Entwurf und Prototyping eines intelligenten Katzenfutterautomaten

Automatisiertes System zur individuellen Fütterung mit Bilderkennung

Maturaarbeit im Rahmen der Kantonsschule Uetikon am See

Klasse: 6a

Autor: Lukas Gerster

Betreuungsperson: Nicolas Diener

Abgabedatum: 20. Oktober 2025

Abstract

In dieser Arbeit wird ein intelligenter und automatischer Futterautomat für Katzen entwickelt. Der Automat erkennt mithilfe KI-Bildererkennung gezielt bestimmte Katzen und stellt für registrierte Katzen eine vorbestimmte Futterration bereit. Die Bilderkennung erfolgt mithilfe trainierter Keras-Modelle, die einzelne Individuen voneinander unterscheiden können, sowie eines vortrainierten YOLO-Modells. Ein integrierter Boardcomputer übernimmt die Steuerung des Systems. Er ist mit einer Gewichtszelle und einer Kamera verbunden und dosiert über die Motoren die Futterausgabe. Das trainierte KI-Modell kann die Katzen zuverlässig unterscheiden und die automatische Futtervergabe fehlerfrei steuern. Diese Arbeit zeigt, dass KI-gesteuerte Systeme vielfältig eingesetzt werden können, praktischen Nutzen bieten und zudem eine genaue Überwachung des Fressverhaltens ermöglichen, wodurch Essstörungen bei Katzen effektiv vorgebeugt werden kann.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Maturitätsarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank meiner Betreuungsperson Nicolas Diener, die meine Maturitätsarbeit betreut und begutachtet hat. Für die hilfreichen Anregungen und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.

Nicht zuletzt danke ich meinen Katzen Bruno und Fläkli, die mir geduldig für Fotos Modell standen und den Futterautomaten testeten, sowie meinem Vater, der mir bei Problemen stets geholfen hat.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In vielen Haushalten mit mehreren Katzen besteht die Herausforderung, dass jede Katze ihre eigene Futterration erhält, ohne dass andere Katzen das Essen wegfressen. Dieses Problem kenne ich auch aus eigener Erfahrung: Mein Kater Bruno frisst meiner anderen Katze Fläkli oft das Futter weg, wodurch sie nicht immer ihre ganze Tagesration bekommt. Solche Situationen können zu mehreren gesundheitlichen Problemen führen, wie Übergewicht, Untergewicht oder anderweitige abnormale Fressverhalten.

Herkömmliche Futterautomaten lösen das Problem nur teilweise, da sie oft nur eine gewisse Menge an Futter an bestimmten Uhrzeiten ausgeben und nicht kontrollieren können, ob andere Katzen auch darauf zugreifen. Weiterentwickelte Automaten können entweder mit Hilfe von RFID-Chips oder KI-Bildererkennung Katzen erkennen, jedoch kontrollieren sie nicht, was mit den übrig gebliebenen Futterresten passiert, also auch nicht, ob diese von anderen Katzen weggefressen werden.

Mein Projekt setzt genau hier an: Der Automat soll die Futterausgabe kontrollieren und sicherstellen, dass nur die registrierten Katzen fressen können, wodurch die individuellen Tagesrationen zuverlässig eingehalten werden.

Ferner ist dieses Projekt für mich eine perfekte Verbindung meiner persönlichen Interessen und zukünftigen Ziele. Schon seit längerem programmiere ich gerne und baue auch physische Dinge selbst. Angefangen habe ich mit Legotechnik und wollte nun den nächsten Schritt gehen und mit richtigen elektronischen Bauteilen arbeiten. Das Arbeiten an diesem Automaten ermöglicht mir, beides zu kombinieren: Softwareentwicklung mit KI-Elementen und mechanische Umsetzung. Ausserdem plane ich, später Maschinenbau zu studieren. Mit diesem Projekt konnte ich praktische Erfahrungen sammeln und ausprobieren, ob mir die Kombination aus Technik, Konstruktion und Problemlösung wirklich liegt, und erhalte einen realistischen Eindruck davon, wie es ist, ein eigenständiges technisches System zu planen, zu bauen und zu testen.

1.2 Projektumfang und Ziele

Das Projekt verfolgt das übergeordnete Ziel, einen automatischen Katzenfutterautomaten zu entwickeln, der Katzen einzeln erkennen kann und ihre Futterrationen zuverlässig verwaltet. Die einzelnen Ziele lassen sich wie folgt zusammenfassen:

1. **Hardware:**
    - Automatische Futterausgabe in die Schale.
    - Öffnen und Schliessen der Schale, sodass unbefugte Katzen nicht essen können. Dies sorgt dafür, dass die Rationen korrekt verteilt werden.
    - Waage, die das Gewicht des Futters in der Schale misst, um die korrekte Menge sicherzustellen und Über- oder Unterfütterung zu vermeiden.
    - Einbau einer Kamera, um die Katzen zu erkennen.
2. **Software:**
    - Training einer KI, die alle registrierten Katzen erkennen kann.
    - Echtzeit-Erkennung der Katzen, damit die Futterausgabe sofort erfolgt, wenn die richtige Katze vor dem Automaten steht.
    - Steuerungssoftware, die die Hardware steuert.
    - Anlegen und Verwalten von Katzenprofilen, damit jede Katze individuell konfiguriert werden kann.
    - Einstellung der individuellen Futterration pro Tag, um die Ernährung der Tiere zu kontrollieren und anzupassen.
3. **Prototypische Umsetzung:**
    - Zusammenführung von Mechanik, Software und KI in einem funktionierenden Prototypen.
    - Testen der Funktionalität und Optimierung der Abläufe.

Wichtig ist dabei zu betonen, dass es sich bei diesem Projekt um die Entwicklung eines Prototyps handelt. Die Kernfunktionen wie Futterausgabe, Katzenidentifikation und Rationsverwaltung sollen vom Automaten erfüllt werden, jedoch wird er nicht wie ein fertiges Kaufprodukt sein. Einige Abläufe sind noch nicht perfekt optimiert, und es können auch kleinere Probleme auftreten. Das Ziel des Projekts ist es, zu zeigen, wie eine solche Lösung technisch umgesetzt werden kann und eine mögliche Grundlage für weitere Entwicklungen zu schaffen.

Kapitel 2

Umsetzung

2.1 Software

2.1.1 KI Bildererkennung

Bildererkennung ist eine Methode, welche in der Lage ist, aus digitalen Bildern Muster, Objekte oder Strukturen zu identifizieren. Regelbasierte Ansätze wie Farbfilter oder Erkennung von Kanten stossen dabei schnell an ihre Grenzen und liefern unzuverlässige Ergebnisse. An dieser Stelle setzt die Künstliche Intelligenz (KI) an: Das Ziel ist, dass die KI ohne vordefinierte Regeln selbst lernt und eigene Unterscheidungsmerkmale benutzt. (IBM, 2025a)

Vor allem im Bereich der Bildererkennung haben sich KI-Systeme als besonders hilfreich erwiesen. Sie erkennen Muster und Strukturen, die dem Menschen gar nicht auffallen, da sie eigenständig lernen und der Mensch keine Vorgaben gibt. Solche Systeme sind in der heutigen Welt nicht mehr wegzudenken, in vielen Branchen sind sie ein wichtiger Bestandteil, wie z.B. bei medizinischen Diagnosen und bei autonomen Fahrzeugen (Mebis, 2025).

Auch in diesem Projekt spielt die KI-Bildererkennung eine wichtige Rolle, da sie die Grundlage für die Katzenerkennung bildet. Simplere Ansätze wie Farbfilter würden bis zu einem gewissen Grad auch funktionieren, aber sobald man z. B zwei Katzen mit ähnlicher Fellfarbe hat, können diese nicht mehr voneinander unterschieden werden.

2.1.2 Grundlagen der Bilderkennung

Digitale Bilder setzen sich aus einer Vielzahl von einzelnen Bildpunkten zusammen, den sogenannten Pixeln. Jeder Pixel speichert Informationen zur Helligkeit und Farbe, die normalerweise im RGB-Farbraum (Rot, Grün, Blau) dargestellt werden. Für den Computer ist ein Bild lediglich eine grosse Matrix von Zahlenwerten. Während ein Mensch sofort Objekte, Muster oder sogar individuelle Gesichter aus einem Bild erkennt, fehlt einem Computer zunächst das Verständnis dafür, welche Strukturen für eine Unterscheidung wichtig sind. Um aus den Rohdaten sinnvolle Informationen zu generieren, sind Methoden erforderlich, die bestimmte Merkmale, sogenannte Features, aus den Bildern extrahieren. Dies wird heute durch neuronale Netzwerke umgesetzt. (IBM, 2025a)

Neuronale Netzwerke und CNNs

Ein neuronales Netzwerk besteht aus einzelnen Neuronen, die in mehreren Schichten (Layers) organisiert sind: Eingabeschicht, versteckte Schichten und Ausgabeschicht. Jedes Neuron empfängt Eingabewerte, gewichtet sie und gibt ein Ergebnis über eine Aktivierungsfunktion weiter. (Datasolut, 2025)

Im Training werden die Gewichte der Neuronen angepasst, um den Unterschied zwischen den Vorhersagen des Modells und den tatsächlichen Klassenzuordnungen zu minimieren. Dieser Lernprozess erfolgt durch Backpropagation: Der Fehler wird rückwärts durch das Netzwerk propagiert, um die Gewichte schrittweise zu optimieren. (Bergmann & Stryker, 2025)

Moderne Modelle wie Convolutional Neural Networks (CNNs) nutzen spezialisierte Schichten, die lokale Muster erkennen, was insbesondere bei Bildern entscheidend ist (IBM, 2025b).

Ein CNN verwendet sogenannte Filter oder „Convolutional Layer", die sich über das Bild bewegen und dabei lokale Strukturen extrahieren. In den ersten Schichten werden grundlegende Merkmale wie Kanten oder einfache Farbverläufe erkannt. In tiefer gelegenen Schichten entwickeln sich daraus komplexere Muster, wie Fellstrukturen, Augenformen oder charakteristische Körperkonturen. So entsteht in der Hierarchie des Netzwerks eine immer abstraktere Darstellung des Bildes, die letztlich zur Klassifikation verwendet werden kann. (DataScientest, 2023)

Das Pooling, bei dem die Bildinformationen komprimiert werden, stellt einen weiteren zentralen Bestandteil dar. Das heisst, die Auflösung wird verringert, während die wesentlichen Merkmale erhalten bleiben. So wird das Modell robuster gegen kleine Verschiebungen, Drehungen oder Verzerrungen und der Rechenaufwand verringert. Für die Praxis heisst das: Selbst wenn die Katze nicht genau in der Bildmitte sitzt oder die Beleuchtung sich verändert, kann das Modell sie weiterhin zuverlässig identifizieren. (DataScientest, 2023)

Für unser Projekt ist genau diese Robustheit entscheidend. Katzen bewegen sich ungezwungen, und sie werden aus verschiedenen Perspektiven fotografiert, gelegentlich im grellen Sonnenlicht, gelegentlich in einem weniger erleuchteten Raum. Ein striktes, regelgebundenes System würde in derartigen Situationen regelmässig falsche Resultate produzieren. Ein CNN ermöglicht es dagegen, die wesentlichen Merkmale der jeweiligen Katze zu erfassen und sie auch unter verändernden Bedingungen verlässlich zu identifizieren.

Transfer Learning

Transfer Learning ist ein weiterer bedeutender Aspekt der modernen Bilderkennung. Hierbei wird ein Modell, das auf Millionen allgemeiner Bilder bereits trainiert wurde (wie z.B. MobileNetV2) für eine spezifische Aufgabe angepasst (Keras, 2025). In den frühen Schichten des Netzes werden universelle Merkmale wie Kanten oder Farbkontraste dargestellt, die in nahezu allen Bildern vorkommen. Diese können unmittelbar übernommen werden. Nur die letzten Schichten, die für die konkrete Klassifikation verantwortlich sind, werden neu trainiert. Durch dieses Vorgehen wird nicht nur eine enorme Rechenzeit eingespart, sondern es wird auch möglich, ein leistungsfähiges Modell mit relativ wenigen Bildern zu trainieren. Für unser Projekt heisst das: Selbst mit einer begrenzten Anzahl von Katzenfotos kann ein Modell trainiert werden, das zwischen verschiedenen Katzen unterscheiden kann. (IONOS Redaktion, 2024)

Bedeutung von CNNs für die Bilderkennung

Die Verbindung von Convolutional Layers, Pooling, hierarchischem Merkmalsaufbau und Transfer Learning macht CNNs zum Massstab in der heutigen Bilderkennung. Es wäre kaum möglich, eine Aufgabe wie die zuverlässige Unterscheidung zwischen einzelnen Katzen ohne diese Grundlagen zu lösen. Während der Mensch instinktiv und schnell zwischen bekannten Tieren unterscheiden kann, erfordert es für einen Computer präzise Strukturen, um dies zu imitieren. Die KI ermöglicht es, diese Strukturen automatisch zu erlernen und so ein flexibles und zuverlässiges Erkennungssystem zu schaffen.

2.1.3 Datenerfassung

Das KI-Modell arbeitet mit zwei Klassen für die beiden Katzen (Bruno und Fläkli), die jeweils einen Datensatz enthalten, sowie einer zusätzlichen Klasse mit einem Datensatz mit fremden Katzen. Auf diese Weise kann das Modell zwischen registrierten und unbekannten Katzen unterscheiden. Ziel ist es, am Ende für Bruno und Fläkli jeweils ein eigenes Modell zu haben, das zuverlässig zwischen der eigenen Katze und fremden Katzen unterscheidet. Für die Klassen Bruno und Fläkli wurden zunächst Videos von ihnen aufgenommen, sowohl in Innenräumen als auch im Freien, um sie vor unterschiedlichen Hintergründen darzustellen. Anschliessend wurden die Videos mit einem Python-Programm (videos_to_pictures.py) in einzelne Bilder zerlegt. Das ursprüngliche Format der Videos blieb dabei erhalten, da die Anpassung der Bildgrösse erst im Rahmen des Preprocessing vorgenommen werden sollte. Zusätzlich wurde für die zweite Klasse ein Datensatz von der Website Kaggle (Crawford, 2015) verwendet, der eine grosse Anzahl verschiedener Katzenbilder enthält. Pro Klasse wurden nur rund 150 Bilder verwendet. Dank des Einsatzes von Transfer Learning ist diese Anzahl für einen ersten Prototypen jedoch ausreichend.

In einem zweiten Schritt wurden die Datensätze der Klassen Bruno und Fläkli erneuert. Dazu wurden mit einem Python-Programm (make_pictures.py) mit der Raspberry-Pi-Kamera am Einsatzort des Automaten neue Bilder der beiden Katzen aufgenommen. Dadurch sind die Modelle stärker auf die spezifischen Bedingungen des Automaten spezialisiert. Die Bilder fremder Katzen stammen aus einem aktuellen Kaggle-Datensatz (Azmeenasiraj, 2022), in dem ausschliesslich die Köpfe der Katzen zu sehen sind. Dies passt gut zu den mit der Raspberry-Pi-Kamera aufgenommenen Bildern, auf denen ebenfalls nur die Köpfe erkennbar sind, was die Vergleichbarkeit der Datensätze verbessert. Zudem wurde die Anzahl der Bilder pro Klasse auf etwa 300 erhöht, um die Trainingsgrundlage weiter zu verbessern.

Alle Datensätze wurden in einem Googledrive-Ordner gespeichert, sodass das KI-Modell von überall Zugriff auf die Bilder hatte.

2.1.4 Preprocessing der Bilddaten

Wie schon erwähnt, wurden Videos von beiden Katzen aufgenommen. Aus diesen Videos wurden mit einem Python-Programm einzelne Bilder erstellt. Als diese Bilder dann verwendet wurden, um die KI zu trainieren, erkannte sie die Katzen sehr unzuverlässig. Das Problem war, dass auf manchen Bildern gar keine Katze zu sehen war oder nur kleine Teile der Katze erkennbar waren. In diesen Fällen analysierte die KI nicht die Katze, sondern vor allem den Hintergrund. Das war ein grosses Problem, weil die KI lernen sollte, die Katzen zu erkennen, und nicht den Hintergrund. Die Lösung für dieses Problem war, der KI nur die Bildausschnitte zu zeigen, auf denen die Katze deutlich zu sehen war und die möglichst wenig Hintergrund enthielten. Es wäre aber zu viel Zeitaufwand gewesen, jedes Bild manuell zuzuschneiden. Deshalb wurde ein vortrainiertes KI-Modell namens YOLOv8 (Jocher et al., 2023) verwendet. Dieses Modell kann automatisch Katzen in Bildern erkennen und eine sogenannte „Bounding-Box“ um die Katze ziehen. Dabei handelt es sich um eine rechteckige Box, die über Koordinaten angibt, wo sich das Objekt im Bild befindet. Mit dieser Box konnten die Bilder automatisch so zugeschnitten werden, dass nur die Katze zu sehen ist. Am Ende musste noch sichergestellt werden, dass beide Datensätze gleich viele Bilder enthalten. Das ist wichtig, damit die KI nicht annimmt, dass eine Katze häufiger vorkommt als die andere und dadurch die Ergebnisse verfälscht werden. Auch das erledigte ein Python-Programm automatisch, indem es in allen Ordnern die Anzahl der Bilder auf die des kleinsten Ordners reduzierte und dabei überschüssige Bilder löschte. Schliesslich wurden beide Programme, das Zuschneiden mit YOLOv8 und das Angleichen der Bilder, zu einem einzigen Preprocessing-Programm (preprocess_images.py) zusammengeführt. So können die Bilder automatisch vorbereitet werden, bevor die KI damit trainiert wird.

2.1.5 Training des Modells

Für Bruno und Fläkli wurde jeweils ein eigenes KI-Modell trainiert. Eines unterscheidet zwischen Bruno und allen fremden Katzen, das andere zwischen Fläkli und allen fremden Katzen. Nach den zuvor beschriebenen Schritten der Datenerfassung und des Preprocessing muss das jeweilige Modell nun trainiert werden.

Beim Training eines neuronalen Netzes werden die Daten in mehrere Mengen unterteilt: Trainings-, Validierungs- und Testdaten. Die Trainingsdaten werden verwendet, um die Gewichte des Modells anzupassen. Die Validierungsdaten dienen dazu, die Leistung des Modells während des Trainings zu prüfen und zu erkennen, wenn das Modell die Trainingsdaten „auswendig lernt“, ein Phänomen, das als Overfitting bezeichnet wird. Die Testdaten dienen dazu, das fertig trainierte Modell einmalig auf völlig neuen, bisher ungesehenen Bildern zu überprüfen. Auf diese Weise kann man objektiv beurteilen, wie gut das Modell generalisiert, ohne dass es auf diese Daten schon vorbereitet wurde. (Ultralytics, 2025b)

Das Training erfolgt in Epochen, also vollständigen Durchläufen durch den Trainingsdatensatz. Um Speicher und Rechenleistung zu optimieren, werden die Daten in Batches aufgeteilt, kleine Gruppen von Bildern, die nacheinander durch das Netzwerk laufen. So kann das Modell Schritt für Schritt lernen, während die Gewichte nach jedem Batch angepasst werden. (Ultralytics, 2025a)

Das Training der Modelle wurde auf Google Colab durchgeführt. Diese Plattform bietet eine kostenlose GPU-Unterstützung, eine einfache Einbindung von Google Drive und eine sofort einsatzbereite Programmierumgebung, was sie ideal für das effiziente Trainieren von KI-Modellen ohne eigene Hochleistungs-Hardware macht. Es wurde dabei die kostenlose Version von Colab verwendet, wodurch der Zugriff auf GPUs nicht immer gewährleistet war. Dank der Nutzung von Transfer Learning war jedoch keine besonders hohe Rechenleistung erforderlich, sodass das Training dennoch reibungslos durchgeführt werden konnte. (zITBOx Redaktion, 2024)

Erklärung des Trainingsprogramms

Im Folgenden wird das Trainingsprogramm (train_model.py) erklärt:

Zunächst werden die benötigten Bibliotheken importiert. Diese beinhalten unter anderem TensorFlow und Keras, die das Erstellen und Trainieren von neuronalen Netzen ermöglichen, sowie Matplotlib, das für die Visualisierung von Bildern verwendet wird. Pandas wird genutzt, um die Bilddaten in einer übersichtlichen Tabelle zu organisieren. Über Google Colab wird zudem der Zugriff auf die eigenen Dateien ermöglicht, wodurch auf die Bilddaten auf Google Drive zugegriffen werden kann.

```
import os
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from keras import layers
from google.colab import drive
drive.mount("/content/drive")
```

Daraufhin erfolgt die Definition der Bildgrösse, der Katze und des Pfads zum Ordner mit den Datensätzen. Die Daten werden in einem DataFrame zusammengefasst. Jede Zeile entspricht einem Bild, und es wird festgehalten, ob es sich um die eigene Katze oder um andere Katzen handelt. Diese Struktur erleichtert das Training, da das Modell die Bilder automatisch den richtigen Kategorien zuordnen kann. Anschliessend wird ein sogenannter ImageDataGenerator verwendet. Dies ist ein Werkzeug, das die Bilder für das Training vorbereitet und gleichzeitig zufällige Veränderungen wie Drehungen, Verschiebungen oder Spiegelungen erzeugt. Solche Veränderungen heissen „Augmentation“ und dienen dazu, das Modell robuster zu machen, indem es lernt, die Katze auch unter leicht veränderten Bedingungen zu erkennen. Zusätzlich werden

die Pixelwerte der Bilder normalisiert, das heisst, die Werte werden von 0 bis 255 auf einen Bereich von 0 bis 1 skaliert. Dies ist wichtig, damit das neuronale Netz stabiler und schneller lernt, da grosse Zahlen das Training erschweren würden.

```
IMG_SIZE = (180, 180)
IMG_SIZE_ = 180
BATCH_SIZE = 16
CAT = "bruno"
dataset_name = f"dataset_{CAT[:3]}_other"
VERSION = "4"

CLASS_0_DIR = f"/content/drive/MyDrive/datasets/{DATASET_NAME}/preprocessed_images/{CAT}"
CLASS_1_DIR = f"/content/drive/MyDrive/datasets/{DATASET_NAME}/preprocessed_images/other_cats"

class_0_images = [os.path.join(class_0_dir, fname) for fname in os.listdir(class_0_dir)]
class_1_images = [os.path.join(class_1_dir, fname) for fname in os.listdir(class_1_dir)]

data = pd.DataFrame({
    "filename": class_0_images + class_1_images,
    "class": [0] * len(class_0_images) + [1] * len(class_1_images)
})

data['class'] = data['class'].astype(str)

datagen = keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=40,
    width_shift_range=0.26,
    height_shift_range=0.26,
    shear_range=0.17,
    zoom_range=0.26,
    horizontal_flip=True,
    fill_mode="nearest"
)
```

Die vorbereiteten Daten werden in zwei Gruppen aufgeteilt: Trainings- und Validierungsdaten.Das Training findet auf 80% der Bilder statt und die Validierung nur auf 20%. Mit dem Befehl flow_from_dataframe werden die Bilder in Batches geladen, sodass das Modell nicht alle Bilder gleichzeitig im Speicher halten muss.

```
train_generator = datagen.flow_from_dataframe(
    dataframe=data,
    x_col="filename",
    y_col="class",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="binary",
    subset="training",
    shuffle=True,
    seed=42,
    validation_split=0.2
)


val_generator = datagen.flow_from_dataframe(
    dataframe=data,
    x_col="filename",
    y_col="class",
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="binary",
    subset="validation",
    shuffle=True,
    seed=42,
    validation_split=0.2
)
```

Die Klassennamen werden ausgegeben, wobei die eigene Katze (Bruno oder Fläkli) die Zahl 0 erhält und alle anderen Katzen die Zahl 1. Anschliessend wird ein Batch von Bildern und den zugehörigen Labels aus dem Trainingsgenerator geladen. Die ersten neun Bilder aus diesem Batch werden in einem 3×3-Raster dargestellt. Jedes Bild wird mit seinem Label angezeigt, und

die Achsen werden ausgeblendet, damit die Darstellung übersichtlich bleibt. Auf diese Weise lässt sich leicht überprüfen, ob die Bilder korrekt geladen und die Labels richtig zugeordnet wurden.

```
print("Class names:", CAT, "= 0 ,", "other_cats", "= 1")

images, labels = next(train_generator)

plt.figure(figsize=(10, 10))
for i in range(min(9, BATCH_SIZE)):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i])
    plt.title(f"Label: {int(labels[i])}")
    plt.axis("off")
plt.show()
```

Für das eigentliche Modell wird MobileNetV2 (Keras, 2025) verwendet, ein vortrainiertes neuronales Netz, das bereits auf Millionen von Bildern trainiert wurde. Durch die Verwendung dieses „vortrainierten“ Modells kann das Netz schneller lernen, da es bereits grundlegende Merkmale wie Kanten und Formen erkennt. Das Modell wird so angepasst, dass nur der obere Teil neu trainiert wird. Diese oberen Schichten sind dabei eigentlich die letzten Schichten im Modell, die die bereits extrahierten Merkmale des vortrainierten Netzwerks nutzen, um die finale Entscheidung zu treffen. Danach wird eine Pooling-Schicht hinzugefügt, die die Merkmale zusammenfasst, und zwei Dense-Schichten, die die finale Entscheidung treffen, ob das Bild „Bruno“ oder „andere Katze“ zeigt. Die Aktivierungsfunktion „sigmoid“ sorgt dafür, dass das Ergebnis als Wahrscheinlichkeit zwischen 0 und 1 ausgegeben wird. Das Modell wird mit der Verlustfunktion „binary_crossentropy“ und dem Optimierer „Adam“ trainiert. Die Verlustfunktion misst, wie stark die Vorhersagen des Modells von den tatsächlichen Werten abweichen, und der Optimierer passt die Gewichte des Netzes so an, dass dieser Fehler minimiert wird. Während des Trainings werden die Genauigkeit und der Verlust sowohl für Trainings- als auch für Validierungsdaten protokolliert. So kann überprüft werden, ob das Modell gut generalisiert und nicht nur die Trainingsdaten auswendig lernt.

```
base_model = keras.applications.MobileNetV2(input_shape=(IMG_SIZE_, IMG_SIZE_, 3), include_top
    =False, weights="imagenet")
base_model.trainable = False

x = layers.GlobalAveragePooling2D()(base_model.output)
x = layers.Dense(128, activation="relu")(x)
x = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs=base_model.input, outputs=x)

model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])

model.fit(train_generator, validation_data=val_generator, epochs=5)

model_path = f"/content/drive/MyDrive/repos/keras-distinguish-own-cat-from-others-model/models
    /model{VERSION}_{CAT}.keras"
model.save(model_path, save_format="keras")
```

Während des Trainings zeigt Keras/TensorFlow für jede Epoche den Fortschritt an. Zum Beispiel:

```
Epoch 1/5
17/17  68s 4s/step - accuracy: 0.5879 - loss: 0.9097 - val_accuracy: 0.9231 - val_loss: 0.2881
Epoch 2/5
17/17  13s 728ms/step - accuracy: 0.9502 - loss: 0.1797 - val_accuracy: 0.9385 - val_loss: 0.2080
Epoch 3/5
17/17  14s 782ms/step - accuracy: 0.9551 - loss: 0.1379 - val_accuracy: 0.9846 - val_loss: 0.0914
Epoch 4/5
17/17  14s 828ms/step - accuracy: 0.9485 - loss: 0.1400 - val_accuracy: 0.9692 - val_loss: 0.0757
Epoch 5/5
17/17  21s 843ms/step - accuracy: 0.9939 - loss: 0.0494 - val_accuracy: 0.9692 - val_loss: 0.0573
```

Hier erkennt man, wie der Trainingsverlust (loss) abnimmt und die Genauigkeit (accuracy) steigt. Gleichzeitig zeigen die Werte für die Validierungsdaten (val_loss, val_accuracy), dass das Modell auch auf unbekannten Bildern immer besser wird, was ein Hinweis auf eine gute Generalisierung ist.

2.1.6 Evaluation und Fehleranalyse

Nach dem Training wird das zuvor gespeicherte Modell wieder geladen. Anschliessend wird das Modell mit den Validierungsdaten getestet, um den Verlust und die Genauigkeit zu berechnen. Diese Werte werden dann ausgegeben.

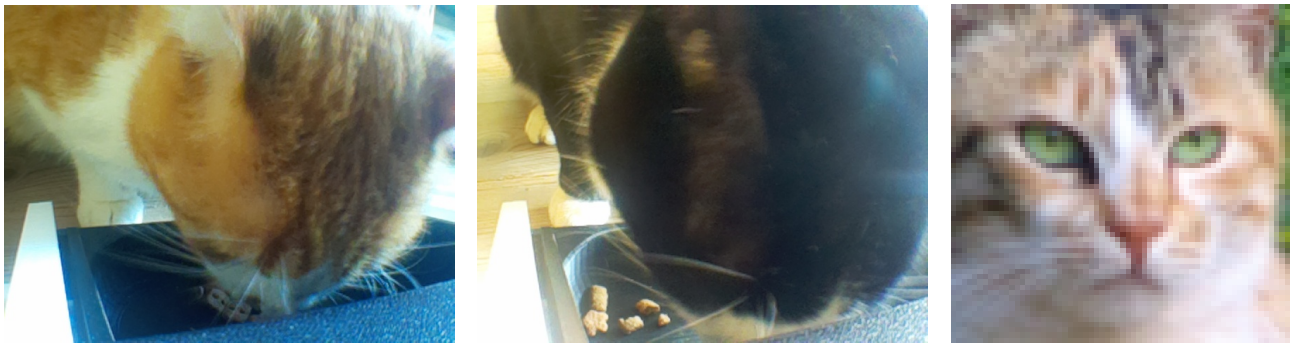
```
model = tf.keras.models.load_model(model_path)

loss, accuracy = model.evaluate(val_generator)

print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

In dieser Evaluation zeigte das Modell, welches mit dem ersten Datensatz trainiert wurde, eine Genauigkeit von etwa 92 % für Bruno und 94 % für Fläkli. Das bedeutet, dass die KI die jeweiligen Katzen auf bisher, während des Trainings nicht zur Anpassung der Gewichte verwendeten, Bilder mit hoher Zuverlässigkeit identifizieren konnte.

Um die Zuverlässigkeit weiter zu prüfen, wurden dem Modell zusätzliche, völlig neue Bilder (Testdaten) gezeigt, die es zuvor nicht gesehen hatte. Dabei wurden beide Katzen immer noch mit einer Wahrscheinlichkeit von etwa 90 % korrekt erkannt.

Im letzten Test musste das Modell Katzen klassifizieren, die Bruno und Fläkli stark ähneln. Hier zeigte sich, dass das Modell Schwierigkeiten hat: Katzen mit ähnlicher Fellfarbe oder Struktur wurden teilweise fälschlicherweise als Bruno oder Fläkli erkannt.

Die Modelle, die mit dem zweiten Datensatz, also dem vor Ort mit der Raspberry Pi-Kamera aufgenommenen, trainiert wurden, erzielten beide eine Genauigkeit von 99 % auf den Validierungsdaten. Dies liegt daran, dass bei den Datensätzen von Bruno und Fläkli die Katzen eher von oben fotografiert wurden und vor allem der obere Teil des Kopfes zu sehen war. Beim Datensatz mit den fremden Katzen war hingegen das Gesicht meist von vorne abgebildet. Das bedeutet, dass die Modelle mit dem zweiten Datensatz schlechter zwischen den registrierten Katzen und fremden Katzen unterscheiden können, dafür aber Bruno und Fläkli schneller und zuverlässiger erkennen, wenn sie am Automaten fressen.



(a) Bruno (b) Fläkli (c) Fremde Katze

Abbildung 2.1: Bilder aus dem zweiten Datensatz

Zudem kam es gelegentlich zu Fehlklassifikationen anderer Objekte, beispielsweise wurde ein oranger Ball mit fellartiger Struktur als Bruno identifiziert. Dies liegt daran, dass das Modell nicht darauf trainiert wurde, zwischen „Katze“ und „Nicht-Katze“ zu unterscheiden, sondern ausschliesslich zwischen den eigenen Katzen und fremden Katzen.

Daraus lassen sich die Grenzen des Modells klar erkennen. Eine Verbesserung wäre möglich, indem man die Datenmenge erhöht oder die Auflösung der Eingabebilder von 180×180 Pixel erweitert, um feinere Fellstrukturen besser zu erfassen.

2.1.7 Steuerungssoftware

Das Hauptprogramm (main.py) des Katzenfutterautomaten ist darauf ausgelegt, automatisch zu erkennen, wann eine Katze am Napf ist, welche Katze es ist, und dementsprechend den Napf zu öffnen, zu schliessen und Futter nachzufüllen. Dazu wird eine Kombination aus YOLOv8-Objekterkennung und individuellen Keras-Modellen genutzt.

Die folgenden zwei Diagramme veranschaulichen den Ablauf des Programms. Anschliessend wird das Hauptprogramm des Katzenfutterautomaten im Detail erläutert.

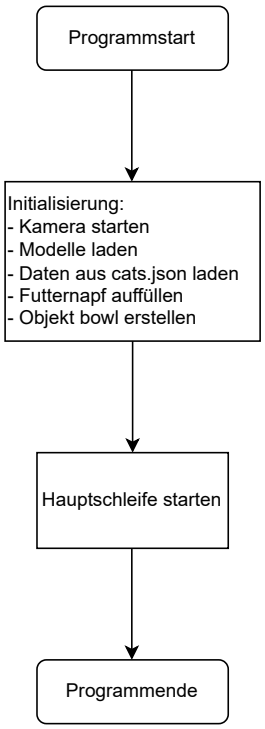


Abbildung 2.2: Gesamtprogramm des Katzenfutterautomaten

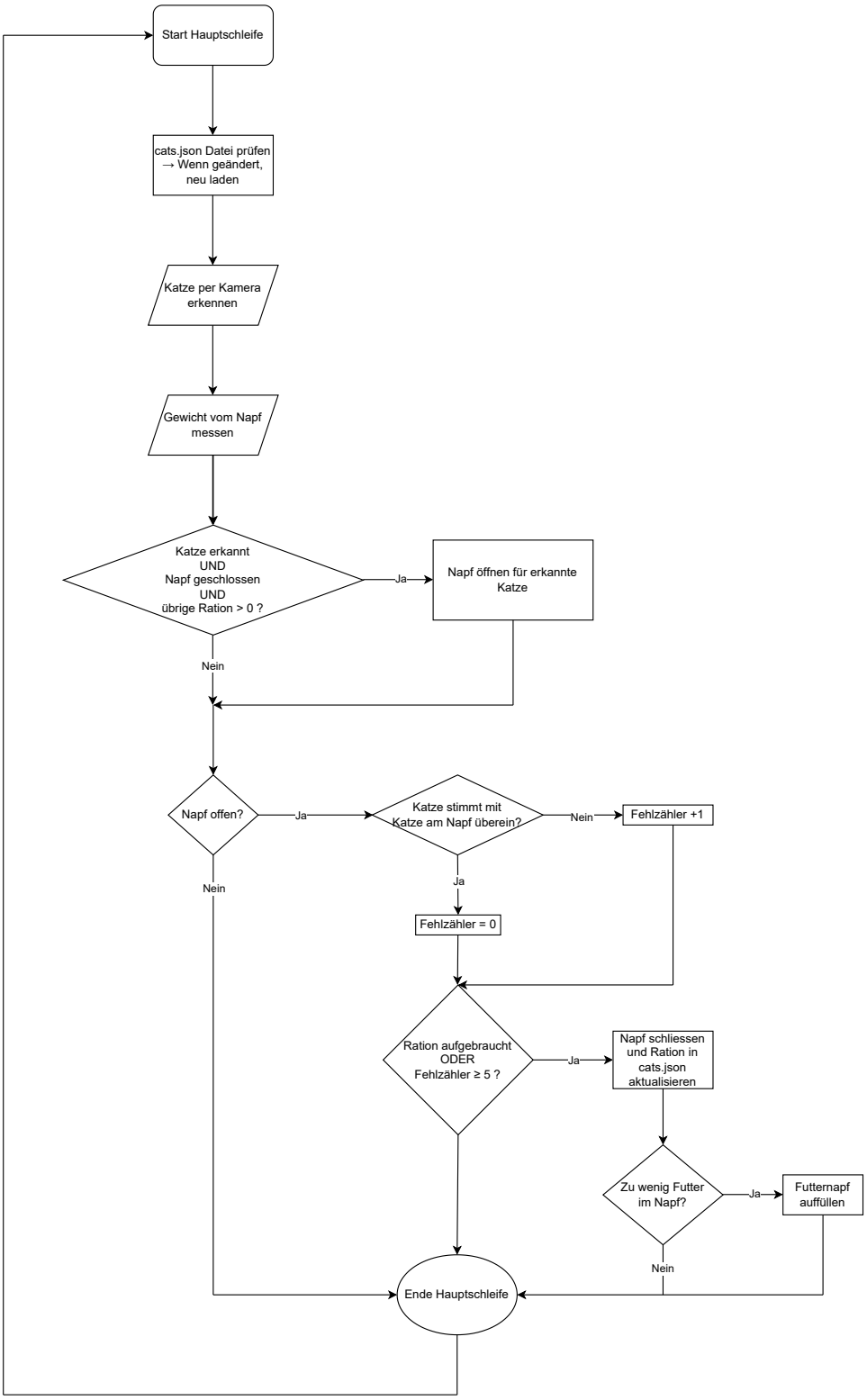


Abbildung 2.3: Hauptschleife des Katzenfutterautomaten

Zu Beginn werden alle nötigen Ressourcen geladen: Die Kamera wird initialisiert, die KI-Modelle für jede Katze werden geladen und die Motoren sowie der Gewichtssensor werden vorbereitet. Anschliessend startet das Hauptprogramm, das in einer Endlosschleife kontinuierlich prüft:

- Ob eine Katze vor dem Napf steht.
- Welche Katze erkannt wurde.
- Das aktuelle Gewicht des Futters im Napf.

Die Steuerlogik funktioniert folgendermassen:

1. Wenn eine bekannte Katze erkannt wird und der Napf geschlossen ist, wird der Napf geöffnet. Die Katzenerkennung funktioniert dabei wie folgt: Zuerst wird ein Bild aufgenommen und mit dem YOLOv8-Modell geprüft, ob eine Katze darauf zu sehen ist. Falls eine Katze erkannt wird, wird das Bild auf den Bereich der Katze zugeschnitten. Anschliessend werden die zugeschnittenen Bilder beiden individuellen Keras-Modellen der eigenen Katzen (Bruno und Fläkli) gegeben. Jedes Modell gibt eine Wahrscheinlichkeit zurück, dass die erkannte Katze die jeweilige Katze ist. Liegt die Wahrscheinlichkeit über 25 %, wird die Katze als mögliche Übereinstimmung betrachtet. Falls beide Modelle Wahrscheinlichkeiten über 25 % liefern, wird die Katze mit der höheren Wahrscheinlichkeit ausgewählt.
2. Solange der Napf geöffnet ist, überwacht das System, ob die richtige Katze am Napf bleibt und ob die tägliche Futterration noch nicht aufgebraucht ist.
3. Der Napf wird geschlossen, wenn eine andere Katze mehr als vier Mal erkannt wird oder die Futterration der aktuellen Katze aufgebraucht ist.
4. Nach dem eine Katze gefressen hat, wird der Napf wiederaufgefüllt.

2.2 Hardware

2.2.1 Elektronikkomponenten

Für den Katzenfutterautomaten werden mehrere verschiedene Komponenten benötigt: ein Raspberry Pi, eine Kamera, ein Motor und eine Waage.

Als Steuerzentrale kommt ein Raspberry Pi 5 mit 8 GB RAM zum Einsatz (Galaxus, 2025). Dieser eignet sich besonders gut für das Projekt, da er ausreichend Rechenleistung hat, um KI-Programme auszuführen. Zudem kann der Raspberry Pi direkt mit Python programmiert werden, was ein Vorteil ist, da alle verwendeten KI-Modelle in Python implementiert sind.

Für die Bilderkennung wird das Raspberry Pi Camera Module v2 mit 8 MP verwendet (Raspberry Pi Foundation, 2025). Die Kamera hat eine ausreichend gute Auflösung, da die Bilder für die KI später auf 180 × 180 Pixel verkleinert werden. Um die Kamera mit dem Raspberry Pi zu verbinden, wird zusätzlich ein offizielles Raspberry Pi Display-Kabel (DSI/-MIPI, 500 mm) benötigt (Pi-Shop.ch, 2025).

Für den Antrieb wird ein Joy-IT Schrittmotor NEMA17-03 mit 0,2 Nm Drehmoment benutzt (Joy-IT, 2025a). Ein Schrittmotor ist ein Motor, der sich nicht kontinuierlich dreht, sondern sich in definierten Schritten bewegt (StudySmarter Redaktion, 2025). Dadurch kann man die Anzahl der Umdrehungen präzise steuern, was für den Automaten entscheidend ist, um den Napf exakt zu öffnen und zu schliessen.

Zur Steuerung des Schrittmotors wird der Joy-IT SBC-MD-TB6600 Schrittmotor-Treiber verwendet (Joy-IT, 2025b). Dieser verstärkt die Signale des Raspberry Pi, steuert die Phasen des Motors und ermöglicht eine präzise Kontrolle von Richtung und Geschwindigkeit.

Zusätzlich benötigt man ein Netzteil, das die notwendige Spannung liefert, sowie einen passenden Adapter, um den Treiber korrekt anzuschliessen.

Für die Gewichtsmessung des Futters kommt die Wägezelle Joy-IT SEN-HX711-01 zum Einsatz (Joy-IT, 2025c). Sie misst Dehnungen und Stauchungen im Metall und kann daraus das Gewicht berechnen (Bosche, 2025), jedoch ist sie nur für Gewichte bis zu einem Kilogramm ausgelegt (Joy-IT, 2025c). Die Wägezelle ist einfach zu montieren und gut mit dem Raspberry Pi kompatibel, was die Integration in den Automaten erleichtert.

Schliesslich werden Jumper-Kabel und ein Steckbrett benötigt, um alle Komponenten problemlos und übersichtlich miteinander zu verbinden. Dadurch lässt sich der Automat präzise steuern und zuverlässig betreiben.

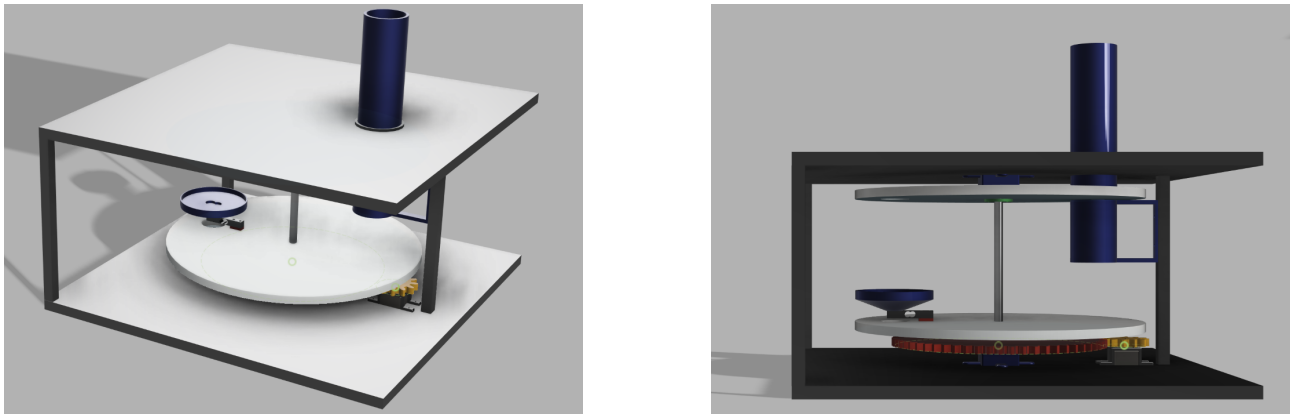
2.2.2 Mechanische Konstruktion

Mechanischer Aufbau und Anforderungen

Für eine zuverlässige Funktion des Futterautomaten ist dessen mechanischer Aufbau zentral. Für eine planmässige Funktion des Automaten muss er die Futterschale selbst öffnen und schliessen sowie Futter eigenständig nachfüllen können. Zudem soll es möglich sein, eine bestimmte Menge an Futtervorrat zu lagern, um ein ständiges manuelles Befüllen des Geräts zu vermeiden. Die Konstruktion wurde mit dem Ziel gestaltet, die Mechanik so einfach wie möglich zu halten, um mögliche Fehlerquellen zu reduzieren und den Aufbau zu erleichtern. Um die Steuerung zu erleichtern und technische Probleme zu verhindern, wird der gesamte Mechanismus nur von einem einzigen Motor angetrieben.

Erstes Konzept: Drehteller mit zwei Schalen

Vom ersten Konzept bis zum endgültigen Prototypen wurde die Konstruktion schrittweise entwickelt. Anfangs war die Idee, einen Drehteller mit zwei Futterschalen zu konstruieren. Er sollte in der Lage sein, sich so zu drehen, dass immer eine der beiden Schalen sichtbar und die andere verborgen ist. Das Futter wäre durch ein Rohr in die Schale gefallen, während das Drehen des Tellers das Rohr geöffnet oder geschlossen hätte. Im Verlauf der Zeit stellte sich indes heraus, dass auch eine einzelne Futterschale genügen würde. Der Automat kann durch geschickte Steuerung und Programmierung so konzipiert werden, dass mehrere Katzen am selben Gerät essen können, ohne sich gegenseitig zu behindern. Dies ermöglichte eine deutliche Vereinfachung des Aufbaus, ohne dass die Funktionalität beeinträchtigt wurde. Trotzdem stellte die Mechanik eines solchen Drehtellers hohe technische Anforderungen. Hochwertige Lagerungen sind notwendig, um Reibung und Spiel zu minimieren, damit sich der Teller gleichmässig und präzise bewegen kann. Diese Lösung hätte einen hohen mechanischen Aufwand erfordert, was den Bau und die Wartung erschwert hätte. Aus diesen Gründen wurde das Konzept letztlich verworfen und weiter vereinfacht.



(a) 3D-Ansicht (b) Seitenansicht

Abbildung 2.4: Drehteller-Konzept mit einer Schale

Weiterentwicklung: Schiebemechanismus

Die nächste Entwicklungsstufe brachte die Idee hervor, eine stationäre Schale zu nutzen, über der sich eine Platte horizontal bewegt, um die Schale zu öffnen oder zu schliessen. In der Platte wurde ein Loch vorgesehen, das über der Futteröffnung des Vorratsbehälters liegt. Beim Vorwärtsbewegen der Platte kommt dieses Loch exakt über der Öffnung zu liegen, wodurch das Futter in die Schale fällt. Allerdings stellte sich in der Praxis heraus, dass diese Lösung einige

Probleme hervorrief. Da der Futtervorrat unmittelbar über der Schale positioniert war, hatte die Katze zu wenig Platz zum Fressen. Ausserdem lag das Futter oft an einer Stelle der Schale zusammen, wodurch die Öffnung verstopft wurde.

Optimierung: Neigung der Konstruktion

Die entscheidende Verbesserung lag darin, die gesamte Konstruktion zu neigen. Dadurch gelangt das Futter automatisch in den unteren Teil der Schale, was Verstopfungen vorbeugt. Zugleich kann die Platte weiterhin zum Schliessen und Nachfüllen verwendet werden. Eine Zahnspange, die mit einem Zahnrad an dem Motor gekoppelt ist, liegt unter der Platte. Die Drehbewegung des Motors bewegt die Platte nach oben oder unten, wodurch das Öffnen, Schliessen und Nachfüllen gesteuert wird. Die Futterschale ist mit einer Wägezelle verbunden, was eine genaue Messung des Futtergewichts ermöglicht. Auf diese Weise kann überprüft werden, ob die Katze ihre gesamte Ration gefressen hat oder ob noch Futter übrig ist.

Als Träger für alle weiteren Komponenten dient eine stabile Holzplatte, die den Automatenboden ausmacht. Die anderen Bauteile wurden mit Fusion 360 entworfen und danach mit 3D-Druckern hergestellt. Da PLA einfach zu drucken und ausreichend stabil ist, bestehen die meisten Teile aus diesem Material (Filamentworld, 2025b). Als Material für alle Bauteile, die unmittelbar mit dem Futter in Berührung kommen, wurde PETG gewählt. Es gilt als lebensmittelecht und weist eine glattere sowie weniger poröse Oberfläche auf (Filamentworld, 2025a). Trotzdem muss hervorgehoben werden, dass diese Lösung nur für den Prototypen geeignet ist, da bei einer tatsächlichen Serienproduktion von Geräten mit Lebensmittelkontakt strenge Vorschriften gelten würden.

Eine Herausforderung bestand darin, das Zahnrad zuverlässig mit der Motorachse zu verbinden. Bei vielen Schrittmotoren ist die Achse abgeflacht oder oval, was dafür sorgt, dass aufgesteckte Zahnräder sich nicht verdrehen. Der Motor, der hier verwendet wurde, hatte jedoch eine vollkommen runde Achse, was dem Zahnrad beim Betrieb eine leichte Drehung ermöglichte. Um das zu vermeiden, wurde das Zahnradloch zunächst absichtlich etwas zu klein gedruckt, um die Reibung zu erhöhen. Das Zahnrad wurde zudem so entworfen, dass ein leichtes Zusammenziehen mit einer Schraube und einer Mutter möglich ist. Es klemmt dadurch auf der Achse fest und kann sich nicht mehr drehen. Diese Lösung ist einfach, aber effektiv und ermöglicht einen zuverlässigen Betrieb des Antriebs, ohne dass zusätzliche Komponenten erforderlich sind.

Die Kamera wurde oberhalb des Napfs befestigt. Da sie jedoch nur einen kleinen Blickwinkel hatte und somit nicht ausreichend abdeckte, um die Katze zuverlässig im Bild zu erfassen, wurde ein Fischauge auf die Kamera montiert, das den Blickwinkel vergrösserte.

Auf diese Weise entstand ein Futterautomat, der funktional, robust und kompakt ist. Seine Mechanik kommt mit nur einem Motor aus und kann dennoch alle erforderlichen Bewegungen – Öffnen, Schliessen und Nachfüllen – ausführen. Der Prototyp stellt durch die Verbindung einfacher mechanischer Prinzipien mit einem zielgerichteten Einsatz von Material eine technisch sinnvolle und praxistaugliche Lösung dar.

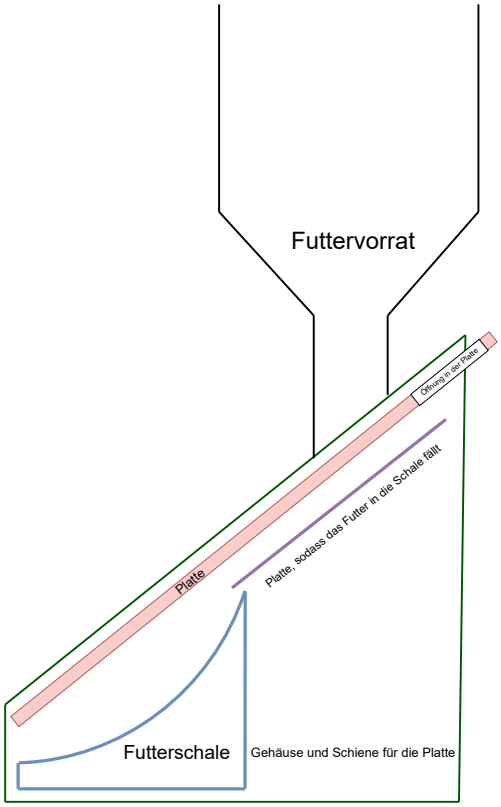
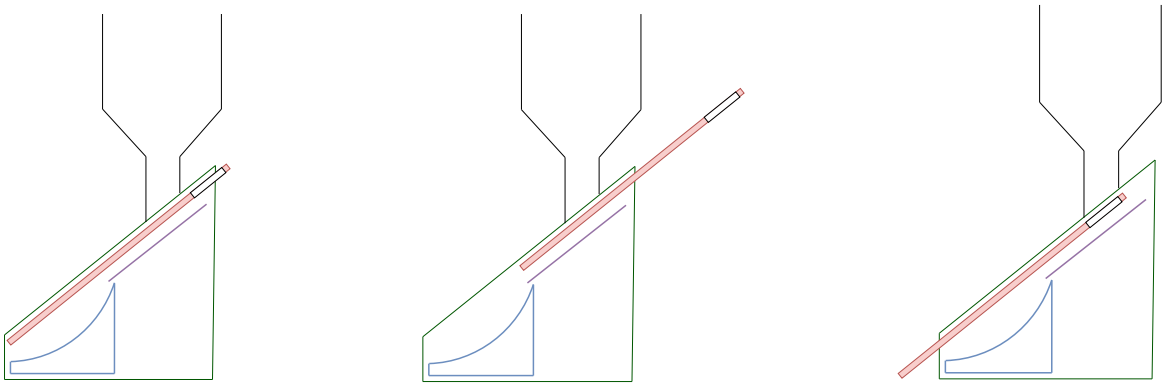


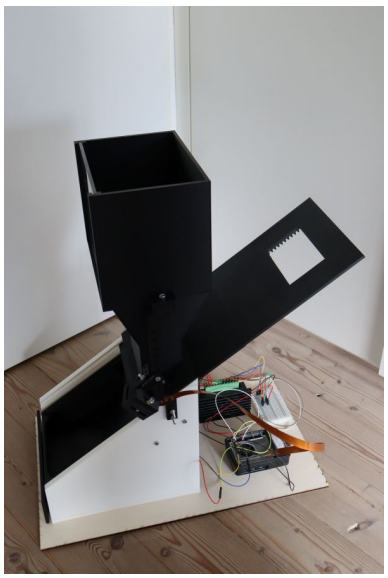
Abbildung 2.5: Beschrifteter Aufbau des Futterautomaten



(a) Geschlossen (b) Offen (c) Auffüllen

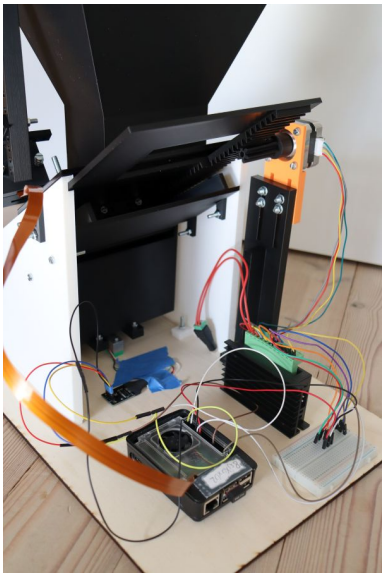
Abbildung 2.6: Die drei Zustände des Automaten

(a) Geschlossen

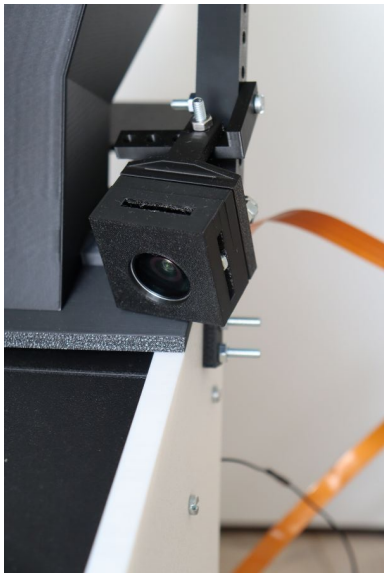


(b) Offen

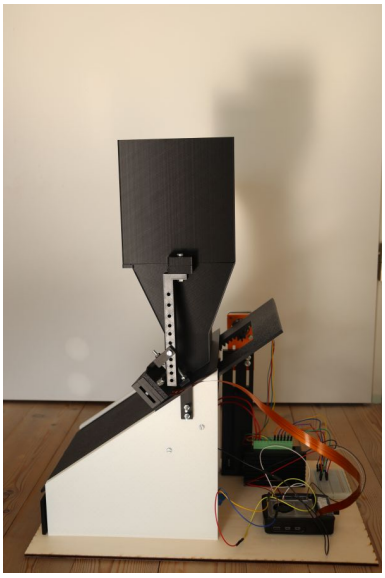
(c) Auffüllen



(d) Verkabelung



(e) Kamera mit Fischauge



(f) Seitenansicht

Abbildung 2.7: Fotos des realen Automaten-Prototyps

2.2.3 Test und Inbetriebnahme

In diesem Abschnitt werden die durchgeführten Tests und die anschliessende Inbetriebnahme des Systems beschrieben. Es sollte kontrolliert werden, ob alle Teile wie vorgesehen arbeiten und ob das Gesamtsystem den Vorgaben entspricht. Der erste Test ergab, dass die Platte nach einer Abwärtsbewegung und anschliessendem Hochfahren mit der gleichen Umdrehungsanzahl nicht wieder am selben Punkt war, sondern etwas zu tief stand. Langfristig würde dieses Verhalten Probleme verursachen, da sich die Platte mit der Zeit immer weiter verschieben würde, bis ein richtiges Öffnen, Schliessen oder Nachfüllen von Futter nicht mehr möglich wäre. Als Erstes wurde im Programm nach einem potenziellen Fehler gesucht, wie etwa, dass die Anzahl der Umdrehungen beim Hoch- und Runterfahren unterschiedlich ist. Das war jedoch nicht der Fall. Es entstand der Verdacht, dass die Schwerkraft beim Herunterfahren eine Rolle spielt und die Platte dadurch etwas weiter nach unten gezogen wird, während der Motor beim Hochfahren dagegen ankämpfen muss und deshalb leicht zu wenig dreht. Es wurde nach einigem Nachden-

ken deutlich, dass das Problem in dem Zahnrad begründet ist, das ein zu grosses Drehmoment aufweist. Beim Herunterfahren ist das Drehmoment so hoch, dass der Motor die Kontrolle kurzzeitig verliert und ein paar Umdrehungen zu viel macht. Auch beim Hochfahren passiert das Gleiche, das Drehmoment ist zu gross, weshalb der Motor ein paar Umdrehungen nicht schafft. Ein kleineres Zahnrad wurde als Lösung eingebaut. Dadurch verringerte sich die Kraftübertragung, der Motor musste zwar mehr Umdrehungen machen, aber die Bewegung wurde deutlich präziser. Nach dem Austausch funktionierte die Plattenverschiebung zuverlässig, und nach mehreren Tests war sie am Ende wieder genau in der Ausgangsposition.

Auch bei der Waage trat ein weiteres Problem auf. Im Laufe der Zeit driftete diese leicht nach oben, also in den positiven Bereich. Die Abweichung lag innerhalb einer Stunde bei etwa 0,3 g. Auch wenn das nach wenig klingt, summiert es sich über längere Zeit stark: Nach einer Woche sind es bereits über 50 g und nach einem Jahr fast 18 kg. Ein solcher Fehler führt dazu, dass der Automat unzuverlässig wird, genau das sollte verhindert werden. Temperatur- oder Feuchtigkeitsschwankungen sowie eine ungleichmässige Gewichtsverteilung können unter anderem die Ursache für die Abweichungen sein. In diversen Online-Foren teilten andere Nutzer ähnliche Erfahrungen, und die Mehrheit war der Ansicht, dass eine hardwareseitige Lösung nicht möglich sei. Eine mögliche Softwarelösung besteht darin, die Waage direkt vor dem Öffnen des Futternapfs auf null zu kalibrieren. Nach dem Fressen der Katze wird die Gewichtsdifferenz als tatsächlich gefressene Menge erfasst. Anschliessend kann der Futternapf wieder so aufgefüllt werden, dass das Gewicht erneut null beträgt. Auf diese Weise hat das Abdriften der Waage nur einen minimalen Einfluss, da die Waage während der kurzen Fressenszeit kaum driftet.

Zusätzlich wurde während der Tests festgestellt, dass der Futtervorratsbehälter zu klein war, um über mehrere Tage hinweg genügend Futter zu lagern, und dass das Futter sich gelegentlich verklemmte. Um beide Probleme zu lösen, wurde der Behälter vergrössert und verbreitert. Der Automat arbeitete nach all diesen Anpassungen zuverlässig. Zwar waren Bruno und Fläkli zu Beginn noch etwas skeptisch, aber sie gewöhnten sich schnell an das Gerät und frassen schliesslich problemlos daraus.

Kapitel 3

Diskussion

3.1 Bewertung

Der entwickelte Katzenfutterautomat funktioniert im Wesentlichen wie vorgesehen. Er kann die Katzen sicher identifizieren, den Futterbehälter öffnen und schliessen sowie Futter nachfüllen. Somit wurden alle gesetzten Ziele erreicht. Es ist jedoch nach wie vor ein klarer Prototyp: Optisch wirkt es eher funktional als ansprechend, und es ist noch weit vom fertigen, verkaufsfertigen Produkt entfernt. In erster Linie war es das Ziel der Arbeit, die Funktionalität zu gewährleisten, weniger ein marktfähiges fertiges Produkt zu entwickeln. Bei der Arbeit kamen unterschiedliche Probleme auf. Gerade beim Programmieren der KI fiel es mitunter schwer, die besten Lösungen mit den gegebenen Ressourcen zu realisieren. Ich musste nicht nur die Erkennung selbst entwickeln, sondern auch darüber nachdenken, wie sie zuverlässig in den Automaten integriert werden kann. Auch der mechanische Aufbau erwies sich als herausfordernd: Löten, Verkabeln, 3D-Drucken und das Zusammenfügen der Teile erforderten oft viel Ausprobieren und Geduld. Viele kleinere Probleme, wie das Anpassen von 3D-Teilen, die Optimierung von Motorbewegungen oder das Einstellen der Waage, mussten durch Tests und Nachjustierungen gelöst werden. Trotz allem konnten letztlich sämtliche Probleme behoben werden, und die Komponenten arbeiten nun ohne Schwierigkeiten zusammen. Insgesamt wurde deutlich, dass der Aufwand für ein solches Projekt leicht unterschätzt wird. Es hat deutlich länger gedauert, als ich ursprünglich gedacht hatte, alle Teile zu planen, aufzubauen und zu integrieren. Es war besonders interessant zu sehen, wie viele Überlegungen und Anpassungen notwendig sind, um ein funktionierendes System zu schaffen, etwas, das von aussen oft unsichtbar ist und leicht als „einfach“ bezeichnet wird.

3.2 Verbesserungspotential und Ausblick

Es gibt mehrere Möglichkeiten, die Funktionalität und Benutzerfreundlichkeit zukünftiger Versionen des Automaten zu erweitern. Zuerst könnte man die Optik optimieren: Eine Abdeckung oder ein attraktives Design könnten dem Automaten ein deutlich professionelleres Erscheinungsbild verleihen, sodass er auch optisch wie ein fertiges Produkt aussieht. Dadurch würde der Prototyp an Wert gewinnen und für die Nutzung im Haushalt attraktiver werden. Zusätzlich könnte eine App oder Weboberfläche integriert werden. Mit einer solchen Steuerung wäre es dem Nutzer möglich, Fütterungszeiten und -mengen nach seinen Wünschen einzustellen und zu kontrollieren. Es wäre auch möglich, zusätzliche Funktionen anzubieten, wie das Anzeigen von Bildern oder kurzen Videos der Katzen beim Fressen, die Dokumentation fremder Katzen oder statistische Analysen des Fressverhaltens. Auf diese Weise wäre das System in der Lage, automatisch zu erfassen, welche Katze wie viel gefressen hat, die Fütterung kontinuierlich zu optimieren und den Automaten insgesamt smarter und nützlicher zu gestalten. Auch technisch

gesehen wären Verbesserungen umsetzbar. Man könnte zum Beispiel ein kleines Brett mit einer integrierten Waage vor dem Futternapf anbringen, auf das die Katze beim Fressen steigen muss. Dadurch wäre es möglich, ihr Gewicht zu bestimmen, und die Menge an Futter könnte automatisch und kontinuierlich angepasst werden. Zusammenfassend kann festgestellt werden, dass der Automat in seiner aktuellen Version zwar funktioniert, aber noch viel Entwicklungspotenzial bietet. Insbesondere die Hardware-Software-Kombination und die Benutzerfreundlichkeit könnten in einer künftigen Version wesentlich optimiert werden, um aus dem Prototyp ein ausgereiftes, verlässliches und ansprechendes Produkt zu machen.

Kapitel 4

Fazit

Mir wurde durch das Projekt bewusst, wie viel Arbeit es erfordert, ein physisches Gerät zum Funktionieren zu bringen, und dass der gesamte Prozess leicht unterschätzt wird, wenn man nur auf das Endprodukt schaut. Bei komplexen Projekten sind Planung und schrittweises Vorgehen entscheidend, zuerst Skizzen, dann Prototypen aus Karton und schliesslich der finale Aufbau mit 3D-Druck und Elektronik. Ich konnte viel Technisches lernen, besonders über KI, die Verwendung von Google Colab für mehr Rechenleistung, das Speichern von Projekten auf GitHub und den Umgang mit Motoren, Sensoren und einfacher Elektronik. Das Programmieren und Testen der KI hat mir ebenfalls neue Perspektiven eröffnet. Ich habe zudem erfahren, dass es oft besser ist, weniger Technik zu verwenden: Die Lösung von Problemen auf der Softwareseite gestaltet sich in der Regel einfacher, schneller und kostengünstiger. Aus eigener Erfahrung weiss ich, dass ein umfangreiches Projekt über einen längeren Zeitraum hinweg viel Ausdauer verlangt und dass selbst kleinste Aufgaben einen hohen Zeitaufwand beanspruchen können. Ich bin mit dem Resultat zufrieden. Obwohl der Automat nicht wie ein marktreifes Produkt aussieht, arbeitet er zuverlässig, erfüllt alle Vorgaben und stellt eine Neuheit auf dem Markt dar. Ich würde das Projekt erneut gleich angehen, vielleicht diesmal mit einer etwas konstanteren Arbeitsweise über den gesamten Zeitraum. Alles in allem war es ein sehr lehrreicher Prozess, der mich sowohl auf technischer als auch auf persönlicher Ebene weitergebracht hat.

Literaturverzeichnis

Azmeenasiraj. (2022). Cat Faces Data Set [Zugriff am 15.10.2025]. https://www.kaggle.com/datasets/azmeenasiraj/cat-faces-data-set

Bergmann, D., & Stryker, C. (2025). *Was ist Backpropagation?* [Zugriff am 15.10.2025]. https://www.ibm.com/de-de/think/topics/backpropagation

Bosche. (2025). *Wie funktioniert eine Wägezelle? Erklärung: Aufbau & Funktion* [Zugriff am 17.10.2025]. https://www.bosche.eu/waagen/funktion-waegezelle.html

Crawford, J. (2015). *Cat Dataset*. Verfügbar 16. Oktober 2025 unter https://www.kaggle.com/datasets/crawford/cat-dataset

DataScientest. (2023). *Convolutional Neural Network (CNN): Alles, was Du wissen solltest* [Zugriff am 15.10.2025]. https://datascientest.com/de/convolutional-neural-network-2

Datasolut. (2025). *Neuronale Netzwerke – Einführung* [Zugriff am 15.10.2025]. https://datasolut.com/neuronale-netzwerke-einfuehrung/

Filamentworld. (2025a). *Was ist PETG? Alles über das 3D-Druckmaterial!* [Zugriff am 17.10.2025]. https://filamentworld.de/3d-druck-wissen/was-ist-petg/

Filamentworld. (2025b). *Was ist PLA? Alles über das 3D-Druckmaterial!* [Zugriff am 17.10.2025]. https://filamentworld.de/3d-druck-wissen/was-ist-pla/

Galaxus. (2025). Raspberry Pi 5 8GB Entwicklungsboard Kit [Zugriff am 17.10.2025]. https://www.galaxus.ch/de/s1/product/raspberry-pi-5-8gb-entwicklungsboard-kit-38955607

IBM. (2025a). *Was ist Bilderkennung?* Verfügbar 15. Oktober 2025 unter https://www.ibm.com/de-de/think/topics/image-recognition

IBM. (2025b). *Was ist ein neuronales Netz?* [Zugriff am 15.10.2025]. https://www.ibm.com/de-de/think/topics/neural-networks

IONOS Redaktion. (2024). *Transfer Learning: Vortrainierte Modelle für neue Aufgaben nutzen* [Zugriff am 15.10.2025]. https://www.ionos.de/digitalguide/websites/web-entwicklung/transfer-learning/

Jocher, G., Chaurasia, A., & Qiu, J. (2023). *Ultralytics YOLOv8* (Version 8.0.0) [Zugriff am 15.10.2025]. https://github.com/ultralytics/ultralytics

Joy-IT. (2025a). *NEMA17-03 Schrittmotor* [Zugriff am 17.10.2025]. https://joy-it.net/de/products/NEMA17-03

Joy-IT. (2025b). *TB6600 Schrittmotortreiber* [Zugriff am 17.10.2025]. https://joy-it.net/en/products/SBC-MD-TB6600

Joy-IT. (2025c). *Wägezelle (1 kg) mit Verstärkerboard* [Zugriff am 17.10.2025]. https://joy-it.net/de/products/SEN-HX711-01

Keras. (2025). *MobileNet, MobileNetV2 und MobileNetV3* [Zugriff am 15.10.2025]. https://keras.io/api/applications/mobilenet/

Mebis. (2025). *Bilderkennung* [Zugriff am 17.10.2025]. https://mebis.bycs.de/beitrag/ki-bilderkennung

Pi-Shop.ch. (2025). *Raspberry Pi Camera Cable Standard Mini, 500 mm* [Zugriff am 17.10.2025]. https://www.pi-shop.ch/raspberry-pi-camera-cable-standard-mini-500mm

Raspberry Pi Foundation. (2025). *Raspberry Pi Camera Module v2, 8MP* [Zugriff am 17.10.2025]. https://www.raspberrypi.com/products/camera-module-v2/

StudySmarter Redaktion. (2025). *Schrittmotor – Funktionsweise, Steuerung und Einsatzgebiete* [Zugriff am 17.10.2025]. https://www.studysmarter.de/studium/ingenieurwissenschaften/elektrotechnik/schrittmotor/

Ultralytics. (2025a). *Epoche im maschinellen Lernen (ML)* [Zugriff am 15.10.2025]. https://www.ultralytics.com/de/glossary/epoch

Ultralytics. (2025b). *Validierungsdaten: ML-Modell-Tuning* [Zugriff am 15.10.2025]. https://www.ultralytics.com/de/glossary/validation-data

zITBOx Redaktion. (2024). *Google Colab – Programmieren im Unterricht ohne IDE* [Zugriff am 15.10.2025]. https://zitbox.ch/external_blogs/google-colab-programmieren-im-unterricht-ohne-ide/

Verwendung von KI-Unterstützung

Für die sprachliche Überarbeitung der Arbeit sowie zur Verbesserung der Lesbarkeit und Formatierung des Python-Codes wurde ChatGPT (GPT-5, OpenAI) eingesetzt. ChatGPT diente ausschliesslich zur stilistischen Optimierung, zur Erklärung von Codeabschnitten und zur Strukturierung einzelner Textstellen. Die inhaltlichen Entscheidungen, Interpretationen und die Programmierung stammen vollständig von dem Autor selbst.

Kantonsschule Uetikon am See

Redlichkeitserklärung

Name Gerster Vorname Lukas Klasse 6a

Titel der Arbeit Entwurf und Prototyping eines intelligenten Katzenfutterautomaten

Hiermit erkläre ich, dass ich die vorliegende Arbeit gemäss dem KUE-Reglement verfasst habe, das heisst im Besonderen:

- Ich habe diese Arbeit selbständig verfasst.
- Alle Hilfsmittel, die ich verwendet habe, sind angegeben.
- Alle wörtlichen und sinngemässen Übernahmen aus anderen Werken sind als solche gekennzeichnet.
- Personen, die einen wesentlichen Beitrag zu dieser Arbeit geleistet haben (Betreuer/-in ausgenommen), habe ich ebenfalls erwähnt.

Datum

17.10.2025

Unterschrift

Anhang

Python-Code Katzenfutterautomat

videos_to_pictures.py

```python
"""
Video to Images Converter
-------------------------
This script extracts frames from a video at a fixed interval, rotates them 90   clockwise,
crops them to a square centered on the image, and saves them as individual image files.
"""

import os
import cv2



# ------------------------------
# Configuration
# ------------------------------

video_path = ("C:/Users/Lukas/OneDrive - Kantonsschule Uetikon am See/"
    "Schule/Maturaarbeit/Videos_to_pictures/bruno.mp4")
output_folder = (
    "C:/Users/Lukas/OneDrive - Kantonsschule Uetikon am See/"
    "Schule/Maturaarbeit/Bruno"
)

# Ensure output folder exists
os.makedirs(output_folder, exist_ok=True)

# Frame extraction settings
frame_interval_sec = 0.2  # Extract one frame every 0.2 seconds
first_image_number = 1
first_image_name = f"bruno_{first_image_number}"



# ------------------------------
# Open video
# ------------------------------

cap = cv2.VideoCapture(video_path)
fps = cap.get(cv2.CAP_PROP_FPS)
frame_interval = max(1, int(fps * frame_interval_sec))

frame_count = 0
image_count = 0



# ------------------------------
# Process video frames
# ------------------------------

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break  # End of video

    if frame_count % frame_interval == 0:
```

```
        # Rotate frame 90   clockwise
        frame = cv2.rotate(frame, cv2.ROTATE_90_CLOCKWISE)

        # Crop frame to square (centered)
        height, width, _ = frame.shape
        min_dim = min(height, width)
        start_x = (width - min_dim) // 2
        start_y = (height - min_dim) // 2
        frame = frame[start_y:start_y + min_dim, start_x:start_x + min_dim]

        # Save frame
        if image_count == 0:
            frame_filename = f"{first_image_name}.jpg"
        else:
            frame_filename = f"bruno_{first_image_number + image_count}.jpg"

        cv2.imwrite(os.path.join(output_folder, frame_filename), frame)
        image_count += 1

    frame_count += 1



# ------------------------------
# Release resources
# ------------------------------

cap.release()
cv2.destroyAllWindows()

print(f"Total images saved: {image_count}")
```

make_pictures.py

```
"""
Raspberry Pi Cat Image Capture
------------------------------
This program captures images from the Raspberry Pi camera and saves them
to a folder corresponding to the selected cat. Change the CAT_NAME variable
to either 'bruno' or 'flaekli' to capture images for the respective cat.
"""

# Standard library
import os
import time

# Third-party libraries
import cv2
from picamera2 import Picamera2



# ---------------------------------------------------------------------
#   Configuration
# ---------------------------------------------------------------------

CAT_NAME = "bruno"
DATA_FOLDER = "data"
CAPTURE_INTERVAL = 0.5  # Time in seconds between captures
IMG_WIDTH, IMG_HEIGHT = 640, 480

# Ensure output folder exists
OUTPUT_FOLDER = os.path.join(DATA_FOLDER, CAT_NAME)
os.makedirs(OUTPUT_FOLDER, exist_ok=True)



# ---------------------------------------------------------------------
#   Functions
# ---------------------------------------------------------------------

def initialize_camera() -> Picamera2:
    """
```

```
    Initialize the Raspberry Pi camera.

    Returns:
        Picamera2: Configured and started camera object.
    """
    print("Initializing camera...")
    picam = Picamera2()
    config = picam.create_preview_configuration(
        main={"size": (IMG_WIDTH, IMG_HEIGHT)}
    )
    picam.configure(config)
    picam.start()
    print("Camera started, warming up for 2 seconds...")
    time.sleep(2)
    return picam


def capture_images(picam: Picamera2, folder: str, interval: float = 0.5) -> None:
    """
    Capture images from the camera at a fixed interval and save them to disk.

    Args:
        picam (Picamera2): Camera object.
        folder (str): Folder to save captured images.
        interval (float): Time between captures in seconds.
    """
    index = len(os.listdir(folder))  # Start counting from existing files

    try:
        while True:
            img = picam.capture_array()
            if img is None:
                print("No image captured, retrying...")
                time.sleep(0.1)
                continue

            # Convert RGB to BGR for OpenCV
            img_bgr = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
            filename = os.path.join(folder, f"{CAT_NAME}_{index}.jpg")
            cv2.imwrite(filename, img_bgr)
            print(f"Saved image: {filename}")

            index += 1
            time.sleep(interval)

    except KeyboardInterrupt:
        print("Image capture stopped by user.")
    finally:
        picam.stop()
        cv2.destroyAllWindows()



# ---------------------------------------------------------------------
#   Main
# ---------------------------------------------------------------------

def main() -> None:
    """
    Main function to initialize the camera and start capturing images.
    """
    picam = initialize_camera()
    print(
        f"Starting image capture for '{CAT_NAME}' in folder '{OUTPUT_FOLDER}'..."
    )
    capture_images(picam, OUTPUT_FOLDER, CAPTURE_INTERVAL)


if __name__ == "__main__":
    main()
```

preprocess_images.py

```
"""
Cat Image Preprocessing Pipeline
--------------------------------
This program detects cats in images using YOLO, crops the detected
cats, saves them in a preprocessed folder, and balances the number
of images between the target cat and other cats.
"""

# Install YOLO
!pip install ultralytics

# Standard library
import os
import random
import shutil

# Third-party libraries
import cv2
from google.colab import drive
from ultralytics import YOLO



# ---------------------------------------------------------------------
#   Constants
# ---------------------------------------------------------------------

CAT = "bruno"
MODEL = YOLO("yolov8n.pt")
DATASET_NAME = f"dataset_{CAT[:3]}_other"
INPUT_FOLDER = f"/content/drive/MyDrive/datasets/{DATASET_NAME}/images"
OUTPUT_FOLDER = f"/content/drive/MyDrive/datasets/{DATASET_NAME}/preprocessed_images"
CLASS_FOLDERS = [CAT, "other_cats"]

# ---------------------------------------------------------------------
#   Functions
# ---------------------------------------------------------------------

def mount_drive():
    """Mount Google Drive in Colab to access the image dataset."""
    drive.mount('/content/drive')


def create_output_dirs():
    """Create output directories for each class (cat and other cats)."""
    for category in CLASS_FOLDERS:
        os.makedirs(os.path.join(OUTPUT_FOLDER, category), exist_ok=True)


def process_images():
    """
    Detect cats in images using YOLO, crop detected cats, and save
    them in the preprocessed folder. Remove images without detected cats.
    """
    for class_name in CLASS_FOLDERS:
        class_path = os.path.join(INPUT_FOLDER, class_name)
        start_index = 0

        for filename in os.listdir(class_path):
            if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.tiff', '.bmp', '.gif')):
                img_path = os.path.join(class_path, filename)
                img = cv2.imread(img_path)

                # Detect cats using YOLO
                results = MODEL.predict(source=img_path, conf=0.47, classes=[15])

                if results[0].boxes:
                    boxes = results[0].boxes.xyxy.tolist()
                    for box in boxes:
                        x_min, y_min, x_max, y_max = map(int, box)
                        cropped_img = img[y_min:y_max, x_min:x_max]

                        # Save cropped image
                        cropped_img_path = os.path.join(
```

```
                            OUTPUT_FOLDER, class_name, f"{class_name}_{start_index}.jpg"
                        )
                        cv2.imwrite(cropped_img_path, cropped_img)
                        start_index += 1
                else:
                    # Remove images without detected cats
                    os.remove(img_path)


def balance_dataset():
    """
    Balance the number of images between CAT and other_cats by removing
    excess files or duplicating images to match the smaller class.
    """
    cat_files = [
        f for f in os.listdir(os.path.join(OUTPUT_FOLDER, CAT))
        if f.endswith((".jpg", ".png"))
    ]
    other_cats_files = [
        f for f in os.listdir(os.path.join(OUTPUT_FOLDER, "other_cats"))
        if f.endswith((".jpg", ".png"))
    ]

    cat_count = len(cat_files)
    other_cats_count = len(other_cats_files)

    if cat_count != other_cats_count:
        target_count = min(cat_count, other_cats_count)

        # Remove excess files from the larger class
        if cat_count > target_count:
            for file in random.sample(cat_files, cat_count - target_count):
                os.remove(os.path.join(OUTPUT_FOLDER, CAT, file))
        elif other_cats_count > target_count:
            for file in random.sample(other_cats_files, other_cats_count - target_count):
                os.remove(os.path.join(OUTPUT_FOLDER, "other_cats", file))

        # Refresh lists after removal
        cat_files = [f for f in os.listdir(os.path.join(OUTPUT_FOLDER, CAT)) if f.endswith((".
            jpg", ".png"))]
        other_cats_files = [f for f in os.listdir(os.path.join(OUTPUT_FOLDER, "other_cats"))
            if f.endswith((".jpg", ".png"))]

        # Duplicate files to balance smaller class
        if cat_count < other_cats_count:
            for _ in range(target_count - cat_count):
                file_to_copy = random.choice(other_cats_files)
                shutil.copy(
                    os.path.join(OUTPUT_FOLDER, "other_cats", file_to_copy),
                    os.path.join(OUTPUT_FOLDER, CAT, file_to_copy)
                )
        elif other_cats_count < cat_count:
            for _ in range(target_count - other_cats_count):
                file_to_copy = random.choice(cat_files)
                shutil.copy(
                    os.path.join(OUTPUT_FOLDER, CAT, file_to_copy),
                    os.path.join(OUTPUT_FOLDER, "other_cats", file_to_copy)
                )


# ---------------------------------------------------------------------
#   Main
# ---------------------------------------------------------------------

def main():
    """Run the full image preprocessing pipeline."""
    mount_drive()
    create_output_dirs()
    process_images()
    balance_dataset()
    print("Images have been processed, balanced, and saved.")


if __name__ == "__main__":
    main()
```

main.py

```python
"""
Automatic Cat Feeder
--------------------
This program controls a smart feeding station with camera recognition
and weight sensor. It identifies cats via AI (YOLO + Keras),
opens the bowl for the correct cat, and monitors the weight to
control the ration.
"""

# Standard library
import json
import os
import time

# Third-party libraries
import cv2
import numpy as np
from PIL import Image
import RPi.GPIO as GPIO
import tensorflow as tf
from tensorflow import keras
from ultralytics import YOLO
from picamera2 import Picamera2

# Local / custom libraries
from JoyIT_hx711py import HX711


# ---------------------------------------------------------------------
#   Hardware Initialization (Scale, Motor, GPIO)
# ---------------------------------------------------------------------

# HX711 scale initialization
HX = HX711(5, 6)
offset = float(input("Enter scale offset: "))
scale = float(input("Enter scale factor: "))
HX.set_offset(offset)
HX.set_scale(scale)

# Motor pins
DIR_PIN = 13
PUL_PIN = 11
ENA_PIN = 15

GPIO.setmode(GPIO.BOARD)
GPIO.setup(DIR_PIN, GPIO.OUT)
GPIO.setup(PUL_PIN, GPIO.OUT)


# ---------------------------------------------------------------------
#   Classes
# ---------------------------------------------------------------------

class FoodBowl:
    """Represents a single food bowl."""

    def __init__(self, state: str, cat: str, weight: float):
        self.state = state
        self.cat = cat
        self.weight = weight



# ---------------------------------------------------------------------
#   JSON Handling (Cat and Ration Data)
# ---------------------------------------------------------------------

def load_cats(file_path: str) -> dict:
    """Load cat information from a JSON file."""
    with open(file_path, "r") as file:
        return json.load(file)


def save_cats(cats_data: dict, file_path: str = "cats.json") -> None:
```

```
    """Save updated cat information to a JSON file."""
    with open(file_path, "w") as file:
        json.dump(cats_data, file, indent=2)



# ---------------------------------------------------------------------
#   Hardware Functions
# ---------------------------------------------------------------------

def weigh_bowl() -> float:
    """Measure the bowl weight via the HX711 sensor."""
    HX.power_up()
    value = HX.get_grams()
    HX.power_down()
    return value


def step_motor(steps: int, direction: bool, delay: float = 0.001) -> None:
    """Control the stepper motor for a specific number of steps."""
    GPIO.output(DIR_PIN, direction)
    for _ in range(steps):
        GPIO.output(PUL_PIN, True)
        time.sleep(delay)
        GPIO.output(PUL_PIN, False)
        time.sleep(delay)


def open_bowl(cat_name: str, bowl: FoodBowl) -> None:
    """Open the bowl for the recognized cat."""
    step_motor(900, direction=True)
    bowl.state = "open"
    bowl.cat = cat_name


def close_bowl(weight: float, cat_name: str, bowl: FoodBowl, cats_data: dict) -> None:
    """Close the bowl and update remaining ration."""
    step_motor(900, direction=False)
    cats_data[cat_name]['ration_left'] = cats_data.get(cat_name, {}).get("ration_left")-(bowl.
        weight-weight)
    save_cats(cats_data)
    bowl.weight=weight
    bowl.state = "closed"
    bowl.cat = ""


def fill_up_bowl(bowl: FoodBowl, weight: float, cats_data: dict) -> None:
    """Automatically refill the bowl if the weight is too low."""
    if bowl.weight < max(cat_info["ration_total"] for cat_info in cats_data.values()):
        step_motor(287, direction=False)
        while int(weigh_bowl())<max(cat_info["ration_total"] for cat_info in cats_data.values(
            )):
            time.sleep(0.001)
        step_motor(287, direction=True)
    bowl.weight = int(weigh_bowl())



# ---------------------------------------------------------------------
#   AI Detection
# ---------------------------------------------------------------------

def detect_cat(
    picam: Picamera2,
    yolo_model: YOLO,
    keras_models: dict,
    cat_names: list,
    img_size: tuple[int, int]
) -> str:
    """
    Detect if a cat is in front of the camera and identify it.

    Returns:
        cat_name (str): Name of the recognized cat or empty string.
    """
    img_array = picam.capture_array()
```

```
    img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB)

    try:
        results = yolo_model.predict(source=img_array, conf=0.3, classes=[15])
    except Exception as err:
        print(f"YOLO detection error: {err}")
        return ""

    if not results or not results[0].boxes or len(results[0].boxes) == 0:
        return ""

    boxes = results[0].boxes.xyxy.tolist()
    for box in boxes:
        x_min, y_min, x_max, y_max = map(int, box)
        cropped = img_array[y_min:y_max, x_min:x_max]
        cropped = cv2.cvtColor(cropped, cv2.COLOR_BGR2RGB)
        image = Image.fromarray(cropped.astype("uint8"))
        resized = image.resize(img_size)

        img_array_resized = keras.utils.img_to_array(resized) / 255.0
        img_array_resized = np.expand_dims(img_array_resized, axis=0)

        # Evaluate with individual cat models
        scores = []
        for cat in cat_names:
            try:
                score = float(keras_models[cat].predict(img_array_resized)[0])
                if 100 * (1 - score) > 25:
                    scores.append((cat, score))
            except Exception as e:
                print(f"Error with model '{cat}': {e}")

        if not scores:
            return ""
        return max(scores, key=lambda x: x[1])[0]

    return ""



# ---------------------------------------------------------------------
#   Main Loop
# ---------------------------------------------------------------------

def main_loop(
    picam: Picamera2,
    yolo_model: YOLO,
    keras_models: dict,
    cats_data: dict,
    bowl: FoodBowl,
    img_size: tuple[int, int]
) -> None:
    """Infinite loop for the cat feeder behavior."""
    wrong_detection_count = 0
    last_modified = os.path.getmtime("cats.json")

    while True:
        # Check if cat configuration changed
        current_modified = os.path.getmtime("cats.json")
        if current_modified != last_modified:
            last_modified = current_modified
            cats_data = load_cats("cats.json")

        cat_names = list(cats_data.keys())

        # Detect cat and measure weight
        cat = detect_cat(picam, yolo_model, keras_models, cat_names, img_size)
        weight = weigh_bowl()

        # Bowl open/close/refill logic
        if cat in cat_names and bowl.state == "closed" and int(cats_data[cat]["ration_left"])
            > 0:
            open_bowl(cat, bowl)

        if bowl.state == "open":
            if cat != bowl.cat:
```

```
                wrong_detection_count += 1
                print(f"Wrong cat detected ({wrong_detection_count})")
            else:
                wrong_detection_count = 0

            if (bowl.weight - weight > cats_data.get(bowl.cat, {}).get("ration_left")) or (
                wrong_detection_count >= 5):
                close_bowl(weight, bowl.cat, bowl, cats_data)
                wrong_detection_count = 0
                fill_up_bowl(bowl, weight, cats_data)

        print(f"Current bowl state: {bowl.state}")
        time.sleep(0.1)



# ---------------------------------------------------------------------
#   Program Start
# ---------------------------------------------------------------------

def main():
    """Initialize camera, models, and start the feeder."""
    print("Starting automatic cat feeder...")

    # Load models
    MODELS_DIR = "models"
    keras_models = {}

    for filename in os.listdir(MODELS_DIR):
        if filename.endswith(".keras"):
            cat_name = filename.replace("modelv1_", "").replace(".keras", "")
            model_path = os.path.join(MODELS_DIR, filename)
            keras_models[cat_name] = tf.keras.models.load_model(model_path)
            print(f"Loaded model: {cat_name}")

    yolo_model = YOLO("yolov8n.pt")

    # Start camera
    picam = Picamera2()
    picam.start()
    print("Camera initialized")

    # Load JSON data
    cats_data = load_cats("cats.json")

    # Fill the bowl initially
    step_motor(287, direction=False)
    while int(weigh_bowl()) < max(cat_info["ration_total"] for cat_info in cats_data.values())
        :
        time.sleep(0.001)
    step_motor(287, direction=True)

    # Create bowl object
    bowl = FoodBowl("closed", "", weigh_bowl())

    # Start main loop
    try:
        main_loop(picam, yolo_model, keras_models, cats_data, bowl, (180, 180))
    finally:
        GPIO.cleanup()


if __name__ == "__main__":
    main()
```

train_model.py

```python
"""
Cat Classifier Training Script
------------------------------
This script mounts Google Drive in Colab, loads a dataset of cat images,
trains a binary image classifier for a specific cat against other cats
using MobileNetV2 as base model, and saves the trained model. It also
displays sample images before training and evaluates the trained model.
"""

# Standard library
import os

# Third-party libraries
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from keras import layers
from google.colab import drive



# ---------------------------------------------------------------------
#   Constants
# ---------------------------------------------------------------------

IMG_SIZE = (180, 180)
IMG_SIZE_ = 180
BATCH_SIZE = 16
CAT = "bruno"
DATASET_NAME = f"dataset_{CAT[:3]}_other"
VERSION = "4"

CLASS_0_DIR = f"/content/drive/MyDrive/datasets/{DATASET_NAME}/preprocessed_images/{CAT}"
CLASS_1_DIR = f"/content/drive/MyDrive/datasets/{DATASET_NAME}/preprocessed_images/other_cats"



# ---------------------------------------------------------------------
#   Functions
# ---------------------------------------------------------------------

def mount_google_drive():
    """Mount Google Drive in Colab to access dataset files."""
    drive.mount("/content/drive")


def load_image_paths():
    """
    Load image file paths and create a DataFrame with labels.

    Returns:
        pd.DataFrame: DataFrame containing image filenames and class labels.
    """
    class_0_images = [
        os.path.join(CLASS_0_DIR, fname)
        for fname in os.listdir(CLASS_0_DIR)
    ]
    class_1_images = [
        os.path.join(CLASS_1_DIR, fname)
        for fname in os.listdir(CLASS_1_DIR)
    ]

    df = pd.DataFrame({
        "filename": class_0_images + class_1_images,
        "class": [0] * len(class_0_images) + [1] * len(class_1_images)
    })
    df['class'] = df['class'].astype(str)
    return df


def create_generators(df):
    """
    Create training and validation generators using ImageDataGenerator.
```

```
    Applies rescaling and data augmentation for training.

    Args:
        df (pd.DataFrame): DataFrame containing filenames and labels.

    Returns:
        tuple: (train_generator, validation_generator)
    """
    datagen = keras.preprocessing.image.ImageDataGenerator(
        rescale=1.0 / 255,
        validation_split=0.2,
        rotation_range=40,
        width_shift_range=0.26,
        height_shift_range=0.26,
        shear_range=0.17,
        zoom_range=0.26,
        horizontal_flip=True,
        fill_mode="nearest"
    )

    train_gen = datagen.flow_from_dataframe(
        dataframe=df,
        x_col="filename",
        y_col="class",
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode="binary",
        subset="training",
        shuffle=True,
        seed=42
    )

    val_gen = datagen.flow_from_dataframe(
        dataframe=df,
        x_col="filename",
        y_col="class",
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode="binary",
        subset="validation",
        shuffle=True,
        seed=42
    )

    return train_gen, val_gen


def display_sample_images(generator):
    """
    Display a batch of images from the generator with labels.

    Args:
        generator (keras.preprocessing.image.DataFrameIterator):
            Image data generator.
    """
    images, labels = next(generator)
    plt.figure(figsize=(10, 10))
    for i in range(min(9, BATCH_SIZE)):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])
        plt.title(f"Label: {int(labels[i])}")
        plt.axis("off")
    plt.show()


def build_model():
    """
    Build and compile a MobileNetV2-based binary classifier.

    Returns:
        keras.Model: Compiled Keras model.
    """
    base_model = keras.applications.MobileNetV2(
        input_shape=(IMG_SIZE_, IMG_SIZE_, 3),
        include_top=False,
        weights="imagenet"
```

```
    )
    base_model.trainable = False

    x = layers.GlobalAveragePooling2D()(base_model.output)
    x = layers.Dense(128, activation="relu")(x)
    output = layers.Dense(1, activation="sigmoid")(x)

    model = keras.Model(inputs=base_model.input, outputs=output)
    model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
    return model


def train_model(model, train_gen, val_gen, cat_name, version):
    """
    Train a binary image classifier using the training generator and
    evaluate it on the validation generator. Saves the trained model
    to Google Drive.

    Args:
        model (keras.Model): Compiled Keras model.
        train_gen (keras.preprocessing.image.DataFrameIterator): Training data generator.
        val_gen (keras.preprocessing.image.DataFrameIterator): Validation data generator.
        cat_name (str): Name of the target cat.
        version (str): Version string for the saved model filename.

    Returns:
        str: Path to the saved model file.
    """
    model.fit(train_gen, validation_data=val_gen, epochs=5)

    model_path = f"/content/drive/MyDrive/repos/keras-distinguish-own-cat-from-others-model/
        models/model{version}_{cat_name}.keras"
    model.save(model_path, save_format="keras")
    return model_path


def evaluate_model(model_path, val_gen):
    """
    Load a saved model and evaluate its performance on validation data.

    Args:
        model_path (str): Path to the saved Keras model.
        val_gen (keras.preprocessing.image.DataFrameIterator): Validation data generator.
    """
    model = tf.keras.models.load_model(model_path)
    loss, accuracy = model.evaluate(val_gen)
    print(f"Test Loss: {loss}")
    print(f"Test Accuracy: {accuracy}")



# ----------------------------------------------------------------------
#   Main
# ----------------------------------------------------------------------

def main():
    """Main function to run the full Google Colab training pipeline."""
    mount_google_drive()

    df = load_image_paths()
    train_gen, val_gen = create_generators(df)

    print(f"Class names: {CAT} = 0, other_cats = 1")
    display_sample_images(train_gen)

    model = build_model()
    model_path = train_model(model, train_gen, val_gen, CAT, VERSION)
    evaluate_model(model_path, val_gen)


if __name__ == "__main__":
    main()
```

update_cats_data.py

```python
"""
Automatic Ration Reset
----------------------
This program resets the remaining food rations ("ration_left")
for all cats stored in a JSON file. It runs continuously and
resets all rations at fixed time intervals (currently every 12 hours).
"""

# Standard library
import json
import time
from datetime import datetime



# ---------------------------------------------------------------------
#   Configuration
# ---------------------------------------------------------------------

# Path to the JSON file storing cat data
FILE_PATH = "cats.json"



# ---------------------------------------------------------------------
#   Functions
# ---------------------------------------------------------------------

def reset_rations() -> None:
    """
    Reset the remaining ration ('ration_left') for each cat to its total ration ('ration_total').
    """
    # Read JSON file
    with open(FILE_PATH, "r", encoding="utf-8") as file:
        cats_data = json.load(file)

    # Update each  c a t s  remaining ration
    for cat, info in cats_data.items():
        info["ration_left"] = info["ration_total"]

    # Save updated data
    with open(FILE_PATH, "w", encoding="utf-8") as file:
        json.dump(cats_data, file, indent=2, ensure_ascii=False)

    # Print timestamped log
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    print(f"[{timestamp}] Rations have been reset.")


def main() -> None:
    """
    Run the reset function in an infinite loop with a fixed delay.
    """
    while True:
        reset_rations()
        # Wait 12 hours (12 * 60 * 60 seconds)
        time.sleep(12 * 60 * 60)



# ---------------------------------------------------------------------
#   Entry Point
# ---------------------------------------------------------------------

if __name__ == "__main__":
    main()
```

cats.json

```
{
  "flaekli": {
    "ration_total": 150,
    "ration_left": 100.0,
    "accuracy": 94
  },
  "bruno": {
    "ration_total": 200,
    "ration_left": -1.0,
    "accuracy": 92
  }
}
```

Hinweis: Der vollständige Code, die Modelle und die für das Training verwendeten Bilder sind ebenfalls auf GitHub verfügbar. https://github.com/luegl/automatic-feeding-automat-final