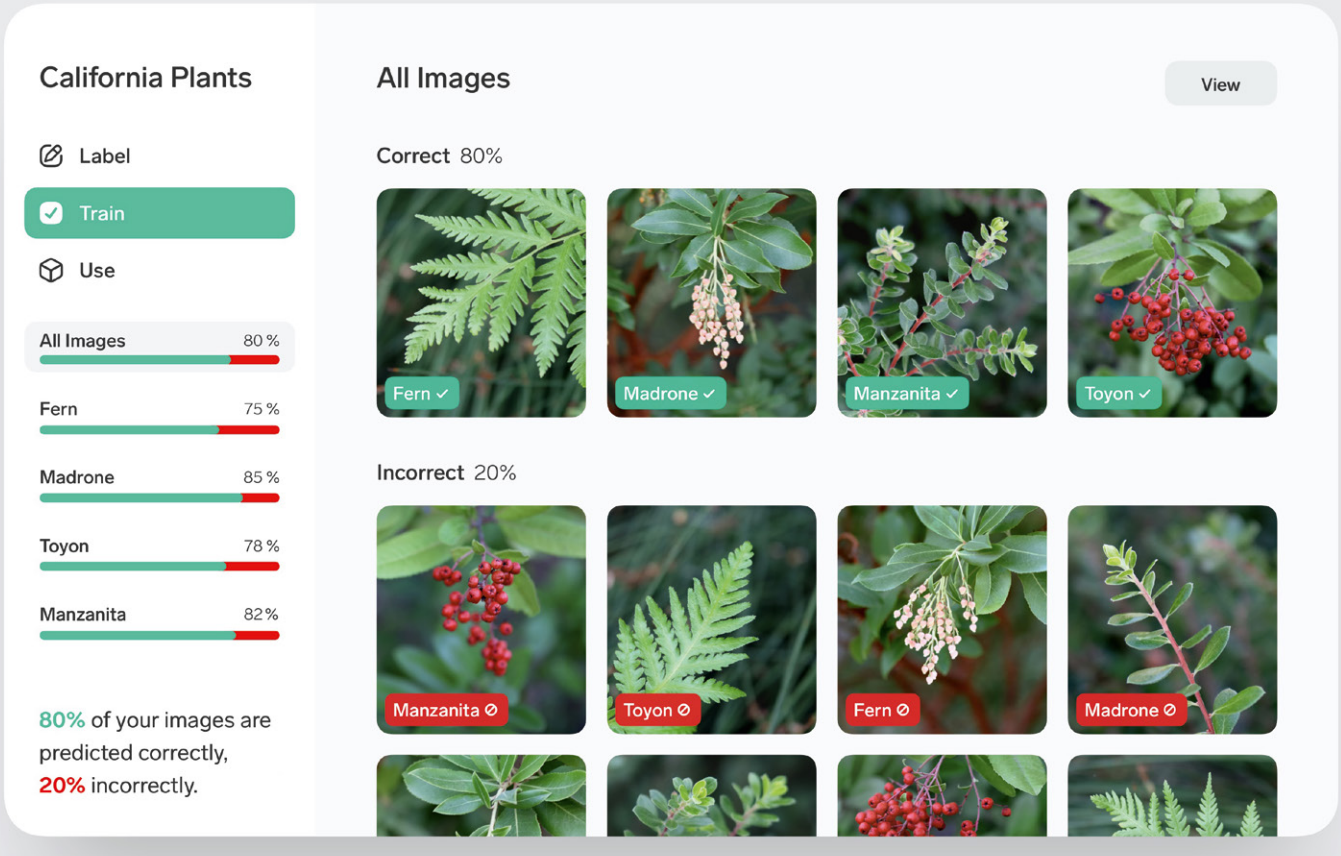
# KI-Praxis VII

Teil 7

## Trainieren selbst erstellter KI-Modelle

**In den bisherigen Beiträgen dieser Reihe wurden überwiegend fertig verfügbare Modelle für die einzelnen Aufgaben eingesetzt. Das ist das übliche Vorgehen, wenn man Machine-Learning-Projekte auf dem Raspberry Pi umsetzen will. Das Internet bietet eine Vielfalt an vortrainierten Neuronalen Netzen und Modellen. Damit kann eine große Anzahl von Anwendungen abgedeckt werden. Mit diesen fertigen Modellen konnten in den letzten Beiträgen Anwendungsbeispiele zur Ziffernerkennung, zur Objekterfassung oder zur Gesichtserkennung in der Praxis erprobt werden. Wenn man jedoch speziellere Aufgaben zu lösen hat, kommt man um das Trainieren eigener Netzwerksysteme nicht herum.**




### Lernfähige Systeme

Viele vorgefertigte Modelle können nur sehr spezielle Probleme lösen. Sie können Ziffern erkennen oder feststellen, dass sich ein Gesicht im Blickfeld einer Kamera befindet. Einige Verfahren besitzen dagegen bereits eine bestimmte Lernfähigkeit. So konnten bei der Gesichtsidentifizierung nicht nur vorgegebene Personen erkannt werden, vielmehr war es möglich, nach einem entsprechenden Training, auch neue, selbst festgelegte Gesichter als „Schlüssel" zu verwenden.

Etwas anders liegen die Dinge, wenn man Neuronale Netze vollständig in Eigenregie trainieren möchte. Dies ist meist mit erheblichem Aufwand verbunden. Die Erstellung des Erkennungsprogramms in Python ist dabei meist das geringste Problem.

Die größte Aufgabe ist das Sammeln umfangreichen „Lernmaterials" für das Neuronale Netz. Bei den im Internet verfügbaren fertigen Modellen wird häufig auf reichlich vorhandenes Datenmaterial aus den allgemeinen Websites oder aber auch aus sozialen Netzwerken zurückgegriffen. Hier haben die Nutzer den Datensammlern oftmals weitgehende Nutzungsrechte eingeräumt, ohne dass ihnen dies wirklich bewusst ist.

Will man sein eigenes spezielles Modell trainieren, müssen die Daten dagegen selbst gesammelt werden. In diesem Beitrag wird gezeigt, wie man dabei vorgehen kann.

## Erstellung von Modellen für den Raspberry Pi

In gewisser Weise ist die Erstellung und das Trainieren eigener Modelle die Königsdisziplin bei der Arbeit mit Machine-Learning-Anwendungen. Man gewinnt so die maximale Freiheit und kann das selbst erstellte Netz auf praktisch alle denkbaren Aufgaben des Maschinellen Lernens anwenden. Die Anwendungen sind vielfältig:

- Erkennung von Tieren und Tierarten
- Sortieren von Waren oder Objekten wie z. B. Werk- oder Spielzeugen
- Abfalltrennung oder Pfandflaschenerkennung etc.
- Erkennung von Fahrzeugen und Fahrzeugtypen
- Sortieren elektronischer Komponenten
- Kategorisierung von Obst- und Gemüsesorten
- Erkennung von Handgesten
- Erfassung emotionaler Reaktionen wie Lächeln, Ärger, Wut, Trauer
- Erkennen, ob eine Gesichtsmaske getragen wird oder nicht
- Gefahrenerkennung

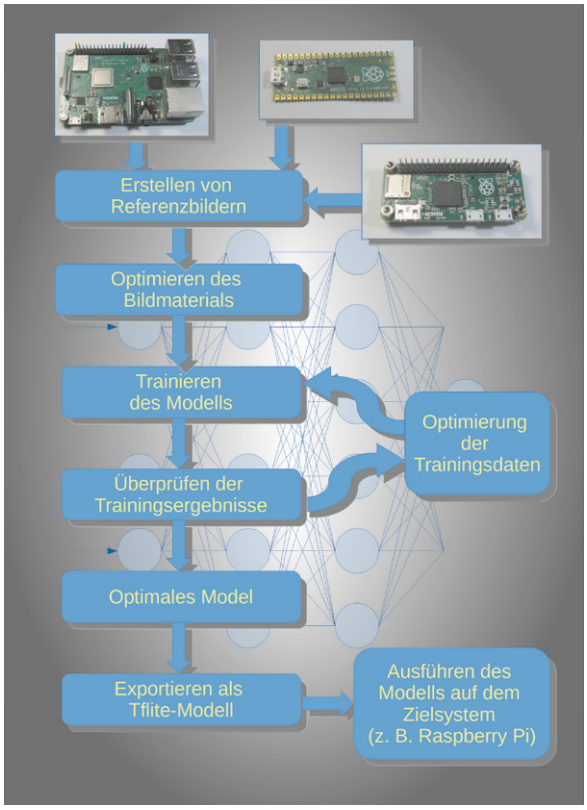



*Bild 1: Prinzipieller Ablauf für das Training mit eigenen Daten*

Insbesondere das Beispiel der Erkennung von Gesichtsmasken zeigt, dass KI-Methoden sehr flexibel anwendbar sind. Lernfähige Systeme können damit auch ohne komplexe Neuprogrammierung schnell an aktuelle Aufgaben angepasst werden.

## Netzwerke trainieren mit Lobe

Prinzipiell wäre es möglich, ein Jupyter-Notebook zu erstellen, das ein Training mit eigenen Bilddaten ermöglicht. Man sollte sich jedoch im Klaren darüber sein, dass es sich bei einem solchen Projekt um eine höchst anspruchsvolle Aufgabe handelt. Darüber hinaus sind hierfür umfangreiche Fachkenntnisse und eine hohes Maß an Erfahrung erforderlich.

Der Arbeitsablauf für das grundlegende Trainieren eigener Neuronaler Netze ist relativ komplex. Zudem sind die entsprechenden Verfahren aufgrund der schnellen Entwicklungen auf diesem Gebiet auch raschen Änderungen unterworfen. Damit kann momentan nicht immer sichergestellt werden, dass die Versionen und Varianten der verschiedenen Programme und Tools problemlos zusammenspielen.

Die Grafik in Bild 1 zeigt am Beispiel einer Klassifizierung für elektronische Komponenten einen Überblick über die einzelnen Schritte, die für den Aufbau und das Training eines Netzes mit eigenen (Bild-)Daten erforderlich sind.

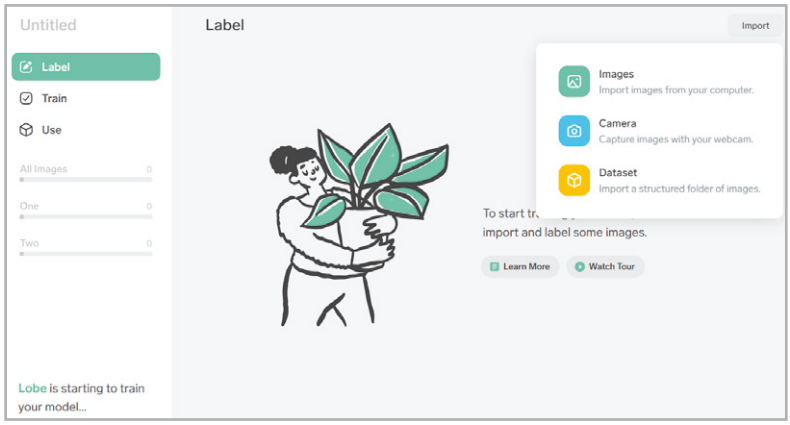



*Bild 2: Lobe nach dem Start*

Auch das Exportieren des Datenmaterials ist prinzipiell möglich. So bietet Python bzw. die Jupyter-Umgebung verschiedene Methoden an, um die Netzwerk-Gewichte im TF-Lite-Format zu exportieren. Allerdings sind diese Methoden relativ aufwendig und fehlerträchtig.

Seit Kurzem steht jedoch ein PC-basiertes Programm namens Lobe zur Verfügung, welches das Trainieren eigener Netzwerke wesentlich vereinfacht. Dieses kann kostenlos aus dem Internet geladen werden [1]. Lobe stellt alle erforderlichen Tools zur Verfügung, um eigene Ideen zum Maschinellen Lernen zum Leben zu erwecken. Einige einfache Bilddaten genügen, um ein benutzerdefiniertes maschinelles Lernmodell zu trainieren. Zudem kann dieses dann bequem per Mausklick als Tflite-Modell exportiert werden. Lobe reduziert damit den Prozess des Maschinellen Lernens auf fünf einfache Schritte (s. auch Bild 1):

– Sammeln und Beschriften der Daten
– Interaktives Trainieren des Modells
– Optimierung
– Exportieren des Modells
– Anwendung auf der Zielhardware

Nach der Installation von Lobe steht eine interaktive Oberfläche zur Verfügung (Bild 2).

Zunächst muss in Lobe ein neues Projekt gestartet werden. Hierzu wird in der Menüauswahl „New Project" ausgewählt. Dem neuen Projekt sollte ein Name zugeordnet werden. Dieser wird im Feld mit der Voreinstellung „Untitled" eingegeben. Im Folgenden wird mit dem Projektnamen „BoardSpecifier" gearbeitet.

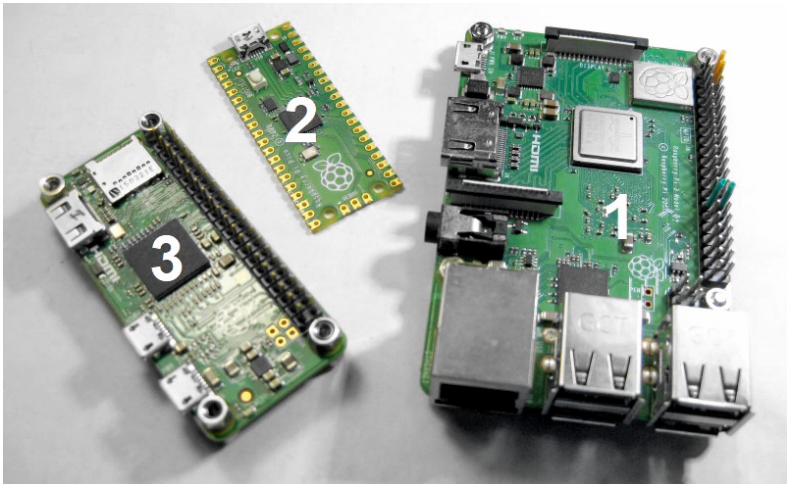



Bild 3: Drei Raspberry-Pi-Varianten

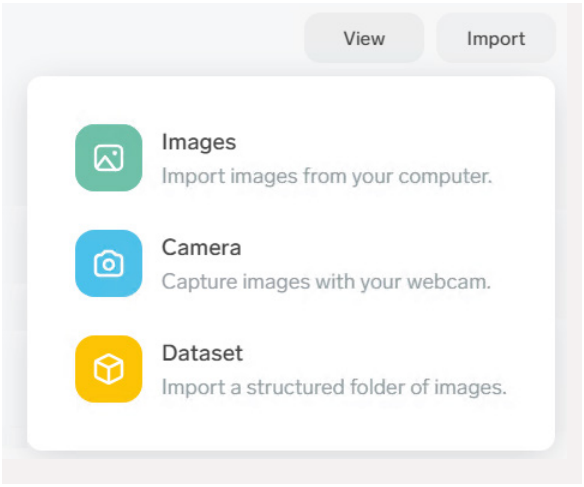



Bild 4: Importieren von Daten

## Raspberry Pi 4, Pi Zero oder Pi Pico?

„Erkenne dich selbst" ist eine der zentralen Forderungen der Philosophie. Was dem Menschen oft schwerfällt, ist für den Raspberry Pi kein Problem. Um dies zu bestätigen, soll im Folgenden ein Neuronales Netzwerk trainiert werden, welches es dem Raspberry Pi erlaubt, sich selbst zu erkennen. Da der Pi inzwischen in mehreren Versionen vorliegt, ist die Frage interessant, ob auch die Varianten korrekt erkannt werden können. In einem ersten Schritt soll die Anzahl der Varianten auf drei beschränkt werden.

Das System wird also darauf trainiert, die folgenden Raspberry-Pi-Boards (Bild 3) zu erkennen:

1. Raspberry Pi Classic
2. Raspberry Pi Pico
3. Raspberry Pi Zero

Um das entsprechende Neuronale Netz zu trainieren, sind wieder Trainingsdaten erforderlich. Wie üblich werden dazu Fotos aufgenommen oder importiert. Dabei ist zu beachten, dass die Aufnahmen von Anfang an den gewünschten Kategorien zugeordnet werden. Die gewählten Kategorie-Bezeichnungen werden später in der Raspberry-Pi-Software benötigt.

Die Fotos können dem in Lobe implementierten Neuronalen Netz über drei verschiedene Wege zugänglich gemacht werden:

1. Über eine Webcam am PC
2. Aufnahme über die Pi-Cam
3. Importieren aus einer Online-Quelle

Jede Methode hat ihre spezifischen Vor- und Nachteile. Allerdings lassen sich die Verfahren auch kombinieren.

## Trainingsdaten mit Lobe erfassen

Die Erfassung von Trainingsdaten mit Lobe kann direkt über eine am PC angeschlossene Webcam erfolgen. Hierzu wird über „Import" (s. Bild 4) ein Untermenü geöffnet. Dieses bietet die Auswahlmöglichkeiten:

**Images:** Importieren von Bildern
**Camera:** Aufnahme von Fotos mit einer Webcam
**Dataset:** Importieren bereits vorbereiteter Verzeichnisstrukturen

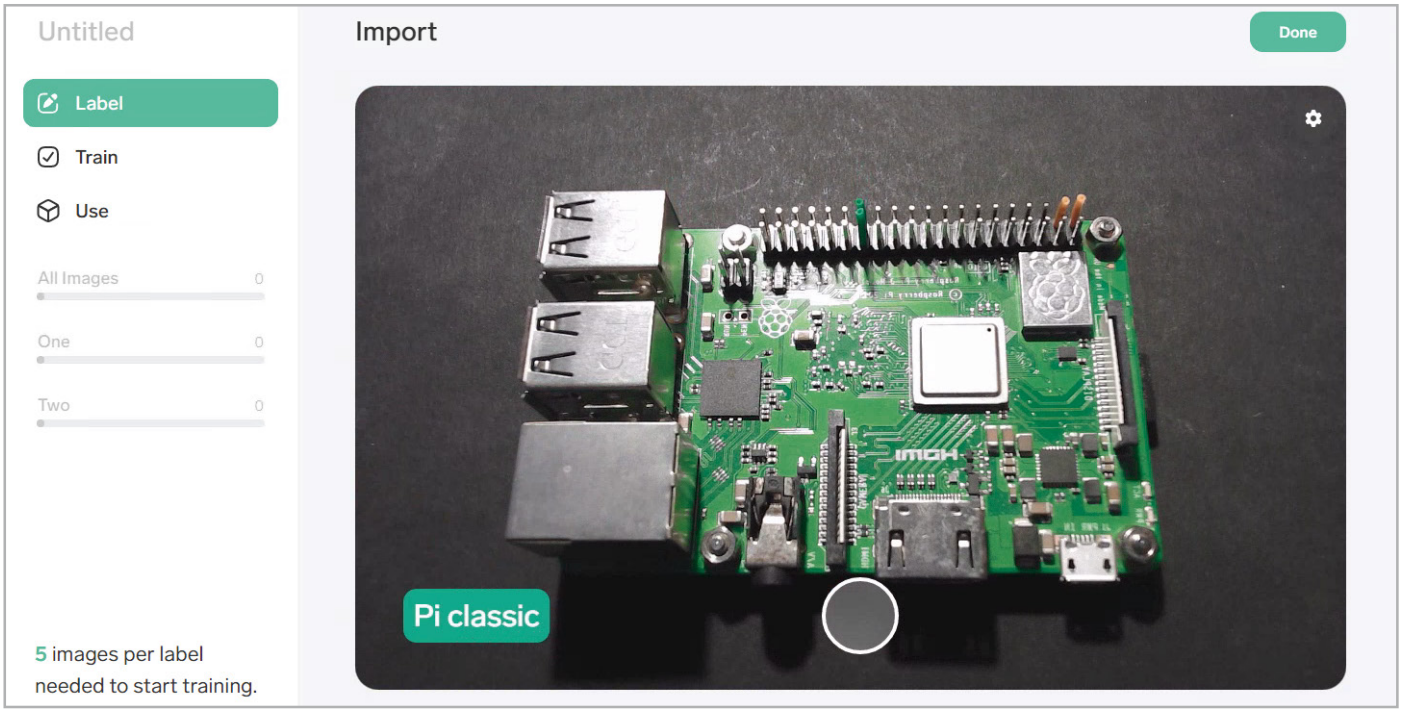



Bild 5: Erfassung von Bildmaterial

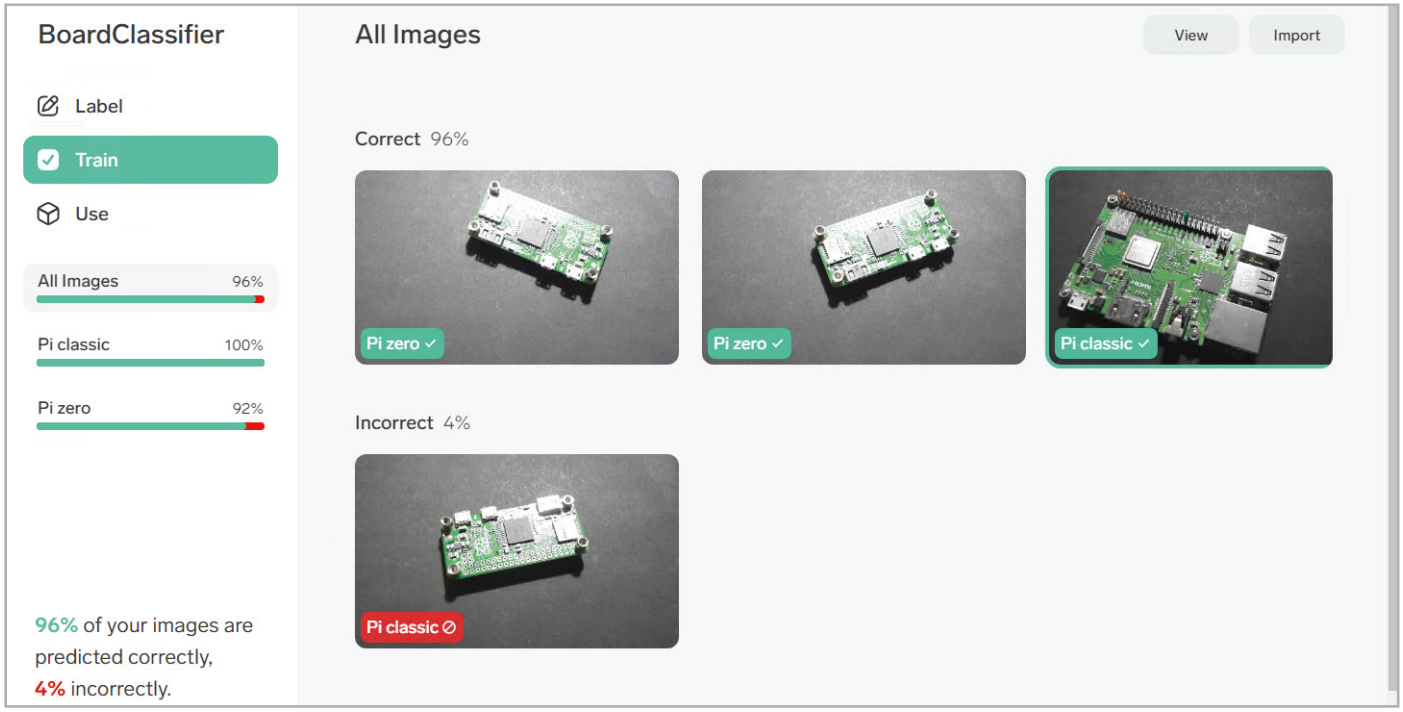



*Bild 6: Trainieren des Neuronalen Netzes*

Hier wird zunächst die Option „Camera“ gewählt. Es erscheint das Livebild der Webcam zusammen mit der Möglichkeit, eine Bildkategorie („Label“) einzugeben (Bild 5). Über den „Auslöser“, d. h. das weiß umrandete Feld in der Mitte des unteren Bildrands, wird das Foto gespeichert. Anschließend kann das nächste Bild aufgenommen werden.

Pro Kategorie sollten zunächst etwa ein Dutzend Bilder erstellt werden. Mit „Done“ wird die Erfassung der Trainingsbilder beendet.

Nun kann das Neuronale Netz trainiert werden (Bild 6). Über die Auswahl „Train“ wird diese Phase gestartet. Da für diesen ersten Test nur wenige Bilder verwendet wurden, ist das Training rasch abgeschlossen. Meist sind bereits nach wenigen Sekunden nahezu 100 % der Bilder korrekt klassifiziert.

Mit der Funktion „Play“ kann die Genauigkeit des Modells getestet werden. Durch Variation von Objektabständen, Änderung der Beleuchtung, verschiedene Positionen und Ausrichtungen etc. kann festgestellt werden, in welchen Situationen das Modell bereits präzise arbeitet und wo eventuell noch Verbesserungen möglich sind. Falls die Erwartungen noch nicht erfüllt wurden, können der Trainingsbasis noch weitere Fotos zuhinzugefügt werden.

Schließlich kann das Modell mit „Use“ nochmals unabhängig einem abschließenden Test unterzogen werden. Dazu wird wieder ein Live-Kamerastream eingeblendet. Wenn nun ein Board im Blickfeld der Kamera erscheint, wird dieses automatisch kategorisiert. Bild 7 zeigt die korrekte Erkennung eines Raspberry Pi Zero.

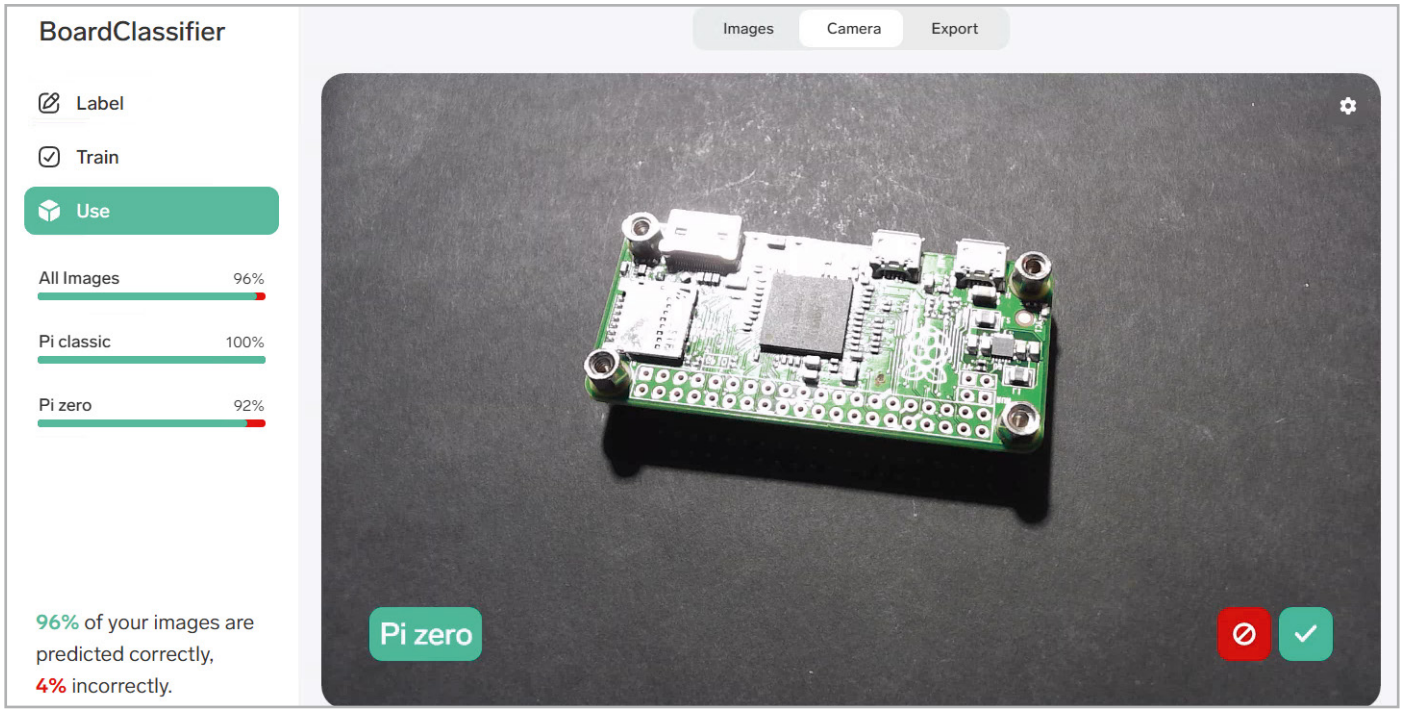



*Bild 7: Erfolgreicher Test am PC*

Tipp: Es ist vorteilhaft, der Testbildsammlung auch eine Kategorie „Kein Board" (bzw. „No board") hinzuzufügen. Die entsprechenden Fotos sollten den typischen Hintergrund ohne ein Board zeigen. Damit wird später auch richtigerweise „Kein Board" angezeigt, wenn auf dem Bild tatsächlich kein Raspberry zu finden ist. Ansonsten wird immer mehr oder weniger zufällig eine Boardversion angezeigt, auch wenn nur ein leerer Hintergrund von der Kamera erfasst wird, da Lobe erfolglos versucht, unbedingt etwas zu „erkennen".

## Die Pi-Cam als Datensammler

Die direkte Verwendung der Pi-Cam zur Aufnahme der Trainingsbilder (s. Bild 8) hat einen entscheidenden Vorteil. Wenn für das Training und die spätere Anwendung die gleiche Kamera verwendet wird, sind die Klassifizierungsergebnisse meist wesentlich besser als beim Einsatz verschiedener Systeme. Dies liegt u. a. daran, dass im ersten Fall alle wichtigen Bilddaten wie Auflösung, Farbqualität oder Bildschärfe für Training und Anwendung identisch sind.

Mithilfe des Programms sample_collector.py (s. Download-Paket) können die Trainingsbilder auch mit dem Raspberry Pi sehr einfach und direkt mittels Pi-Cam erstellt werden:

```
from picamera import PiCamera
from time import sleep

number_of_samples=12
delaytime=1

camera = PiCamera()
camera.resolution = (640, 480)
camera.start_preview()
sleep(2)

print("Camera started...")

for i in range(number_of_samples):
    print("taking sample no ", i)
    camera.capture('/home/pi/Lobe/samples/image%s.jpg' % i)
    sleep(delaytime)

camera.stop_preview()
print("Ready")
```

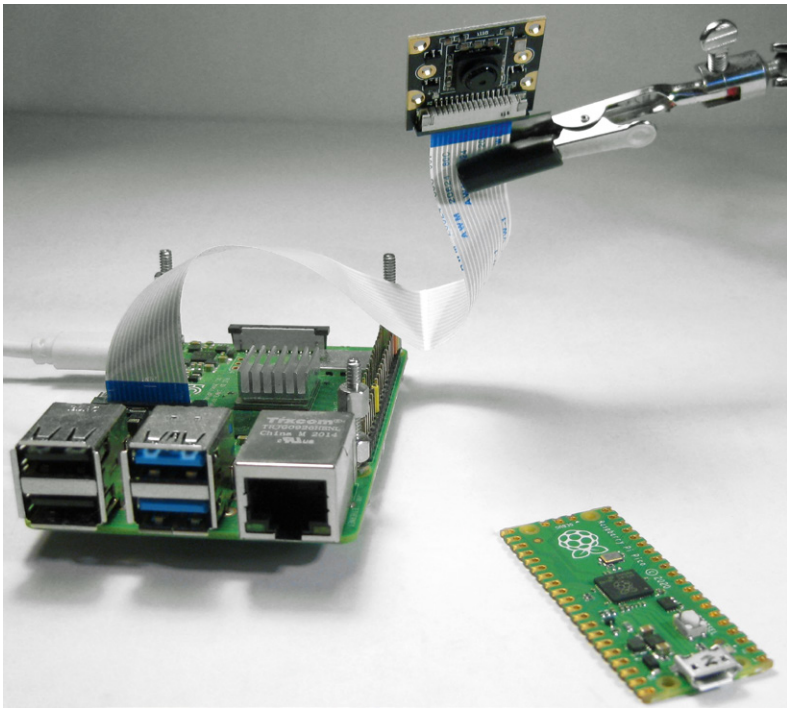


*Bild 8: Elektronische Selbsterkenntnis*

Über die Variable „number_of_samples" kann die Anzahl der aufzunehmenden Trainingsbilder eingestellt werden. Die Voreinstellung von zwölf Bildern genügt typischerweise für erste Versuche. Für komplexere Anwendungen können jedoch auch weit über 100 Bilder erforderlich sein, um zuverlässige Ergebnisse zu erhalten.

Mit „delaytime" kann die Zeitverzögerung zwischen zwei Aufnahmen in Sekunden gewählt werden. Im Verzeichnis „/home/pi/Lobe/samples" werden die Bilder durchnummeriert abgespeichert. Es ist also sinnvoll, die verschiedenen Objekte separat aufzunehmen. So können die Bilder dann direkt nach der Aufnahme in die jeweiligen Unterverzeichnisse mit der korrekten Bezeichnung (Pi Classic, Pi Zero etc.) einsortiert werden.

## Importieren von Trainingsdaten aus dem Internet

Neben dem Erstellen eigener Bilddaten ist es möglich, entsprechende Sammlungen aus dem Internet zu nutzen. Hier sind inzwischen mehrere Projekte verfügbar, die u. a. auch den Raspberry Pi als minimalistisches KI-System einsetzen. Diese reichen von kuriosen Anwendungen wie einem „Waschbärdetektor" bis hin zu durchaus praxistauglichen Systemen wie einem automatischen Mülltrenner [2].

Für das Training der Systeme kamen dabei bislang speziell entwickelte Python-basierte Netze zum Einsatz. Mit der Verfügbarkeit fertiger, universeller Trainingsprogramme wie Lobe wird die Erstellung neuer Anwendungen wesentlich vereinfacht. Die vorhandenen Bilddatenbanken können dabei nutzbringend weiterverwendet werden. So enthält die Racoon-Datenbasis eine Vielzahl von Fotos mit Waschbären [3], sodass das Trainieren eines Lobe-Netzwerks zu sehr guten Ergebnissen führen sollte.

Zudem findet sich eine Vielzahl von Abbildungen und Fotos der verschiedenen Raspberry-Varianten im Netz. Auch hier wäre es sicherlich hochinteressant zu testen, wie gut sich allein mit diesen Daten ein Lobe-Netzwerk trainieren ließe.

Natürlich darf hier der Hinweis nicht fehlen, dass bei Verwendung von internetbasierten Bilddatenbanken die Urheber- und Persönlichkeitsrechte zu beachten sind.

## Exportieren von Modellen

Eine der wichtigsten Funktionen von Lobe ist die Exportmöglichkeit der Netzwerkdaten in verschiedenen Formaten (s. Bild 9). So ist es möglich, die Daten direkt im klassischen TensorFlow-Format zu exportieren. Damit könnte man dann auf einem PC eine Python-basierte Anwendung umsetzen. Auch der Export in TensorFlow.js ist vorgesehen. Damit wird eine einfache Erstellung von Websites mit KI-relevanten Inhalten möglich.

Hier ist jedoch die Variante TensorFlow Lite von besonderer Bedeutung. Sie gestattet es beispielsweise, Bilderkennungsverfahren auf mobilen Geräten wie Smartphones oder Tablets umzusetzen. Darüber hinaus ist dieses Export-Format auch bestens für Anwendungen auf dem Raspberry Pi geeignet.

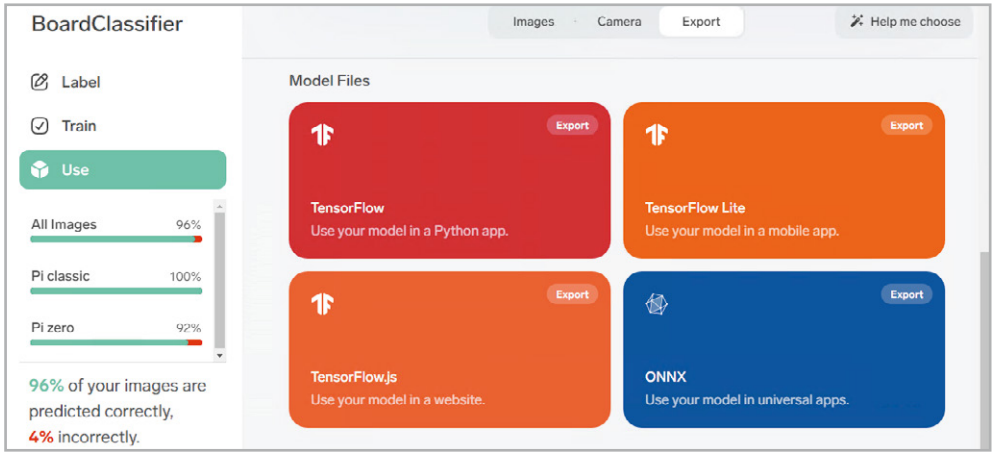



*Bild 9: Export-formate von Lobe*

Ist das gewünschte Machine-Learning-Modell mithilfe von Lobe fertiggestellt und optimiert, wird es im TensorFlow(TF)-Lite-Format exportiert.

## Raspberry Pi als Elektronikexperte

Wenn das fertig trainierte Modell als *.tflite-Datei zur Verfügung steht, kann es auf den Raspberry Pi kopiert werden (via USB-Stick, FileZilla o. Ä.). Im Home-Verzeichnis des Raspberry (home/pi/) sollte man sich dafür eine feste Verzeichnisstruktur anlegen, z. B.:

home/pi/
lobe/
/model
/samples

Das tflite-Modell wird direkt in das „/model"-Verzeichnis kopiert. Hier muss auch noch die Datei „labels.txt" erstellt werden, welche die Bezeichnungen der einzelnen Boards enthält:

- Pi classic
- Pi pico
- Pi zero
- No board

Im Verzeichnis „lobe" wird nun das Python-Programm für die Anwendung des Modells erstellt. Hierzu kann wieder Thonny verwendet werden. Das Download-Paket [4] enthält einen entsprechenden Vorschlag (classifier_live.py):

```
import cv2, os
from lobe import ImageModel

cap = cv2.VideoCapture(0)
cap.set(3,640) # set Width
cap.set(4,480) # set Height
green=(0,255,0)
path = '/home/pi/Lobe'

print("Loading model...")
model = ImageModel.load('/home/pi/Lobe/model')
path = '/home/pi/Lobe'

while(True):
    ret, frame = cap.read()
    cv2. imwrite(os. path. join(path , 'camimage.jpg'), frame)
    result = model.predict_from_file('/home/pi/Lobe/camimage.jpg')
    cv2.putText(frame,result.prediction,(30,30), cv2.FONT_HERSHEY_SIMPLEX, 1,green,2)
    cv2.imshow('press "q" to quit', frame)
    print(result)
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        print("bye...")
        break

cap.release()
cv2.destroyAllWindows()
```

Die erforderliche Lobe-Bibliothek kann über drei Konsolenanweisungen installiert werden:

```
pip3 install setuptools
pip3 install tensorflow==1.13.1
pip3 install git+https://github.com/lobe/lobe-python
```

Zudem muss OpenCV verfügbar sein. Die zugehörigen Installationsanweisungen wurden bereits im Beitrag V „Objekterkennung mit einer See-and-Talk-Box" im ELVjournal 1/2022 erläutert und können dort bei Bedarf nachgeschlagen werden [5].

Im Programm kann nach dem Import der erforderlichen Bibliotheken wieder die Pi-Cam aktiviert werden. Diese muss natürlich zuvor angeschlossen und im Konfigurationsmenü des Raspberry Pi eingetragen sein. Details dazu finden sich in den früheren Beiträgen zu dieser Serie (z. B. ELVjournal 5/2021, „KI-Praxis III: Handschriftenerkennung") [6]. Die Kameraauflösung wird auf 640 x 480 reduziert, um eine zügige Bildverarbeitung zu ermöglichen.

Danach wird das Modell aus dem angegebenen Verzeichnis geladen. Dies kann auch auf einem Raspberry Pi 4 einige Sekunden in Anspruch nehmen. In der Hauptschleife des Programms wird ein Kamerabild aufgenommen und unter /home/pi/Lobe/camimage.jpg zwischengespeichert.

Die Bildauswertung erfolgt dann über

```
result = model.predict_from_file('/home/pi/Lobe/camimage.jpg')
```

unter Verwendung des aktuell geladenen Modells.

Das Resultat wird über die putText-Funktion in das Kamerabild eingeblendet. Der Rest des Codes dient wieder dem koordinierten Beenden des Programms über die „Q"-Taste („quit").

Wenn das Programm gestartet wird, erscheint nach dem Laden des Modells ein Livebild der angeschlossenen Kamera. Sobald ein Raspberry-Pi-Board im Gesichtsfeld der Pi-Cam auftaucht, wird dieses erkannt und klassifiziert.

Bild 10 zeigt beispielsweise die korrekte Erkennung eines Raspberry Pi Pico.


*Bild 10: Ein Raspberry Pi Pico wurde korrekt erkannt.*

## YOLO

Die mit Lobe erstellten Modelle zur Bilderkennung bzw. -klassifizierung verwenden das gesamte Bild als Eingabe. Dabei ist es günstig, wenn das jeweilige Objekt einen großen Teil des Bilds einnimmt. Die genaue Lokalisierung des Objekts im Gesichtsfeld der Kamera ist dagegen nicht von Bedeutung. Viele Anwendungen wie etwa das Objekt- oder Gesichtstracking erfordern jedoch, dass nicht nur die Art des Objekts im Bild erkannt wird, sondern auch seine exakte Position. Auch die Erfassung mehrerer Objekte in einem Bild erfordert die Bestimmung der jeweiligen Positionen der erkannten Gegenstände im Kamerabild.

Für diese Aufgaben sind komplexere Objekterkennungsmodelle erforderlich. Sogenannte YOLO-Architekturen (You Only Look Once) können hier weiterhelfen. Sie fügen den zu untersuchenden Bilddaten zunächst ein Koordinatengitter hinzu. Dann wird mit den bereits bekannten Methoden die An- oder Abwesenheit des gesuchten Objekts bzw. der Objekte in jedem Gitter bestimmt.

Zusätzlich kommen sogenannte Anker zum Einsatz. Dabei werden „Anchor Points" auf die zu erwartende Objektgröße angepasst. Diese erleichtern eine Vorauswahl von infrage kommenden Bildmerkmalen. Bei den vortrainierten Modellen in den letzten Beiträgen zu dieser Serie wurden bereits Ankerpunkte verwendet. Werden beim YOLO-Verfahren einzelne Merkmale erkannt, wird für jede Gitterzelle ein Satz von Vorhersagen berechnet. Vorhersagen mit geringer Wahrscheinlichkeit werden zunächst verworfen. Die verbleibenden Zellen enthalten dann mit hoher Wahrscheinlichkeit die gesuchten Objekte.

Die zentrale Idee hinter You Only Look Once besteht also darin, das Gesamtbild in kleine Regionen bzw. Gitterbereiche aufzuteilen und ein einzelnes Neuronales Netzwerk auf jedes Gitternetz anzuwenden. Dabei spielen drei Parameter eine wichtige Rolle:

- Anzahl der Gitterzellen, in die das Bild aufgeteilt wird
- Anzahl der Begrenzungsrahmen, die für jede Gitterzelle berechnet wird
- Anzahl verschiedener Klassen, die für jede Gitterzelle vorhergesagt werden können

Das Bild wird z. B. in 5 x 10 Gitterzellen (S = 50) unterteilt. Jeder Zelle wird eine gewisse Wahrscheinlichkeit zugeordnet, dass sie ein gesuchtes Objekt enthält. Die Zellen mit hohen Wahrscheinlichkeiten werden mit Einzelrahmen gekennzeichnet. So wird klar, wo sich Objekte im Bild befinden könnten. Es ist jedoch noch nicht festgelegt, um was es sich handelt. Daher wird im Anschluss jeder Zelle eine Klassenwahrscheinlichkeit zugeordnet. Die Einzelrahmen werden dann zu Objektrahmen zusammengefasst.

Anschließend wird die Wahrscheinlichkeit dafür bestimmt, dass der Objektrahmen ein bestimmtes Objekt wie z. B. ein Fahrzeug, eine Person oder aber einen Raspberry Pi enthält. Überschreitet diese Wahrscheinlichkeit einen bestimmten Schwellenwert, wird dem Objekt die entsprechende Kategorie zugeteilt.

Das Trainieren von YOLO-Systemen ist allerdings nochmals deutlich aufwendiger als das im letzten Abschnitt vorgestellte Verfahren. Insbesondere sind nicht nur eine Vielzahl von Bildern bzw. Fotos erforderlich, vielmehr muss auf jedem der Bilder das gesuchte Objekt zunächst manuell markiert und bestimmt werden. Dies ist auch mit speziell dafür entwickelten Tools mit erheblichem Aufwand verbunden. Mit der jeweiligen Kategorie und den festgelegten Koordinaten kann dann wiederum das YOLO-System trainiert werden.

Allerdings wurde von Entwicklern des Lobe-Systems in Aussicht gestellt, dass die Software um die Erstellung von YOLO-Modellen erweitert wird. Es ist zu erwarten, dass dies das Trainieren entsprechender Netzwerke deutlich vereinfachen würde.

Die Klassifizierung und Koordinatenbestimmung auf Hunderten von Bildern kann Tage oder Wochen in Anspruch nehmen. Damit dürfte klar sein, dass dieser Aufwand nur in ganz speziellen Fällen gerechtfertigt ist, beispielsweise, wenn eine bestimmte Tierart oder fest definierte Fahrzeugtypen automatisch erfasst werden sollen.

Im Allgemeinen wird man daher zunächst immer versuchen, ein bereits vortrainiertes Modell zu einem bestimmten Projekt zu finden. Im Internet finden sich hierzu viele Ansatzpunkte. Nur wenn diese Möglichkeit keinen Erfolg bringt, kann es sinnvoll sein, eigene YOLO-Systeme zu trainieren.

## Fazit und Ausblick

In diesem Beitrag wurde gezeigt, wie Neuronale Netzwerke in Eigenregie trainiert werden können. Da dieses Vorgehen „zu Fuß", also durch Erstellen eines neuen Netzwerks in Python sehr aufwendig ist, wurde auf ein universelles Trainingswerkzeug zurückgegriffen. Mit Lobe steht ein derartiger Trainer kostenlos zur Verfügung.

Damit lassen sich über selbst erstellte Datenbanken eigene Modellanwendungen schnell und effizient umsetzen. Über den Datenexport zu einem Raspberry Pi können die fertig trainierten Netzwerkstrukturen dann auch auf einem Kleinrechner eingesetzt werden.

Auf diese Weise wurde es möglich, dass sich der „RasPi" sozusagen selbst erkennt. Interessanterweise hat diese Fähigkeit zur Selbsterkenntnis in der Biologie, der Psychologie oder Philosophie eine immense Bedeutung. Dennoch ist bekannt, dass nur wenige Primaten und einige Meeressäugetiere wie Delfine oder Killerwale in der Lage sind, ihr eigenes Spiegelbild als solches zu erkennen. Natürlich darf man die hier vorgestellten Ergebnisse nicht überinterpretieren, jedoch zeigen sie, dass die Methoden der KI auch auf Kleinsystemen zu erstaunlichen Resultaten führen können.

Im nächsten Beitrag wird es schließlich um autonome Systeme gehen. Es soll demonstriert werden, wie die in den letzten Beiträgen erarbeiteten Konzepte auf praxisrelevante Aufgaben angewendet werden können. Auch hier wird natürlich wieder der Raspberry Pi als maschinelles Lernsystem im Vordergrund stehen. ELV

| Material | Artikel-Nr. |
|---|---|
| Raspberry Pi 4 Model B, 8 GB RAM | 250567 |
| Standard-5-MP-Kamera für Raspberry Pi | 145134 |

## i Weitere Infos

[1] https://www.lobe.ai
[2] https://utopia.de/news/racoon-die-muelltonne-die-den-muell-sortiert/
[3] https://github.com/datitran/raccoon_dataset/tree/master/images
[4] Download-Paket zu diesem Beitrag: Artikel-Nr. 252711
[5] Fachbeitrag „Objekterkennung mit einer See-and-Talk-Box", ELVjournal 1/2022: Artikel-Nr. 252461
[6] Fachbeitrag „Handschrifterkennung", ELVjournal 5/2021: Artikel-Nr. 252233

Alle Links finden Sie auch online unter: de.elv.com/elvjournal-links