betriebssystemen ist gemäss Beiträgen in Foren praktisch unmöglich, da gewisse Abhängigkeiten nicht erfüllt werden [66] [67] [68]. Für die Implementierung der Inferenz auf Raspberry Pi mit 32 Bit OS kann das Framework TensorFlow Lite [69] eingesetzt werden, wobei im Vergleich zu PyTorch grössere Inferenzzeiten resultieren.

Die Auswertung der Bilder besteht aus dem Erkennen der Blumen und dem Erkennen von Bestäubern auf den Ausschnitten. Abbildung 80 zeigt vier Ansätze unter Verwendung unterschiedlicher Hardware.

1. Die gesamte Auswertung wird direkt auf den Kameras durchgeführt.
2. Auf den Kameras werden die Blumen erkannt und ausgeschnitten, im AP Gateway werden Bestäuber erkannt.
3. Die gesamte Auswertung wird im AP Gateway durchgeführt.
4. Die gesamte Auswertung wird im Backend durchgeführt.

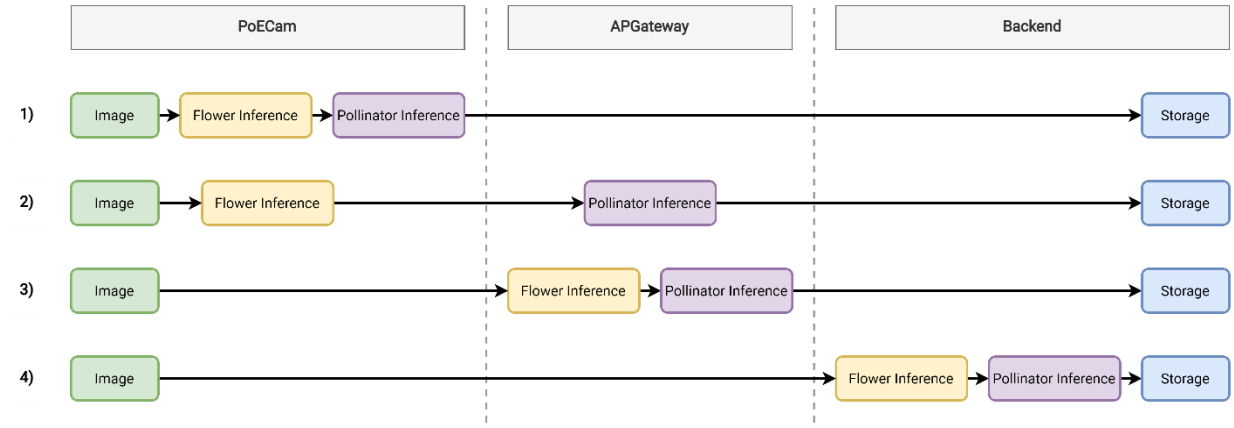


Abbildung 80 Konzepte für die Automatisierung der Auswertung

Die Komponenten der jeweiligen Ansätze sollen möglichst modular sein, um Kombinationen zu ermöglichen. Die vier Versionen der Automatisierung wurden entwickelt und getestet. Die Kamera wurde simuliert, um bei allen Tests die selben Bilder zu verarbeiten. Für die Simulation wurde der *mock-mjpg-streamer* entwickelt, der Bilder aus einem Ordner liest und bereitstellt. Der Source Code ist auf GitHub im Repository P8-Tools [38] abgelegt. Insgesamt 236 Testbilder wurden zufällig aus den Aufnahmen der ersten Fallstudie gewählt. Im Durchschnitt zeigt ein Bild im Test Set 5.45 Blumen.

5.1 Datenformate

Die Resultate der Auswertung werden im JSON Format gespeichert und übermittelt, um die Abhängigkeiten möglichst gering zu halten. Es wurde eine standardisierte Struktur festgelegt, in welcher die Nachrichten vorliegen müssen. Sie wird unterteilt in die Objekte *detections* und *metadata*.

Die Detektionen bestehen aus Listen für Flowers und Pollinators. Pro erkannte Blume wird ein Index, die Klasse, der Score und die Ausschnittgrösse aufgeführt. Ein Bestäuber Objekt besteht aus einem Index, dem Index der Blume, auf welcher er gefunden wurde, der Klasse, dem Score und einem Bildausschnitt, das Base64 encodiert ist. Falls die Multi-Class Option angewendet wird, können mehrere Bestäuber den selben Index besitzen.

Die Metadaten beinhaltet die ID der Kamera, mit welcher das Bild erfasst wurde, den Aufnahmezeitpunkt, Originalbildgrösse und Informationen über die beiden Inferenzschritte. Die Metadaten der Inferenz umfassen:

- Modelname
- IoU- und Confidence Threshold
- Angewandte Optionen wie Multi-Class oder Inference-Time-Augmentation
- Margin
- Die für die Auswertung benötigte Zeit

Um die durchschnittliche Grösse eines Result Files zu ermitteln wurden knapp 50'000 Bilder mit gut erkennbaren Blumen ausgewertet. Insgesamt wurden knapp 250'000 Blumen und 7'500 Bestäuber erkannt. Die resultierende Durchschnittsgrösse beträgt 2.46 kB.

5.2 Gesamte Auswertung auf der Kamera

Für die komplette Auswertung der Bilddaten auf der Kamera wurde eine Applikation entwickelt, die auf dem TFLite Framework basiert und kompatibel mit der 32 Bit Version von Raspberry Pi OS ist. Der Code ist im GitHub Repository RPiCamPollinatorDetection [70] abgelegt. Es werden YOLOv5 Models im TFLite Format mit FP16 Quantisierung, die beim Export standardmässig angewandt wird, unterstützt. Ein Flowchart der Anwendung ist in Abbildung 81 ersichtlich.

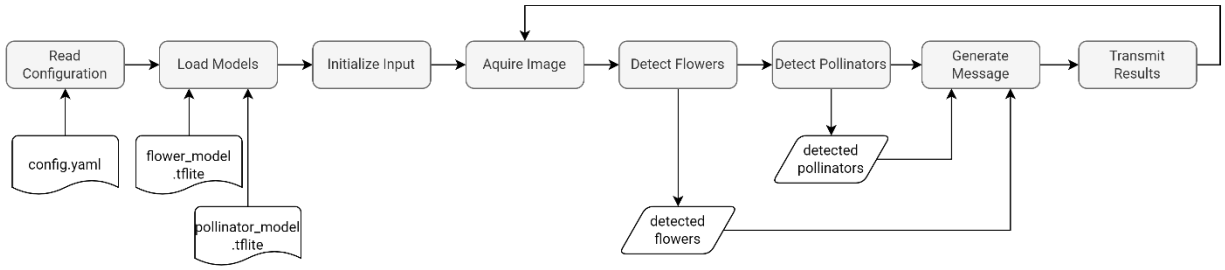


Abbildung 81 Flowchart Gesamte Auswertung auf Kamera

5.2.1 Konfiguration

Die Konfiguration der Applikation wird aus einem YAML File gelesen. Das File ist in drei Abschnitte gegliedert. Der erste Abschnitt beinhaltet die Konfiguration der Models:

- Das Model File
- Die zu erkennenden Klassen
- Die erforderliche Bildgrösse
- Confidence und IoU Threshold
- Ein Margin, der beim Ausschneiden hinzugefügt wird

Im zweiten Abschnitt wird der Input definiert. Es kann zwischen URL und Camera gewählt werden. Wenn die Bilder über den MJPG-Streamer heruntergeladen werden, muss der Endpunkt und optional ein Benutzername / Passwort Paar definiert werden. Falls die Bilder direkt von der Raspberry Pi Kamera aufgenommen werden, muss die Auflösung der Kamera konfiguriert werden.

Der dritte Abschnitt dient zur Konfiguration des Outputs. Die Resultate können via HTTP POST Request an einen Endpunkt übertragen werden. Dafür ist eine URL und optional Benutzername / Passwort zu definieren. Eine weitere Möglichkeit zur Übertragung der Resultate ist die Publizierung an einen MQTT Broker, wobei Broker, Port, Topic, Benutzername und Passwort zu konfigurieren sind. Optional kann die TLS Verschlüsselung deaktiviert werden. Das Topic des MQTT Outputs und die URL des HTTP Outputs können Platzhalter für Filename, Node ID und Hostname beinhalten, die vor dem Übermitteln durch die effektiven Werte ersetzt werden. Somit ist es möglich, die Node ID in ein MQTT Topic zu binden, was saubere Strukturierungen vereinfacht.

Um die Kamera als Standalone System einzusetzen, das offline betrieben wird, kann die Speicherung der Resultate auf der SD Karte oder einem USB Stick gewählt werden.

Zusätzlich zu den drei Abschnitten kann ein *capture_interval* festgelegt werden, der die maximale Erfassungsfrequenz festlegt.

5.2.2 Performance

Die Version mit der gesamten Auswertung auf der Kamera wurde auf einem Raspberry Pi 3 B+ mit einem 32 Bit OS getestet. Um die Wärmeentwicklung zu analysieren wurde das Raspberry Pi, wie die Kameras im Feld, in ein Gehäuse verbaut, welches keine Luftzirkulation zulässt. Die verwendeten Models wurden im TFLite Format exportiert und sind FP-16 quantisiert. Das Flower

Model hat die YOLOv5 Modelgrösse n und eine Bildgrösse von 640 Pixel. Das Pollinator Model hat die YOLOv5 Modelgrösse s und eine Bildgrösse von 480 Pixel.

Die Auswertung der 236 Bilder dauerte 59 Minuten und 30 Sekunden. Die durchschnittliche Auswertungszeit pro Bild beträgt 15.1 Sekunden. Die Zeit der Auswertung teilt sich in Download, Blumen Inferenz und Bestäuber Inferenz auf. Der Download betrug im Durchschnitt 1’514 ms, die Erkennung der Blumen 2’129 ms und die Erkennung der Bestäuber 11'457 ms. Die Dauer der Bestäuber Inferenz pro Ausschnitt beträgt durchschnittlich 2'102 ms.

Abbildung 82 zeigt den Durchsatz, die Zeit pro Auswertung, die Anzahl Detektionen, die Temperatur und Memoryauslastung während dem Test. Die Dauer der Auswertung variiert mit der Anzahl der vorhandenen Blumen. Die Temperatur erreicht mit 80.6 °C sehr hohe Werte, die aber noch nicht zu einer automatischen Reduzierung der Leistung geführt haben. Neben mechanischen Massnahmen kann die CPU Frequenz limitiert werden, um die Temperatur auf einem moderaten Level zu halten. Von den 800 MB Memory wurden maximal 273 MB für die Inferenz gebraucht.

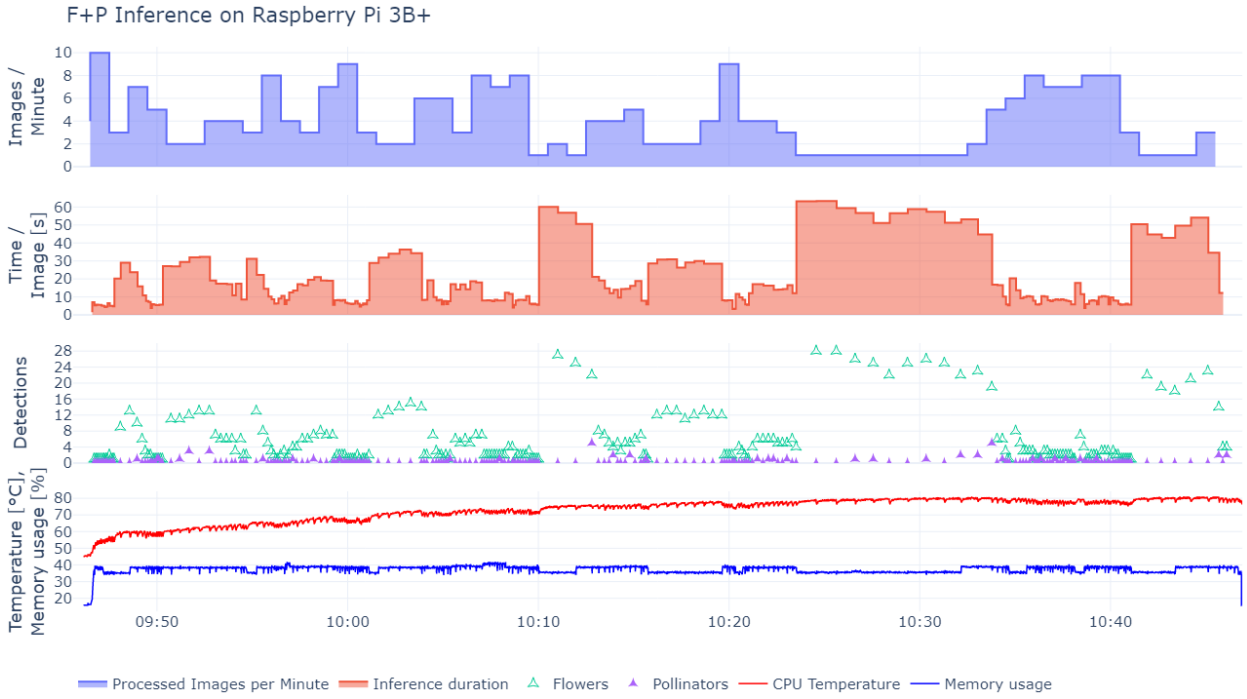


Abbildung 82 Performance Auswertung auf Kamera

Die Verarbeitungszeit eines Bildes ist primär abhängig von der Anzahl sichtbarer Blumen. Abbildung 83 zeigt diese Relation.

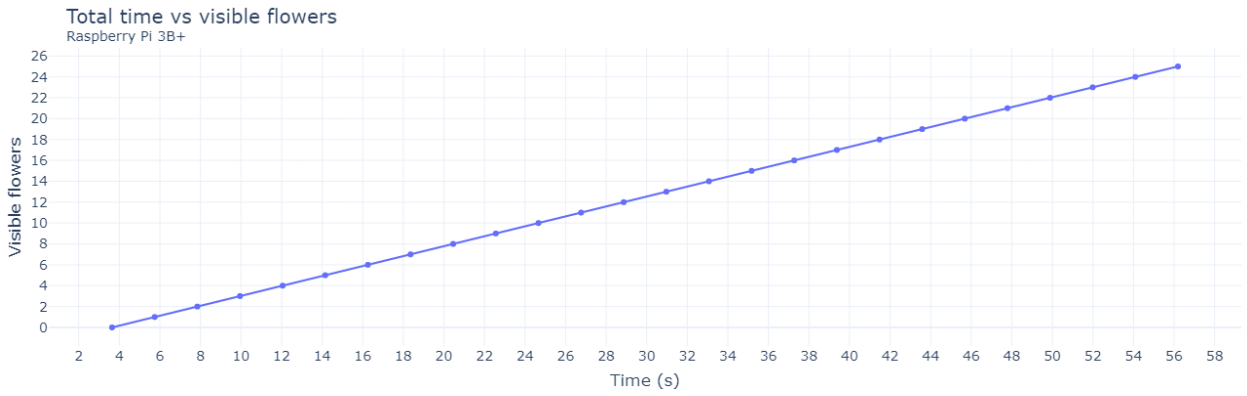


Abbildung 83 Gesamte Auswertung auf Kamera: Verarbeitungszeit vs. Anzahl Blumen

Um einen Aufnahmeintervall von unter 15 Sekunden zu erhalten, dürfen bei der Blumenerkennung maximal fünf Objekte detektiert werden. Diese Anzahl kann durch die Wahl der Blumen, die Einstellung der Kamera oder durch den Parameter *max_detections* beeinflusst werden. Bei der softwareseitigen Limitierung werden nur die Resultate mit den höchsten Wahrscheinlichkeiten beibehalten. Bei knapp 50'000 ausgewerteten Bildern mit gut erkennbaren Blumen wurde eine durchschnittliche Blumenanzahl von 5.16 beobachtet.

5.2.3 Vor- und Nachteile

Ein Vorteil der Auswertung auf der Kamera liegt darin, dass das System alleinstehend betreibbar ist. Es setzt keinen AP Gateway voraus und kann offline betrieben werden. Durch die kleine Grösse der Resultate ist bei einer Speicherung der Daten auf einem USB Stick ein langer Einsatz möglich.

Ein Nachteil ist die relativ hohe Inferenzzeit, falls viele Blumen vorhanden sind. Die Temperatur des SBC steigt ohne Pause zwischen den Auswertungen stark an, was über längere Zeit zu Defekten der Hardware führen kann. Durch das definieren einer Pause zwischen den Auswertungen oder dem Limitieren der CPU Frequenz kann die Temperatur auf einem moderaten Level gehalten werden.

Das Konzept hat trotz der hohen Inferenzzeit potential für den Einsatz im Biodiversitätsmonitoring, da es einfach nachzubauen ist und autonom, offline und ohne Backend über längere Zeit eingesetzt werden kann.

5.3 Auswertung aufgeteilt auf Kamera und AP Gateway

Der zweite Ansatz ist die Aufteilung der Auswertungsschritte auf die Kamera und den AP Gateway. Auf der Kamera werden Blumen auf dem Bild detektiert und ausgeschnitten. Die Ausschnitte, die Arten der Blumen und die Wahrscheinlichkeiten werden anschliessend an den AP Gateway übertragen, welcher Bestäuber auf den Ausschnitten detektiert.

5.3.1 Blumenerkennung auf Kamera

Die Applikation für das Erkennen von Blumen ist eine leicht abgeänderte Version der Applikation für die gesamte Auswertung auf der Kamera. Der Code im GitHub Repository RPiCamPollinator-Detction [70] unter dem Branch *flower-only* abgelegt. Der Ablauf der Applikation ist in Abbildung 84 visualisiert.

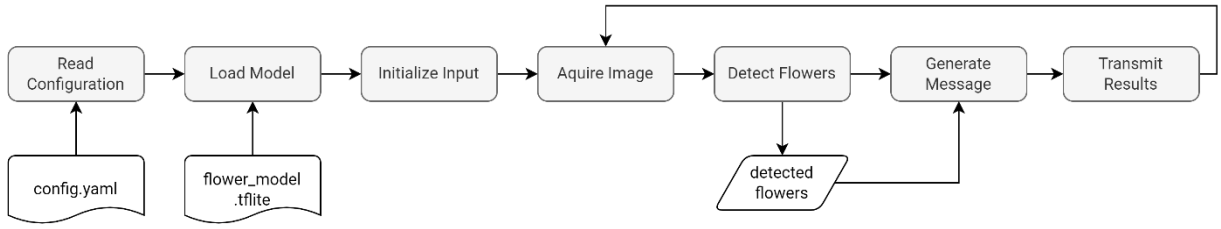


Abbildung 84 Flowchart Flower Inference auf Kamera

Die Konfiguration wird in einem YAML File definiert und ist bis auf die Anzahl Models identisch zu der Konfiguration der Applikation für die komplette Auswertung auf der Kamera. Die Resultate werden mittels HTTP POST Request an den AP Gateway gesendet, wo sie in einer Queue gespeichert werden.

5.3.2 Message Queue auf AP Gateway

Bei einem Aufbau sind standardmässig vier Kameras angeschlossen. Wenn viele Blumen vorhanden sind und alle Kameras in einem Intervall von 15 Sekunden die Resultate übermitteln, ist es nicht möglich, die Bestäuber Inferenz in Echtzeit zu realisieren.

Um die Daten zu buffern wurde ein asynchrones System mit einer Queue entwickelt, die eingehende Daten auf der Disk speichert und für die Inferenz wieder bereitstellt. Der Code ist auf GitHub im Repository ZMQMessageQueue [71] abgelegt. Sie kommuniziert über ZMQ, eine performante, leichte Messaging Library, die keinen zusätzlichen Message Broker voraussetzt [72]. ZMQ unterstützt mehrere Message Patterns, wovon Publish-Subscribe und Request-Reply für die Implementierung genutzt wurde.

Für das Schreiben von Nachrichten in die Queue verbindet sich der ZMQ Subscriber auf den Port eines ZMQ Publishers. Sobald eine Nachricht publiziert wird, wird sie am Ende der Queue angefügt. Ein ZMQ Server stellt die Nachrichten wieder zur Verfügung. Mit einem Request kann ein Client die Nachricht an erster Stelle anfordern. Nach dem Anfordern wird die Nachricht aus der Queue entfernt. Falls sich keine Nachricht in der Queue befindet, wird mit einem Integer mit Wert 0 geantwortet.

Für den Empfang der Resultate der Kameras wurde ein Webserver entwickelt, der auf definierten Routen POST Requests entgegennimmt. Sofern die Payload im JSON Format vorliegt, wird sie von einem ZMQ Publisher verschickt und somit in die Queue geschrieben. Abbildung 85 visualisiert auf der linken Seite den Ablauf der REST API Applikation, die Requests der Kameras entgegennimmt. Auf der rechten Seite ist der Ablauf der Message Queue Applikation ersichtlich. Der orange Pfeil visualisiert das Schreiben einer von der REST API empfangenen Nachricht in die Message Queue.

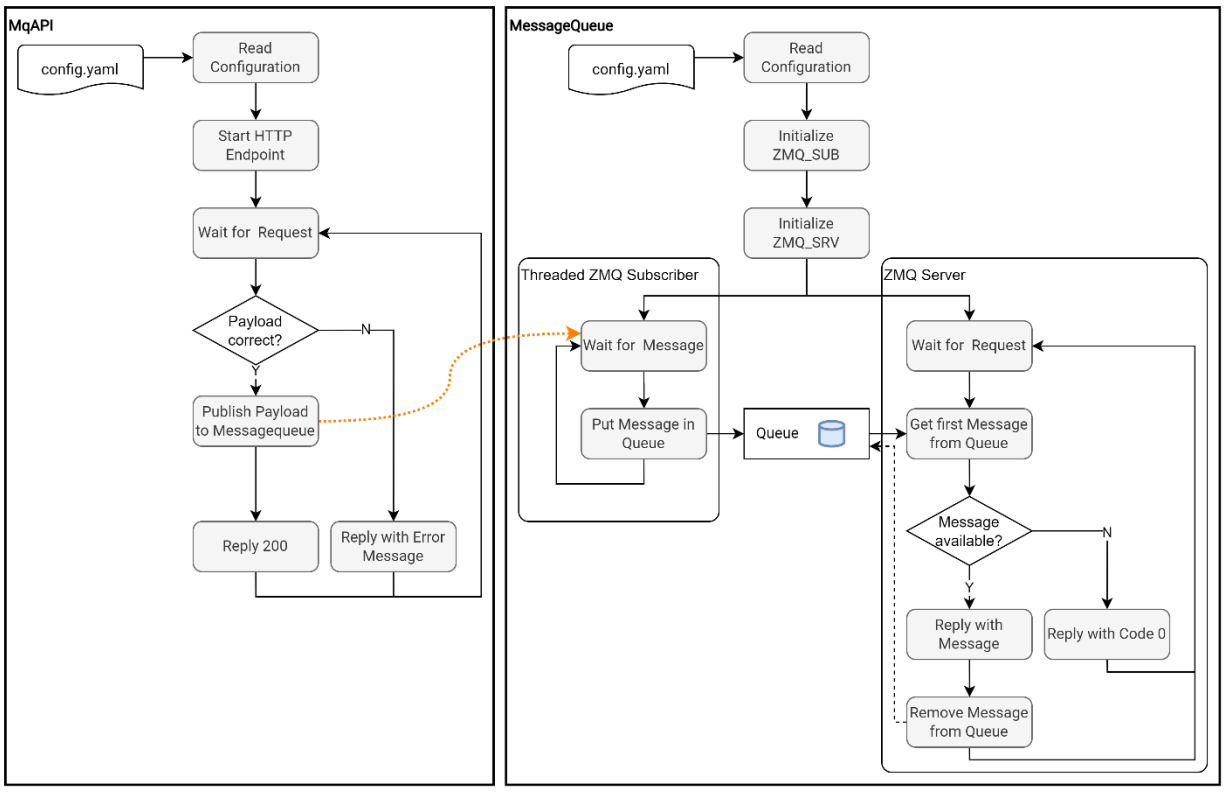


Abbildung 85 Flowchart MessageQueue & REST API

Die Implementierung der Queue als eigenständiger Service wurde gewählt, damit der Zugriff auf das Filesystem von einer einzigen Applikation verwaltet wird. Durch die Speicherung der Zwischenresultate auf der Disk gehen bei einem Neustart des Raspberry Pi oder bei einem Fehler der Bestäuber Auswertung keine Daten verloren. Für die Implementierung der Message Queue wurden Codeausschnitte von der Python Library PersiZMQ [73] übernommen.

5.3.3 Bestäubererkennung auf AP Gateway

Für die Pollinator Inferenz auf dem AP Gateway wird das PyTorch Framework eingesetzt, welches ein 64 Bit OS erfordert. In der Konfiguration werden die Parameter für das Model, der Endpunkt der Message Queue und der Speicherort der Result Files definiert. Optional können die Resultate über HTTP oder MQTT verschickt werden. Der Ablauf der Applikation ist in Abbildung 86 ersichtlich. Mit einem ZMQ Request werden die ältesten Resultate aus der Message Queue abgefragt. Falls Daten vorhanden sind, werden die Informationen über die erkannten Blumen aus dem File extrahiert und die Bestäuber auf den Ausschnitten erkannt. Aus den Informationen der Zwischenresultate und den Ergebnissen der Pollinator Infernez wird dann ein JSON Objekt erstellt, welches auf der Disk gespeichert oder gemäss Output Konfiguration verschickt wird.

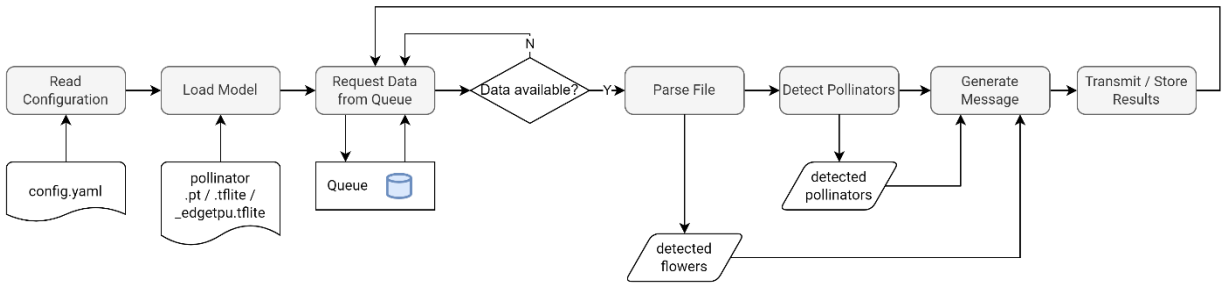


Abbildung 86 Flowchart Pollinator Inferenz auf AP Gateway

5.3.4 Performance

Die Performance der Blumen- und Bestäuber Erkennung wurde in zwei Testläufen nacheinander getestet.

5.3.4.1 Blumenerkennung auf Kamera

Das Model für die Blumenerkennung wurde ein FP-16 quantisiertes TFLite Model mit Bildgrösse 640 Pixel und YOLOv5 Modelgrösse n eingesetzt. Die Bilder wurden vom mock-mjpg-streamer heruntergeladen. Zwischen den Bildern wurde keine Wartezeit definiert, um den maximalen Durchsatz und die Temperaturentwicklung zu analysieren. Im Durchschnitt dauerte der Download 1'457 ms, die Erkennung der Blumen 2'184 ms. Die Verteilung der Zeiten sind in Abbildung 87 visualisiert. Die Outliers nach unten bei der Inferenzzeit sind Bilder mit kleineren Auflösungen.

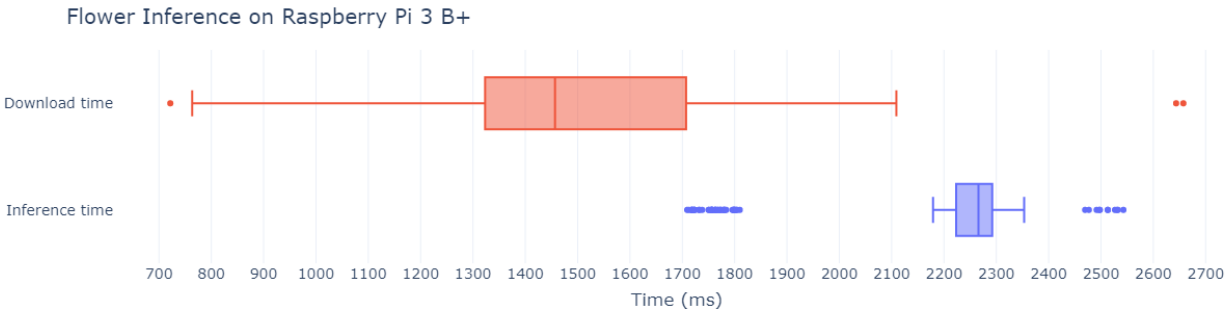


Abbildung 87 Aufteilung Zeiten für Blumenerkennung auf Raspberry Pi 3 B+

Abbildung 88 zeigt die Performance der Blumenerkennung auf der Kamera. Der durchschnittliche Durchsatz beträgt 15.7 Bilder pro Minute. Die Geschwindigkeit ist hauptsächlich abhängig von den Dimensionen der Bilder. Die maximale Memory Auslastung lag bei knapp 33 % der 800 MB. Die Temperatur liegt konstant bei 70 °C.

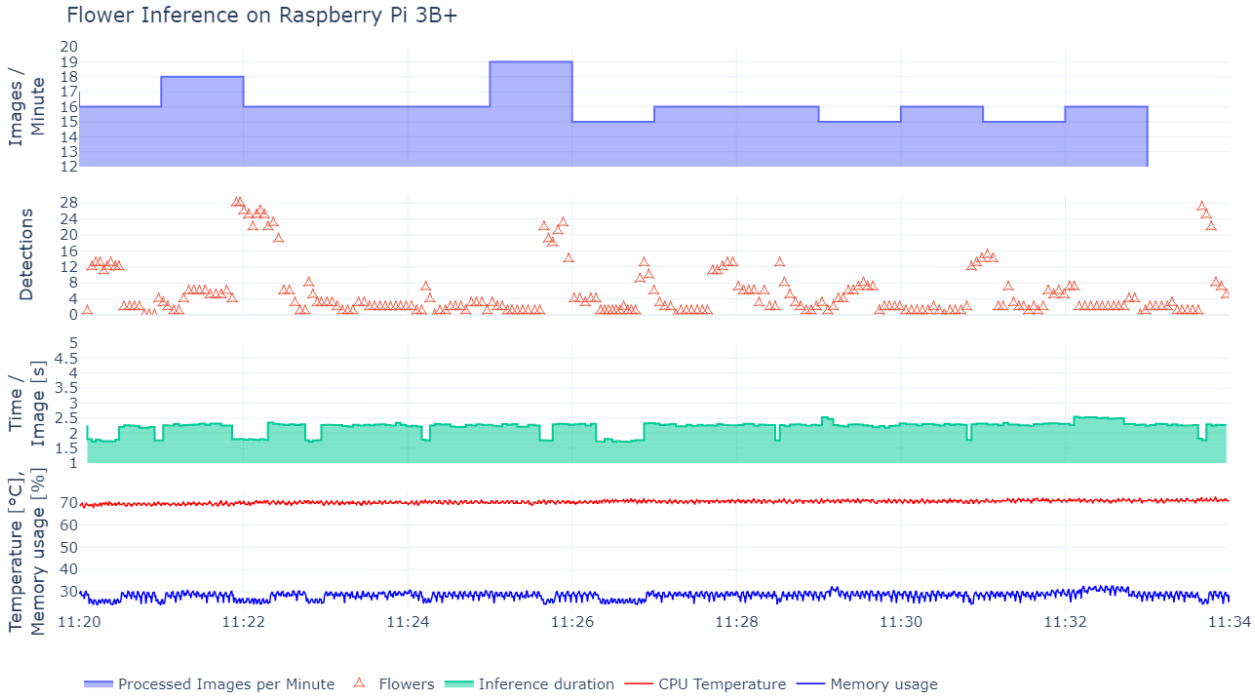


Abbildung 88 Performance Blumenerkennung auf Kamera

5.3.4.2 Bestäubererkennung auf AP Gateway

Die von der Kamera detektierten Blumen wurden mit hinzugefügten Metainformationen im JSON Format in die Message Queue auf dem AP Gateway geschrieben. Für die Messung der Performance wurde ein Raspberry Pi 4 mit 4GB Memory und einem 64 Bit OS eingesetzt.

Das Pollinator Model ist ein nicht quantisiertes PyTorch Model mit Bildgrösse 480 Pixel und YOLOv5 Modelgrösse s. Die Inferenz wurde ohne Inference-Time-Augmentation und mit Multi-Class Option durchgeführt.

Die durchschnittliche Verarbeitungszeit für ein Bildausschnitt mit einer Blume liegt bei 763 ms, pro Originalbild wurden 5'445 ms gebraucht. Die Erkennung der Bestäuber ist auf dem AP Gateway somit fast drei mal so schnell wie auf der Kamera. Abbildung 89 zeigt die Verarbeitungszeit pro Bild in Relation zu der Anzahl Blumen.

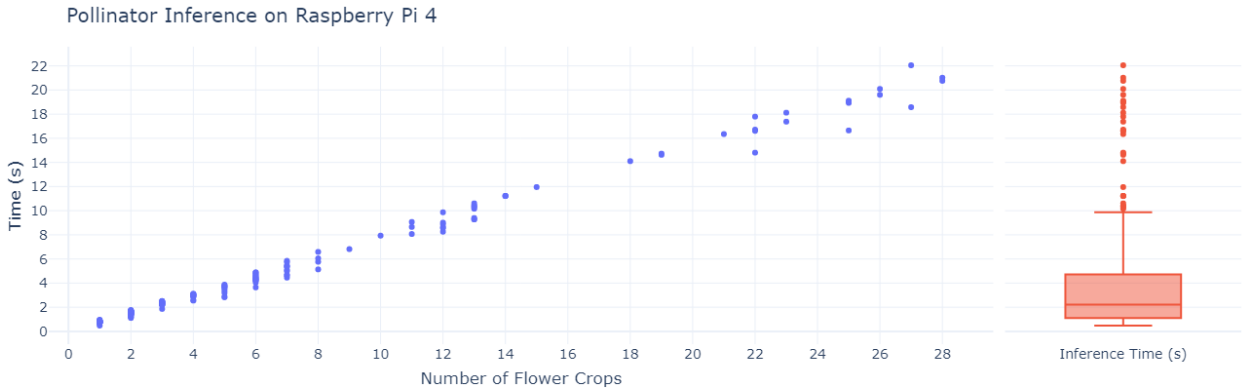


Abbildung 89 Inferenzzeiten in Relation zu Anzahl Ausschnitte für Pollinator Detection auf AP Gateway

Die maximale Memory Auslastung betrug 14.8 % der insgesamt 4 GB. Die CPU Temperatur des nicht gekühlten Raspberry Pi lag zwischen 80 und 85 °C. Die CPU Auslastung lag durchschnittlich bei 91 %.

Die Blumenerkennung auf der Kamera läuft parallel zu der Erkennung von Bestäubern auf dem AP Gateway. Die maximale Erfassungsfrequenz wird durch die Anzahl angeschlossener Kameras und die Anzahl der erkennbaren Blumen auf den Bildern Begrenzt. Durch die Zwischenspeicherung der erkannten Blumen auf dem AP Gateway stehen 24 Stunden pro Tag für die Auswertung zur Verfügung. Bei einem Setup mit vier Kameras, die jeweils von 08:00 bis 14:00 alle 15 Sekunden ein Bild erfassen und auswerten, entstehen pro Tag 5'760 Bilder. Wenn im Durchschnitt 10 Blumen pro Kamera erkannt werden, entstehen 57'600 Ausschnitte, die auszuwerten sind. Beträgt die Inferenzzeit, wie im durchgeführten Test, 763 ms, dauert die Pollinator Inferenz knapp 12 Stunden und 13 Minuten. Die Limite einer Verarbeitung im Dauerbetrieb ohne Rückstau liegt bei 113'237 Ausschnitten, was im oben genannten Setup 19.65 Blumen pro Topf entspricht.

5.3.5 Vor- und Nachteile

Mit der verteilten Auswertung ist ein hoher Aufnahmeintervall möglich, welcher hauptsächlich durch die Inferenzzeit der Blumenerkennung limitiert wird. Die Pollinator Inferenz wird parallel dazu auf dem AP Gateway durchgeführt und kann auch nach dem Aufnahmezeitfenster weiterlaufen. Wenn 25 % des Tages Bilder erfasst werden, darf die Dauer der Bestäubererkennung bei vier angeschlossenen Kameras so lange wie der Capture Intervall sein.

Anstelle der gesamten Bilder werden nur die Ausschnitte der Bestäuber gespeichert, womit der Speicherplatz auch bei einer hohen Aufnahmefrequenz keine Beschränkung darstellt. Falls ein Fehler vorliegt, gehen durch die persistente Speicherung keine Daten verloren. Es existiert keine Abhängigkeit zu einem Backend, womit das System offline betreibbar ist und das Monitoring der

Biodiversität in abgelegenen Regionen ermöglicht. Es werden alle gängigen Model Formate, inklusive TPU, unterstützt, womit das System eine hohe Flexibilität bietet.

Durch den Einsatz von PyTorch wird ein 64 Bit Betriebssystem vorausgesetzt. Die bestehenden AP Gateways müssen für den Einsatz neu aufgesetzt werden. Die CPU Temperatur des AP Gateway ist ohne Kühlung stark angestiegen. Wie sich die Temperatur bei einem Einsatz im Feld mit verbautem Lüfter verhält, muss an einem Test Setup analysiert werden. Die Temperatur der Kamera erreichte während dem Testen keine kritischen Werte.

5.4 Gesamte Auswertung auf AP Gateway

Die Implementierung der gesamten Auswertung auf dem AP Gateway muss asynchron durchgeführt werden, da die Dauer der Inferenz abhängig von der Anzahl Blumen auf den Bildern ist. Die Applikation wurde modular entwickelt, um das bestehende System einfach zu erweitern. Nach dem Herunterladen der Bilder auf den AP Gateway werden die Filenames in eine Message Queue [71] geschrieben, um die Reihenfolge zu gewährleisten. Der Capture Service muss um die Publizierung der Filenames ergänzt werden, wie es in Abbildung 90 ersichtlich ist.

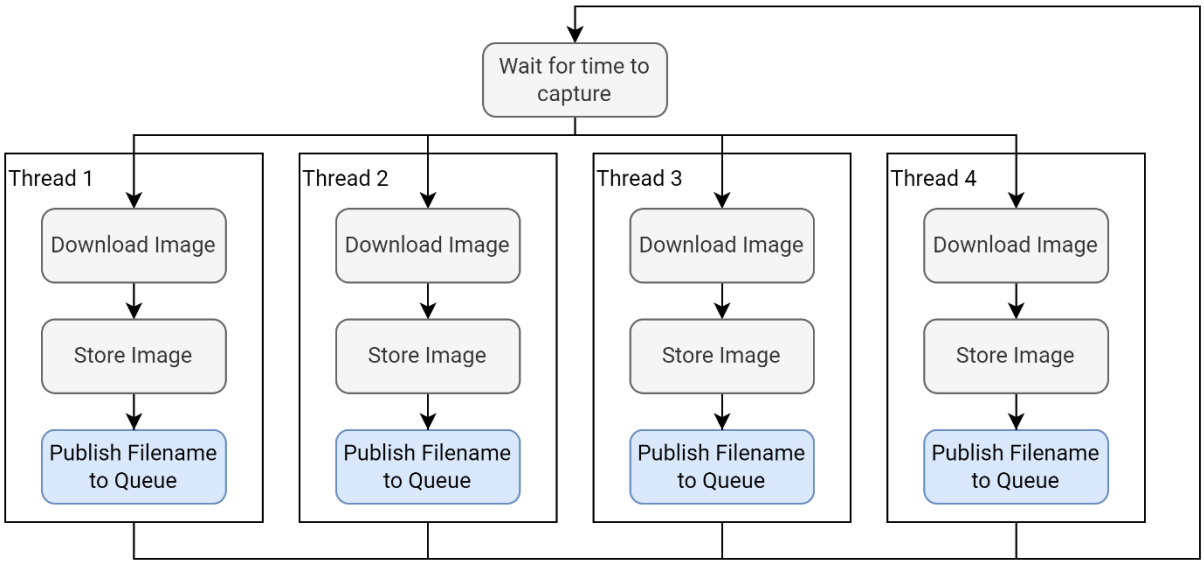


Abbildung 90 Flowchart Capture Service

Die Auswertung basiert auf dem PyTorch Framework, wobei eine Vielzahl Model Formaten unterstützt werden. Die Konfiguration der Applikation beinhaltet neben den Model Parametern den Endpunkt für die Abfrage der Nachrichten aus der Message Queue. Der Output kann ein File sein, welches auf der Disk gespeichert wird, oder eine Übertragung via HTTP oder MQTT.

Der Ablauf der Inferenz ist in Abbildung 91 visualisiert. Die Konfiguration wird aus dem YAML File gelesen und anschliessend werden die Models geladen. Ein ZMQ Client Fordert den ältesten Eintrag aus der Message Queue an und liest den Speicherort des Bildes aus der Nachricht, sofern Bilder zur Analyse bereit sind. Auf dem Bild werden die Blumen detektiert und anschliessend die Bestäuber. Aus den Resultaten wird ein JSON Objekt erstellt und gemäss der Output Konfiguration als File gespeichert oder versendet.

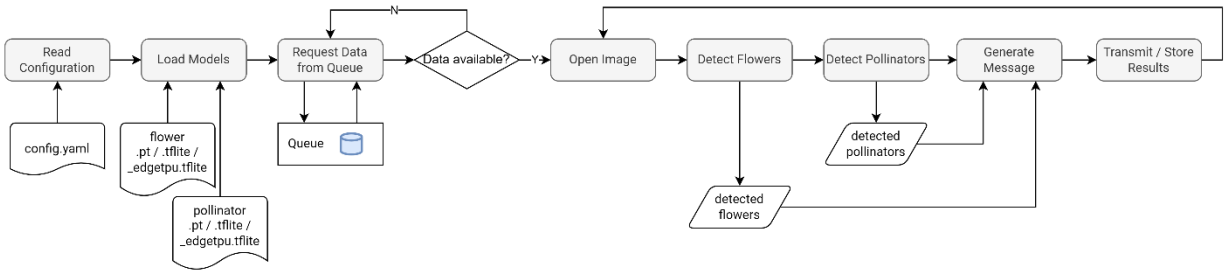


Abbildung 91 Flowchart Gesamte Auswertung auf AP Gateway

Als Alternative zu der Message Queue kann ein Ordner als Input angegeben werden. Der Ordner und die Unterordner werden regelmässig nach neuen Bildern abgesucht, die dann verarbeitet werden. Optional können die ausgewerteten Bilder gelöscht werden, um Speicherplatz freizugeben.

5.4.1 Performance

Die Test Bilder wurden auf dem Raspberry Pi gespeichert und als Directory Input eingelesen. Die Parameter für die Inferenz wurden die selben wie für die Bestäuber Detektion auf dem AP Gateway gewählt.

Der Zeitaufwand für die Auswertung der 236 Bilder beträgt 18 Minuten und 8 Sekunden, im Durchschnittlich 4610 ms pro Bild. Die benötigte Zeit teilt sich auf in 796 ms Flower Inferenz und 3823 ms Pollinator Inferenz. Das Detektieren von Bestäubern dauert 701 ms pro Ausschnitt. Abbildung 92 zeigt Messungen während des Tests. Die Memory Auslastung ist mit unter 20 % sehr gering. Die Temperatur steigt auf knapp 85 °C an und bleibt konstant.

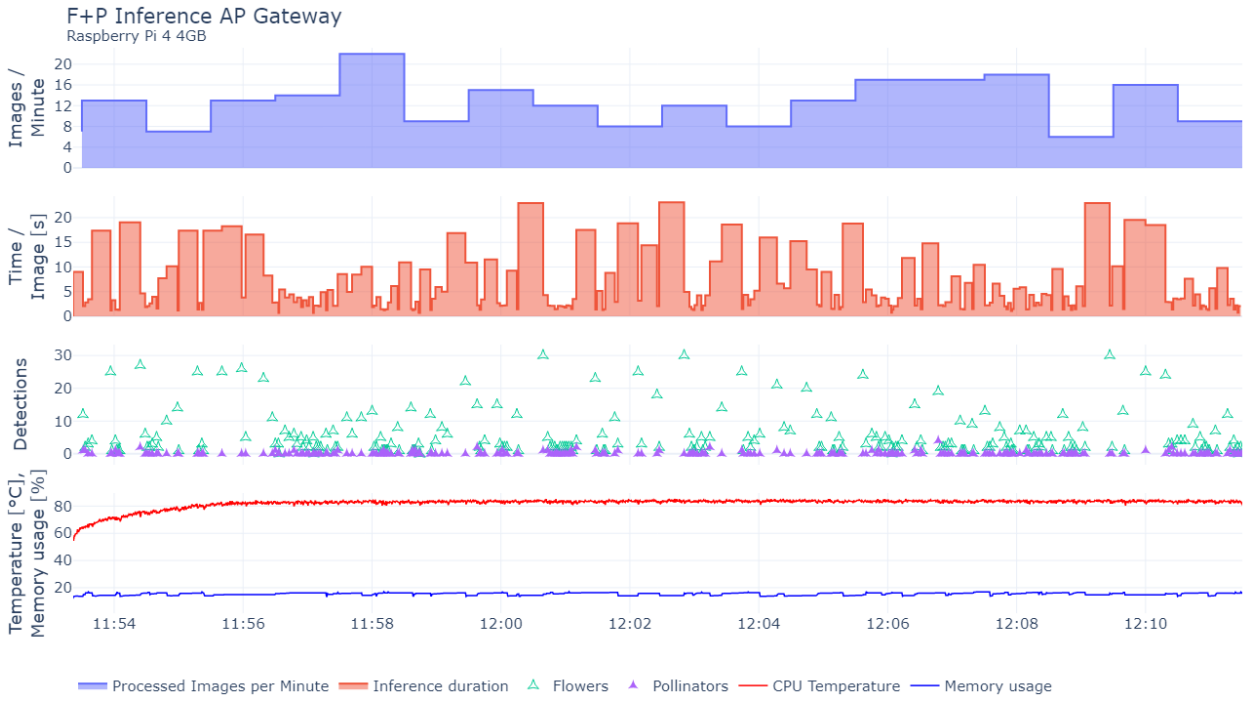


Abbildung 92 Performance Auswertung auf AP Gateway

Ein Setup mit vier angeschlossenen Kameras, die täglich während sechs Stunden alle 15 Sekunden eine Aufnahme starten, generiert 5760 Bilder pro Tag. Mit 10 erkennbaren Blumen pro Topf dauert die gesamte Auswertung knapp 12.5 Stunden. Durch die asynchrone Implementierung der Auswertung besteht die Möglichkeit, sie während kühlen Tageszeiten durchzuführen und bei höheren Temperaturen zu pausieren.

5.4.2 Vor- und Nachteile

Die Auswertung auf dem AP Gateway ist modular aufgebaut und kann bei dem bestehenden System ohne Anpassungen an den Kameras ergänzt werden. Die Geschwindigkeit ist hoch genug, dass alle aufgenommenen Bilder innerhalb eines halben Tages analysiert werden. Ein Offlinebetrieb ist möglich und es existiert keine Abhängigkeit zu einem Backend. Die Originalbilder können nach der Auswertung gelöscht werden, um längere Einsätze zu ermöglichen.

Ein Nachteil sind die hohen Temperaturen, die vom SBC erreicht werden. Bei einer Implementierung der Auswertung auf den bestehenden AP Gateway müssen diese mit einem 64 Bit OS neu aufgesetzt werden.

5.5 Gesamte Auswertung im Backend

Die gesamte Auswertung im Backend zu implementieren ist insbesondere Sinnvoll, wenn die Daten sowieso im Backend gespeichert werden und dafür übermittelt werden müssen. Wie bei der gesamten Auswertung auf dem AP Gateway wird das PyTorch Framework eingesetzt, womit die selbe Applikation unverändert auf einer VM installiert werden kann. Anders als auf den Raspberry Pi besteht im Backend die Möglichkeit, eine GPU für die Inferenz einzusetzen. Je nach Berechnungsleistung der entsprechenden Infrastruktur können grosse Models für die Inferenz eingesetzt werden.

Für die Implementierung in eine bestehende Infrastruktur können Input- und Output Schnittstellen angepasst werden. Eine mögliche Integrationsmöglichkeit ist das Lesen von Bildern von einem S3 Storage. Die Resultate können via HTTP API oder direkt in eine Datenbank eingefügt und die Resultate in einem Objektspeicher abgelegt werden.

5.5.1 Performance

Die Auswertung im Backend wurde auf einem Windows Rechner mit einer Nvidia GeForce GTX 1650 Ti [74] mit 4GB GPU RAM getestet. Die Testbilder wurden aus einem Ordner gelesen und ausgewertet. Es wurden die selben Models wie beim Testen der Auswertung auf dem AP Gateway eingesetzt (Flower n 640, Pollinator s 480). Die Auswertung der 236 Testbilder war nach einer Minute und 26 Sekunden abgeschlossen, woraus sich eine Durchschnittszeit von knapp 365 ms pro Bild errechnet. Die Zeit teilt sich auf in 179 ms Flower Inferenz, 179 ms Pollinator Inferenz, 5 ms zum Speichern der Resultate und 2 ms zum Öffnen der Bilder. Die Erkennung von Bestäubern auf einer ausgeschnittenen Blume dauert durchschnittlich 32.7 ms.

Abbildung 93 zeigt eine Zusammenfassung des Testlauft. Als höchste Auswertungszeit pro Bild wurden 1290 ms aufgezeichnet. Der beanspruchte GPU Speicher lag konstant bei 24.85 %, die Load betrug im Durchschnitt 22 %.

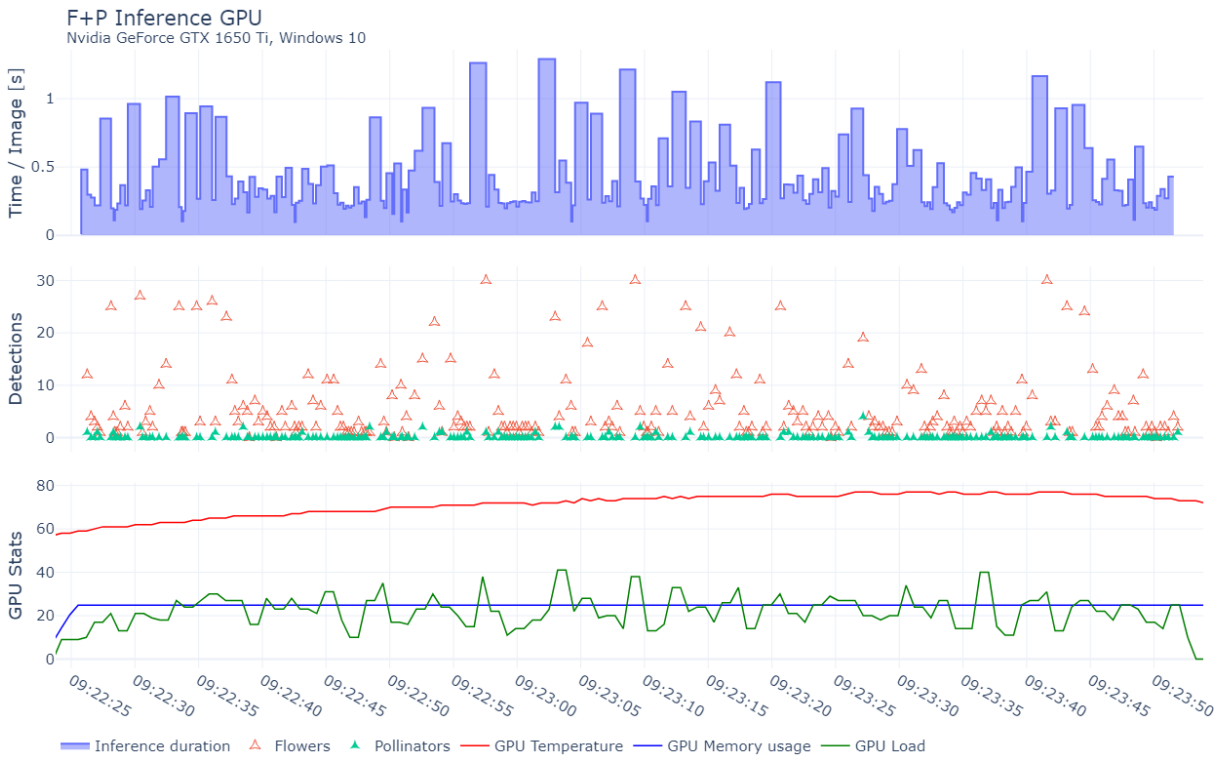


Abbildung 93 Performance Auswertung mit GPU

Für ein weiterer Test wurde ein Flower Model mit Bildgrösse 1280 und YOLOv5 Modelgrösse n6 und ein Pollinator Model mit Bildgrösse 640 und YOLOv5 Modelgrösse m analysiert. Die gesamte

Auswertungszeit für die 236 Bilder betrug zwei Minuten und neun Sekunden, pro Bild im Durchschnitt 546 ms. Davon sind 232 ms Flower Inferenz und 301 ms Pollinator Inferenz. Die Zeit für das Erkennen von Bestäubern beträgt 55.9 ms pro Ausschnitt. Die GPU Load betrug im Schnitt 37 %, die GPU Memory Auslastung lag konstant bei 29.2 %.

Die Zeiten für die Blumen- und Bestäuber Erkennung wurden für jeweils drei unterschiedliche Models analysiert und verglichen. Abbildung 94 visualisiert die Ergebnisse. Die Unterschiede zwischen den Grössen sind bei der Inferenz mit einer GPU wesentlich kleiner als mit einer CPU.

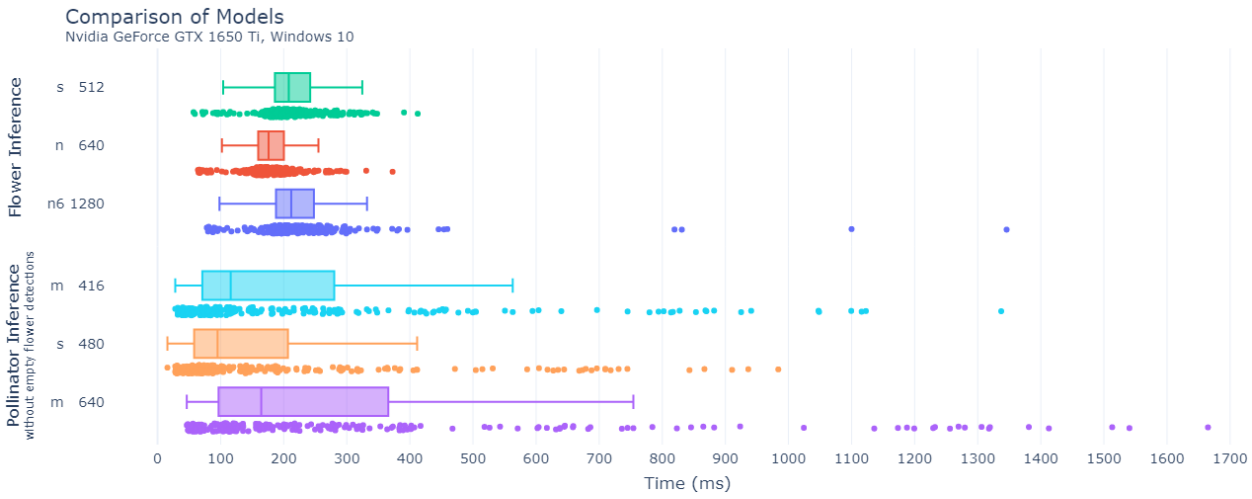


Abbildung 94 Vergleich Inferenzzeiten unterschiedlicher Modelgrössen

Eine Auswertung von allen in der ersten Fallstudie erfassten Bildern würde mit den schnellsten Models auf der Test Infrastruktur gut 152 Stunden dauern.

5.5.2 Vor- und Nachteile

Der grösste Vorteil liegt in der hohen Performance der Inferenz auf einer GPU. Sie bietet die Möglichkeit, komplexere Models einzusetzen. Die Gefahr, dass die Hardware im Feld durch hohe Temperaturen aufgrund der CPU Auslastung beschädigt wird, fällt weg. Die Auswertungspipeline ist nicht direkt vom Kamerasystem abhängig. Die Erfassung von Bildern kann auch mit einem anderen Kamerasystem durchgeführt werden.

Der grösste Nachteil der Auswertung im Backend ist der Aufwand zum Aufsetzen der Infrastruktur. Abhängig von der gewählten Infrastruktur können hohe Kosten anfallen. Ein weiterer Nachteil ist der hohe Datentransfer durch die Übertragung aller Bilder in voller Auflösung. Der Upload Speed der Internetverbindung muss entsprechend hoch gewählt werden.

Die Auswertung im Backend macht Sinn, wenn die Bilder standardmässig an ein Backend gesendet werden und die Infrastruktur vorhanden ist. Für den allgemeinen Einsatz für das Biodiversitätsmonitoring stellen das Deployment des Backends und die damit verbundenen Kosten die grössten Hürden dar.

6 Analyse von Resultaten

Für jedes Bild wird nach der Auswertung ein JSON File generiert. Die Bildausschnitte der Bestäuber sind serialisiert und können ohne zusätzliche Tools nicht geöffnet werden. Für die Visualisierung der Resultate und das Generieren eines PDF Reports wurden Tools entwickelt, die folgend dokumentiert werden.

6.1 Visualisierung von Resultaten

Für die Visualisierung der Resultate wurde ein interaktives Dashboard entwickelt. Als Input wird ein Ordner, in welchem Resultate gespeichert sind, angegeben. Die Applikation basiert auf dem Dash Framework von Plotly [40]. Der Source Code ist im Repository P8-Tools abgelegt [38]. Abbildung 95 zeigt ein Screenshot der Visualisierung. Es werden Daten von Aufnahmen einer Kamera während acht Tagen visualisiert. Auf der linken Seite ist die Anzahl von erkannten Bestäubern pro Kategorie in einem Bar Chart ersichtlich. Dem Chart rechts davon werden die Zeiträume, an welchen die Bestäuber aufgenommen wurden, gezeigt. Die Tageszeiten, an welchen die Bestäuber der jeweiligen Klassen am aktivsten sind, wurden mit einer Kernel Density Estimation berechnet und dargestellt. Die Zusammensetzung der Blumenarten wird als Kuchendiagramm zusammengefasst.

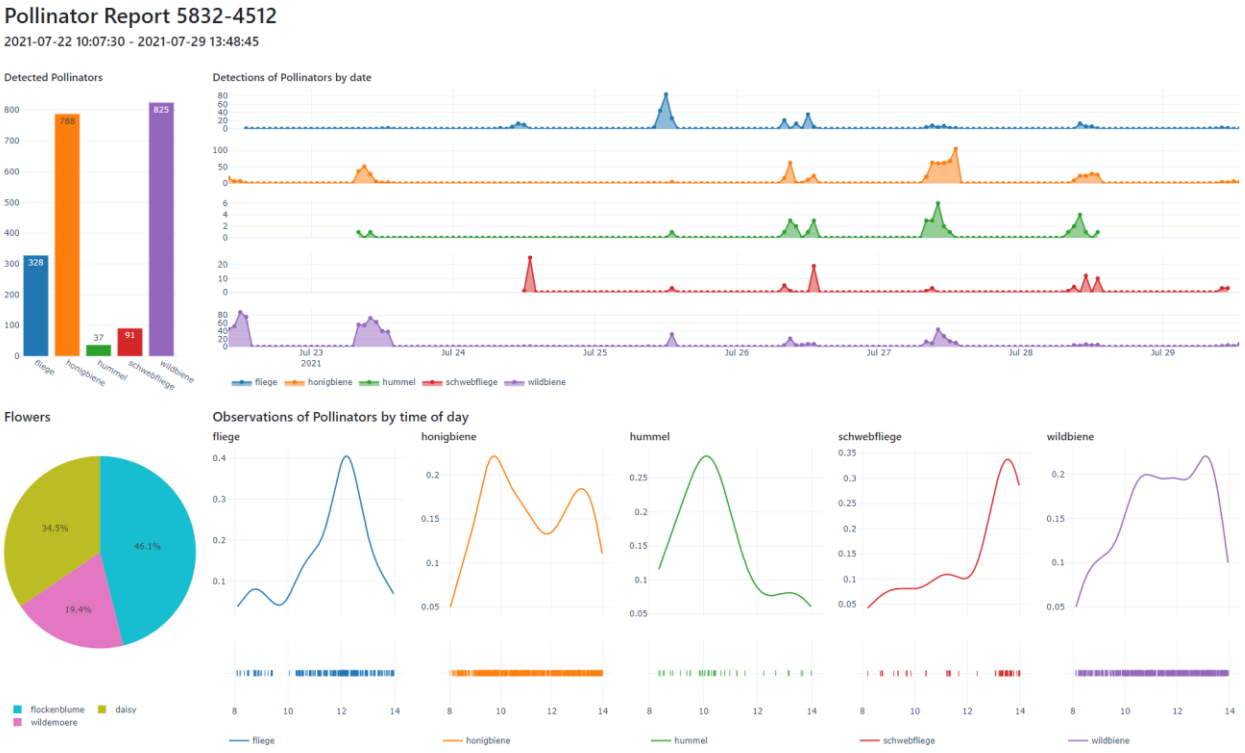


Abbildung 95 Screenshot interaktives Dashboard Visualisierung der Resultate

Unterhalb der Charts befindet sich pro Bestäuberklasse ein Abschnitt, worin ein Mosaik aus zufällig selektierten Bildausschnitten gezeigt wird. Das Mosaik dient zur Sichtkontrolle, um False Positives möglichst schnell zu erkennen und die Resultate genauer zu analysieren. Die Wahrscheinlichkeit, mit welcher die Bestäuber erkannt wurden, ist pro Ausschnitt annotiert. Abbildung 96 zeigt ein Mosaik aus Resultaten der Klasse Schwebfliege. Die Grösse des Mosaiks kann konfiguriert werden.

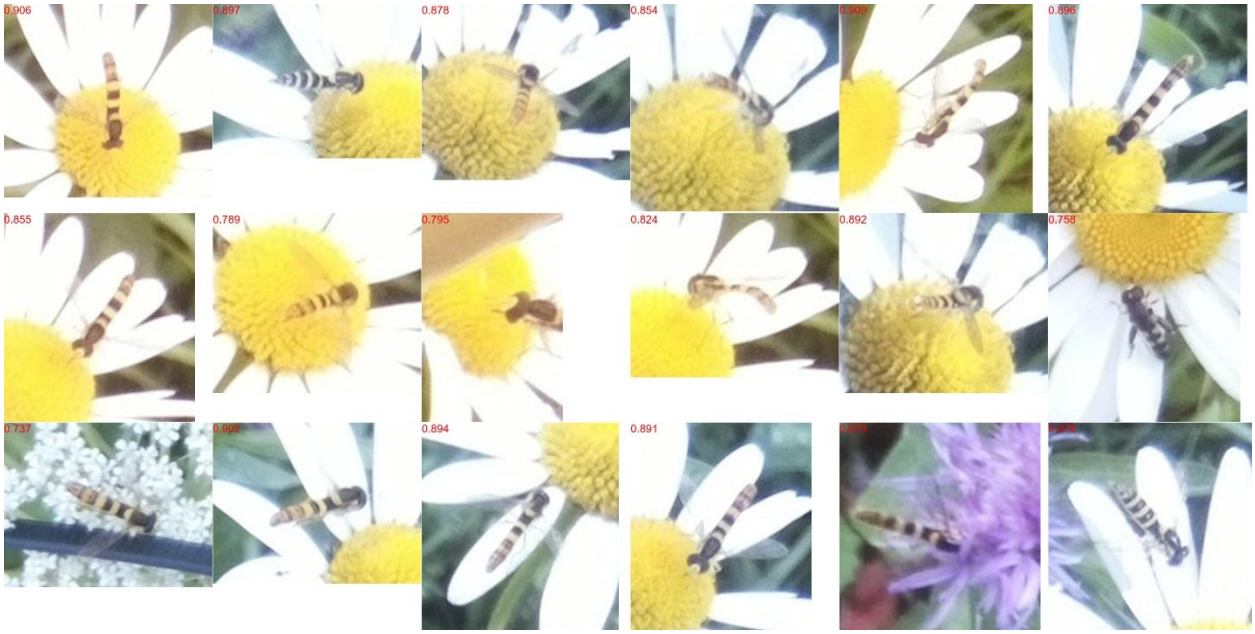


Abbildung 96 Mosaik aus Resultaten der Klasse Schwebfliege

Mit dem report_generator Tool [38] kann aus den Resultaten ein Report im PDF Format generiert werden. Es beinhaltet die selben Elemente wie das interaktive Dashboard in Abbildung 95. Alle Charts werden als Bilder gespeichert. Optional kann ein zusätzlicher Export der Charts im SVG Format aktiviert werden.

6.2 Validierung der Resultate

Eine Analyse der Resultate von knapp 50'000 ausgewerteten Bildern hat gezeigt, dass False Positives oft in mehreren aufeinanderfolgenden Bildern auftreten und somit in den Grafiken der Visualisierung als Ausreisser erkennbar sind. Für eine automatisierte Erkennung von potentiellen False Positives kann eine Anomalie Detektion mit statistischen Werkzeugen oder mit Hilfe eines ML Models implementiert werden.

Für die Verbesserung der Models ist es sinnvoll, Resultate mit falschen Klassen oder ohne Bestäuber zu sammeln, korrekt zu annotieren und dem Datenset anzufügen. Sobald das Datenset um einen relevanten Anteil neuer Bilder erweitert wurde, kann eine neue Version der Models damit trainiert werden. Somit wird das Model robuster und passt sich Veränderungen an.

7 Fazit

Im Rahmen dieser Projektarbeit wurde erforscht, wie eine Auswertung von Bilddaten automatisiert werden kann. Konkret sollen Bestäuber auf Bildern von Blumen detektiert werden, um eine Aussage über die Biodiversität zu machen. Für die Entwicklung der Auswertung standen rund 7 TB Bilder, die in einer Fallstudie des Forschungsprojekts Mitwelten erhoben wurden, zur Verfügung. Um möglichst viele Informationen über die vorhandenen Aufnahmen zu erhalten und sie zu kategorisieren wurde das Image Exploration Tool entwickelt. Eine Analyse hat gezeigt, dass rund die Hälfte der Aufnahmen keine Blumen zeigen und somit für die Entwicklung der Auswertung irrelevant sind.

Recherchen haben ergeben, dass der Einsatz von Deep Learning Models für die Objekterkennung dem State of the Art entspricht. Die entwickelte Auswertungspipeline besteht aus zwei Models. In einem ersten Schritt werden Blumen erkannt und aus dem Originalbild ausgeschnitten. In einem zweiten Schritt werden die Ausschnitte nach Bestäubern abgesucht.

Für die Blumenerkennung wurde ein Datenset erstellt, welches 501 Bilder umfasst und drei Klassen von Blumen unterscheidet. Das Datenset für die Erkennung von Bestäubern besteht aus 3'679 Bildern und unterscheidet fünf Morphospezies. Für die Wahl des optimalen Models wurden verschiedene Typen trainiert und getestet. Mit YOLOv5 Models wurden die besten Ergebnisse in den Bereichen Genauigkeit und Performance erzielt. Auch in Hinsicht auf die Weiterentwicklung des Models ist YOLOv5 mit einer hohen Benutzerfreundlichkeit die beste Option.

Die Blumenerkennung hat bei Testläufen praktisch alle Blumen korrekt erkannt. Die Bestäuber wurden bis auf einige Ausnahmen als solche erkannt. Gewisse Bestäuber wurden, bedingt durch die Aufteilung der Klassen, fälschlicherweise als Wildbiene erkannt. Falsch erkannte Objekte können korrekt annotiert werden und dem Datenset hinzugefügt werden, um eine neue Version des Models zu trainieren.

Insgesamt vier Konzepte für eine Automatisierung der Auswertung wurden entwickelt und getestet. Am einfachsten nachzubauen und somit potentiell tauglich für den Einsatz in der Umweltforschung hat die Auswertung direkt auf der Kamera. Für den Einsatz in Kamera Clustern ist die Verteilte Auswertung am effizientesten. Die höchste Flexibilität und Performance wird mit der Auswertung im Backend erreicht. Mit allen Konzepten können wissenschaftlich relevante Daten aus den Aufnahmen extrahiert werden.

Insgesamt wurden die in der Aufgabenstellung festgelegten Ziele erreicht und das Projekt wurde erfolgreich abgeschlossen.

8 Literaturverzeichnis

[1] D. M. Weber and B. IT-Services, “Künstliche Intelligenz,” *Künstl. Intell.*, p. 228.

[2] Google Developers, “Reducing Loss: An Iterative Approach | Machine Learning Crash Course,” *Google Developers*. https://developers.google.com/machine-learning/crash-course/reducing-loss/an-iterative-approach (accessed Jul. 05, 2022).

[3] A. B. Jones, “Sentiment analysis on reviews: Train Test Split, Bootstrapping, Cross Validation & Word Clouds,” *Medium*, Mar. 09, 2018. https://medium.com/@annabiancajones/sentiment-analysis-on-reviews-train-test-split-bootstrapping-cross-validation-word-clouds-4ae65e745f59 (accessed Jul. 05, 2022).

[4] T.-Y. Lin *et al.*, “Microsoft COCO: Common Objects in Context.” arXiv, Feb. 20, 2015. Accessed: Jul. 05, 2022. [Online]. Available: http://arxiv.org/abs/1405.0312

[5] “YOLOv5 (6.0/6.1) brief summary · Issue #6998 · ultralytics/yolov5,” *GitHub*. https://github.com/ultralytics/yolov5/issues/6998 (accessed Jul. 06, 2022).

[6] Z. J. Wang *et al.*, “CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization,” *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 1396–1406, Feb. 2021, doi: 10.1109/TVCG.2020.3030418.

[7] R. Szeliski, “Deep Learning,” in *Computer Vision: Algorithms and Applications*, R. Szeliski, Ed. Cham: Springer International Publishing, 2022, pp. 187–271. doi: 10.1007/978-3-030-34372-9_5.

[8] Unsplash, “Photo by Maicol Santos on Unsplash.” https://unsplash.com/photos/JI7aL1aM-LHc (accessed Jun. 08, 2022).

[9] Unsplash, “Photo by Anusha Barwa on Unsplash.” https://unsplash.com/photos/ppKcYi1CXcI (accessed Jun. 08, 2022).

[10] N. I. Glumov, E. I. Kolomiyetz, and V. V. Sergeyev, “Detection of objects on the image using a sliding window mode,” *Opt. Laser Technol.*, vol. 27, no. 4, pp. 241–249, Aug. 1995, doi: 10.1016/0030-3992(95)93752-D.

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation.” arXiv, Oct. 22, 2014. Accessed: Jun. 17, 2022. [Online]. Available: http://arxiv.org/abs/1311.2524

[12] R. Girshick, “Fast R-CNN.” arXiv, Sep. 27, 2015. Accessed: Jun. 17, 2022. [Online]. Available: http://arxiv.org/abs/1504.08083

[13] R. Gandhi, “R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms,” *Medium*, Jul. 09, 2018. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e (accessed Jun. 17, 2022).

[14] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” arXiv, Jan. 06, 2016. Accessed: Jun. 17, 2022. [Online]. Available: http://arxiv.org/abs/1506.01497

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection.” arXiv, May 09, 2016. Accessed: Jun. 17, 2022. [Online]. Available: http://arxiv.org/abs/1506.02640

[16] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger.” arXiv, Dec. 25, 2016. Accessed: Jun. 17, 2022. [Online]. Available: http://arxiv.org/abs/1612.08242

[17] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement.” arXiv, Apr. 08, 2018. Accessed: Jun. 17, 2022. [Online]. Available: http://arxiv.org/abs/1804.02767

[18] J. Redmon, “Darknet.” Jun. 17, 2022. Accessed: Jun. 17, 2022. [Online]. Available: https://github.com/pjreddie/darknet/blob/b1ab3da442574364f82c09313a58f7fc93cea2bd/cfg/darknet53.cfg

[19] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature Pyramid Networks for Object Detection.” arXiv, Apr. 19, 2017. Accessed: Jun. 17, 2022. [Online]. Available: http://arxiv.org/abs/1612.03144

[20] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection.” arXiv, Apr. 22, 2020. Accessed: Jun. 17, 2022. [Online]. Available: http://arxiv.org/abs/2004.10934

[21] T. DeVries and G. W. Taylor, “Improved Regularization of Convolutional Neural Networks with Cutout.” arXiv, Nov. 29, 2017. Accessed: Jun. 18, 2022. [Online]. Available: http://arxiv.org/abs/1708.04552
[22] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, “CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features.” arXiv, Aug. 07, 2019. Accessed: Jun. 18, 2022. [Online]. Available: http://arxiv.org/abs/1905.04899
[23] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “CBAM: Convolutional Block Attention Module.” arXiv, Jul. 18, 2018. Accessed: Jun. 18, 2022. [Online]. Available: http://arxiv.org/abs/1807.06521
[24] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, “Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression.” arXiv, Nov. 19, 2019. Accessed: Jun. 18, 2022. [Online]. Available: http://arxiv.org/abs/1911.08287
[25] “PyTorch.” https://www.pytorch.org (accessed Jun. 20, 2022).
[26] G. Maindola, “A Brief History of YOLO Object Detection Models From YOLOv1 to YOLOv5,” *MLK - Machine Learning Knowledge*, Aug. 26, 2021. https://machinelearning-knowledge.ai/a-brief-history-of-yolo-object-detection-models/ (accessed Jun. 18, 2022).
[27] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” vol. 9905, 2016, pp. 21–37. doi: 10.1007/978-3-319-46448-0_2.
[28] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge.” arXiv, Jan. 29, 2015. Accessed: Jun. 20, 2022. [Online]. Available: http://arxiv.org/abs/1409.0575
[29] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition.” arXiv, Apr. 10, 2015. Accessed: Jun. 20, 2022. [Online]. Available: http://arxiv.org/abs/1409.1556
[30] G. Jocher *et al.*, “ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference.” Zenodo, Feb. 22, 2022. doi: 10.5281/zenodo.6222936.
[31] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, “Soft-NMS -- Improving Object Detection With One Line of Code.” arXiv, Aug. 08, 2017. Accessed: Jun. 15, 2022. [Online]. Available: http://arxiv.org/abs/1704.04503
[32] “🌟💶 YOLOv5 🚀 EXPORT Competition!! $10000 in Prizes · Discussion #3213 · ultralytics/yolov5,” *GitHub*. https://github.com/ultralytics/yolov5/discussions/3213 (accessed Jul. 12, 2022).
[33] K. Jakhar, “Yolov5 export to Raspberry Pi.” Jul. 08, 2022. Accessed: Jul. 12, 2022. [Online]. Available: https://github.com/karanjakhar/yolov5-export-to-raspberry-pi
[34] N. P. Jouppi *et al.*, “In-Datacenter Performance Analysis of a Tensor Processing Unit.” arXiv, Apr. 16, 2017. doi: 10.48550/arXiv.1704.04760.
[35] “USB Accelerator,” *Coral*. https://coral.ai/products/accelerator/ (accessed Jul. 26, 2022).
[36] CoralAi, “Edge TPU Compiler,” *Coral*. https://coral.ai/docs/edgetpu/compiler/ (accessed Jul. 14, 2022).
[37] A. M. Kist, “Deep Learning on Edge TPUs.” arXiv, Aug. 31, 2021. Accessed: Jul. 14, 2022. [Online]. Available: http://arxiv.org/abs/2108.13732
[38] WullT, “P8-Tools.” Jul. 27, 2022. Accessed: Jul. 27, 2022. [Online]. Available: https://github.com/WullT/P8-Tools/blob/a3c950a5368bf6483bbd44c58b29c84b033b15c5/mock-mjpg-streamer/main.py
[39] R. D. Hipp, “SQLite,” 2020. https://www.sqlite.org/index.html (accessed Jun. 07, 2022).
[40] Plotly, “Dash Documentation & User Guide | Plotly.” https://dash.plotly.com/ (accessed Jun. 07, 2022).
[41] iNaturalist, “iNaturalist,” *iNaturalist*. https://www.inaturalist.org/ (accessed Jul. 19, 2022).
[42] wullti, “mitwelten-flower-dataset,” Jul. 29, 2022. https://www.kaggle.com/datasets/wullti/mitweltenflowerdataset (accessed Jul. 29, 2022).
[43] “Fluginsekten (Unterklasse Pterygota),” *iNaturalist*. https://www.inaturalist.org/taxa/184884-Pterygota (accessed Jul. 20, 2022).
[44] wullti, “mitwelten-pollinator-dataset,” 27 2022. https://www.kaggle.com/datasets/wullti/mitweltenpollinatordataset (accessed Jul. 29, 2022).
[45] M. Tan, R. Pang, and Q. V. Le, “EfficientDet: Scalable and Efficient Object Detection.” arXiv, Jul. 27, 2020. Accessed: Jul. 21, 2022. [Online]. Available: http://arxiv.org/abs/1911.09070

[46] Tensorflow, “tflite_model_maker.object_detector.EfficientDetSpec | TensorFlow Lite,” Feb. 07, 2022. https://www.tensorflow.org/lite/api_docs/python/tflite_model_maker/object_detector/EfficientDetSpec (accessed Jul. 21, 2022).
[47] N. Tsukamoto, “Benchmarks.” Jul. 12, 2022. Accessed: Jul. 21, 2022. [Online]. Available: https://github.com/NobuoTsukamoto/benchmarks/blob/106f8abd7054bf8dd8ee858b69b029d9bd447f0a/tensorflow/lite/efficentdet/efficientdet.md
[48] Tensorflow, “Module: tflite_model_maker | TensorFlow Lite,” *TensorFlow*, Jul. 2022. https://www.tensorflow.org/lite/api_docs/python/tflite_model_maker (accessed Jul. 21, 2022).
[49] Tensorflow, “Object Detection with TensorFlow Lite Model Maker,” *TensorFlow*, May 26, 2022. https://www.tensorflow.org/lite/models/modify/model_maker/object_detection (accessed Jul. 21, 2022).
[50] Towards AI Team, “Understanding Pascal VOC and COCO Annotations for Object Detection – Towards AI.” https://towardsai.net/p/machine-learning/understanding-pascal-voc-and-coco-annotations-for-object-detection, https://towardsai.net/p/machine-learning/understanding-pascal-voc-and-coco-annotations-for-object-detection (accessed Jul. 21, 2022).
[51] L. Vladimirov, “TensorFlowObjectDetectionTutorial.” Jul. 15, 2022. Accessed: Jul. 21, 2022. [Online]. Available: https://github.com/sglvladi/TensorFlowObjectDetectionTutorial/blob/97dc1c92d6d67bcda1593bafd4f6f11a69f7edcb/docs/source/index.rst
[52] Tensorflow, “Welcome to the Model Garden for TensorFlow.” tensorflow, Jul. 21, 2022. Accessed: Jul. 21, 2022. [Online]. Available: https://github.com/tensorflow/models/blob/a5a6522807b9aad516d33494fa83d7a0a7e8d194/research/object_detection/g3doc/tf2_detection_zoo.md
[53] Tensorflow, “TensorBoard,” *TensorFlow*. https://www.tensorflow.org/tensorboard (accessed Jul. 21, 2022).
[54] “Weights & Biases,” *W&B*. https://wandb.ai/site (accessed Jul. 21, 2022).
[55] “Google Colaboratory.” https://colab.research.google.com/notebooks/intro.ipynb?hl=de (accessed Jul. 21, 2022).
[56] Kaggle, “Notebooks Documentation.” https://www.kaggle.com/docs/notebooks (accessed Jul. 21, 2022).
[57] Tensorflow, “Post-training quantization | TensorFlow Lite,” *TensorFlow*, May 26, 2022. https://www.tensorflow.org/lite/performance/post_training_quantization (accessed Jul. 11, 2022).
[58] CoralAi, “TensorFlow models on the Edge TPU,” *Coral*. https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview (accessed Jul. 11, 2022).
[59] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning Deep Features for Discriminative Localization.” arXiv, Dec. 13, 2015. Accessed: Jul. 22, 2022. [Online]. Available: http://arxiv.org/abs/1512.04150
[60] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization,” *Int. J. Comput. Vis.*, vol. 128, no. 2, pp. 336–359, Feb. 2020, doi: 10.1007/s11263-019-01228-7.
[61] P. M. Kazaj, “YOLO-V5 GRADCAM.” Jul. 21, 2022. Accessed: Jul. 22, 2022. [Online]. Available: https://github.com/pooya-mohammadi/yolov5-gradcam
[62] “Test-Time Augmentation (TTA) Tutorial · Issue #303 · ultralytics/yolov5,” *GitHub*. https://github.com/ultralytics/yolov5/issues/303 (accessed Jul. 25, 2022).
[63] Raspberry Pi Ltd, “Operating system images,” *Raspberry Pi*. https://www.raspberrypi.com/software/operating-systems/ (accessed Jul. 12, 2022).
[64] jacksonliam, “mjpg-streamer.” Nov. 28, 2021. Accessed: Nov. 29, 2021. [Online]. Available: https://github.com/jacksonliam/mjpg-streamer
[65] Tristan Rice, “Real Time Inference on Raspberry Pi 4 (30 fps!) — PyTorch Tutorials 1.12.0+cu102 documentation,” Feb. 09, 2022. https://pytorch.org/tutorials/intermediate/realtime_rpi.html (accessed Jul. 12, 2022).
[66] “Pytorch on a raberry Pi 4 (32-bit OS),” *PyTorch Forums*, Dec. 07, 2021. https://discuss.pytorch.org/t/pytorch-on-a-raberry-pi-4-32-bit-os/138771 (accessed Jul. 12, 2022).

[67]“Building PyTorch for the Raspberry Pi (32 bits) - deployment,” *PyTorch Forums*, Feb. 10, 2022. https://discuss.pytorch.org/t/building-pytorch-for-the-raspberry-pi-32-bits/143784 (accessed Jul. 12, 2022).
[68]“cannot torch.jit.trace on ARM 32 bit (Raspbian) · Issue #27040 · pytorch/pytorch,” *GitHub*. https://github.com/pytorch/pytorch/issues/27040 (accessed Jul. 12, 2022).
[69]“Quickstart for Linux-based devices with Python | TensorFlow Lite,” *TensorFlow*. https://www.tensorflow.org/lite/guide/python (accessed Jul. 12, 2022).
[70]WullT, “RPiCamPollinatorDetction.” Jul. 06, 2022. Accessed: Jul. 12, 2022. [Online]. Available: https://github.com/WullT/RPiCamPollinatorDetction
[71]WullT, “ZMQMessageQueue.” Jul. 02, 2022. Accessed: Jul. 12, 2022. [Online]. Available: https://github.com/WullT/ZMQMessageQueue
[72]ZeroMQ, “ZeroMQ,” 2022. https://zeromq.org/ (accessed Jul. 12, 2022).
[73]“Parquery/persizmq.” Parquery, Jan. 15, 2021. Accessed: Jul. 12, 2022. [Online]. Available: https://github.com/Parquery/persizmq
[74]Nvidia, “GEFORCE GTX 16-SERIE,” *NVIDIA*. https://www.nvidia.com/de-de/geforce/graphics-cards/16-series/ (accessed Jul. 28, 2022).

9 Abbildungsverzeichnis

10 Tabellenverzeichnis

11 Ehrlichkeitserklärung

Hiermit bestätige ich, dass die eingereichte Projektarbeit mit dem Titel «Automated Analysis for Urban Biodiversity Monitoring» das Resultat meiner persönlichen Erarbeitung der Themen ist. Ich habe keine anderen als die angegebenen Quellen benutzt.

Ort, Datum Windisch, 31.07.2022 Unterschrift _____

12 Anhang

Die im Rahmen dieser Projektarbeit entwickelte Software ist auf GitHub abgelegt.

Tools:
https://github.com/WullT/P8-Tools

Automatisierte Auswertung auf der Kamera (gesamte Auswertung im *main* Branch, nur Blumenerkennung im Branch *flower-only*):
https://github.com/WullT/RPiCamPollinatorDetction

Automatisierte Auswertung auf AP Gateway oder im Backend:
https://github.com/WullT/Pollinatordetection

Pollinator Detection auf AP Gateway:
https://github.com/WullT/RPiPollinatorInference

ZMQ Message Queue + Message Queue API :
https://github.com/WullT/ZMQMessageQueue