KI & Maschinenlernen auf dem Mikrocontroller

Teil III: Tensorflow Lite Micro auf Arduino Nano BLE Sense

Urheber: Thomas Jörg

Stand 12. März 2023

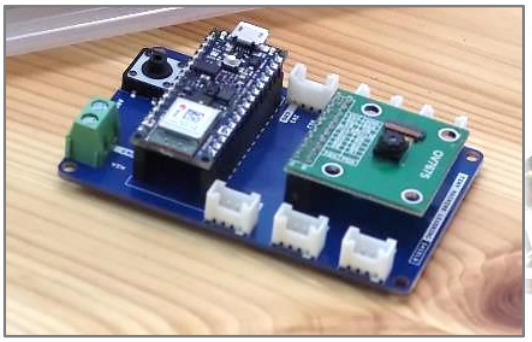


Abb. 2 Eigenes Foto TF Kit

Abb. 1 TensorFlow Lite Logo [Apache License 2.0]

Abb. 3 [Gemeinfrei] erzeugt mit DALL-E; Prompt „a cute little robot staring at many different pictures, white background, anime style“ von Jörg [CC BY-SA 4.0 International]

Inhaltsverzeichnis

Abb. 4 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „cat rocket scientist assembling a highly complicated clockwork, steampunk version, digital art“ von Jörg [CC BY-SA 4.0 International]

Vorwort

Künstliche Intelligenz auf Mikrocontrollern (KI auf IoT-µC) entwickelt sich gerade zu einer wichtigen Anwendung, deren Tragweite noch nicht voll abzusehen ist – man kann die Bedeutung aber erahnen.

Abb. 5 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „cute little robot works hard with hammer drill, steampunk version, digital art“ von Jörg [CC BY-SA 4.0 International]

KI auf IoT-µC übernehmen zum Beispiel in Smart-Factories zentrale Aufgaben wie Steuerung, Qualitätskontrolle, predictive Maintenance, Anomalie Detections usw.

Wären es simple IoT-µCs, so würden diese enorme Datenströme erzeugen, die anderswo von Zentralrechnern ausgewertet werden müssten, und die zugehörige Bandbreite für die Telemetriedaten wären eine enorme Belastung für die jeweiligen Netzwerke. Man stelle sich eine simple Webcam mit QVGA-Auflösung und 25 fps vor. In jeder Sekunde erzeugt solch eine RGB-Kamera 25*320*240*3 Byte = 5625 kB = 5,5MB. Soll in einer Produktionshalle alles vernetzt werden, gerät eine Netzwerkinfrastruktur womöglich an seine Belastungsgrenze.

Findet allerdings ein Großteil der Auswertung vor Ort statt, wird lediglich die relevante Information – im Falle einer KI oftmals nur eine Klassifikation – übertragen. Zudem werden die Zentralrechner entlastet. Eine engmaschige Ausstattung mit IoT-Geräten wird so erst möglich. Weitere Faktoren wie Echtzeitfähigkeit und Robustheit spielen ebenfalls eine Rolle.

Von vielen Firmen werden intelligente IoT-Geräte deshalb als strategisch eingeordnet, die Forschung an diesem Thema unterliegt oftmals der Geheimhaltung.

Es stellt sich die Frage, wer damit wie umgehen soll? Das Personal muss geschult werden, denn zwei komplexe Themenbereiche werden hier miteinander verknüpft: IoT und Künstliche Intelligenz.

Aber:

KI in Form von Neuronalen Netzen, die auf Mikrocontrollern laufen, bedeuten Frust.
Das Gebiet ist auch drei Jahre nach seinem Start immer noch unausgereift. Viele gängige Mikrocontroller werden unterschiedlich gut unterstützt. In einigen Fällen, wie z.B. dem ESP32, existieren zwar Portierungen von Tensorflow Lite Micro (TFLM), sind aber im „Experimental“-Stadium steckengeblieben. Das bedeutet, dass zum Beispiel die wichtige, zentrale Umstellung von uint_8 auf int_8 hier nicht implementiert wurde, weshalb lediglich alte Tensorflow-Versionen (1.7) möglich sind und viele wichtige Features nicht laufen (diverse Layeroperationen im MicroOps-Resolver). Ganz abgesehen davon, dass Codes zwischen Mikrocontrollern nicht portierbar sind.

TFLM unterliegt einer ständigen Weiterentwicklung, weshalb die Skripten und Workflows sich ebenfalls ständig ändern. Die Inhalte von Bibliotheken werden laufend abgeändert, weshalb sich oftmals früher lauffähige Versionen plötzlich nicht mehr kompilieren lassen (Beispiel: `version.h` fehlt in den aktuellen TFLM-Versionen nach 2021).

Das System, wofür man sich bei Google entschieden hat – wohl hauptsächlich aus historischen Gründen – ist der Arduino 33 BLE Sense. Er ist im Bereich der KI in etwa das, was in der ‚normalen‘ Mikrocontroller-Szene der Arduino Uno ist: Das Brot-und-Butter-System, das zu Lehrzwecken eingesetzt wird. Und deshalb wird er hier in diesem Skript verwendet.

Warum das TinyML-Kit (Arduino 33 BLE Sense)?

Das TinyML-Kit wurde von der Universität Harvard entwickelt als Modell- und Lehrsystem für Tensorflow Micro. Es handelt sich zwar einerseits nicht um die stärkste Hardware, aber die bei weitem am besten unterstützte! Diese Unterstützung führt dazu, dass man weniger Hürden und Schwierigkeiten überwinden muss. Außerdem gibt es sehr gute – auch kostenfreie – Kurse dazu, nämlich von der Universität Harvard:

Harvard:

https://www.edx.org/course/fundamentals-of-tinyml
https://www.edx.org/course/applications-of-tinyml
https://www.edx.org/course/deploying-tinyml

Pete Warden, einer der Architekten von TensorFlow Lite Micro bei Google, hat auch bei der Erstellung der Harvard-Kurse als Konzepter und Dozent mitgewirkt. Er hat einen eigenen Blog mit wichtigen Beiträgen:

https://petewarden.com/

(zum Beispiel dieser Blogeintrag, der in diesem Tutorial verwendet wird:
https://petewarden.com/2021/08/05/one-weird-trick-to-shrink-convolutional-networks-for-tinyml/)

Was hat das Board, bzw. das Kit zu bieten?

„Alles“.

(Etwas genauer: Alles, was man für die umfassende Einführung in das Thema ‚KI auf µC‘ benötigt)

https://docs.arduino.cc/tutorials/nano-33-ble-sense/cheat-sheet
https://docs.arduino.cc/hardware/nano-33-ble-sense
https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf

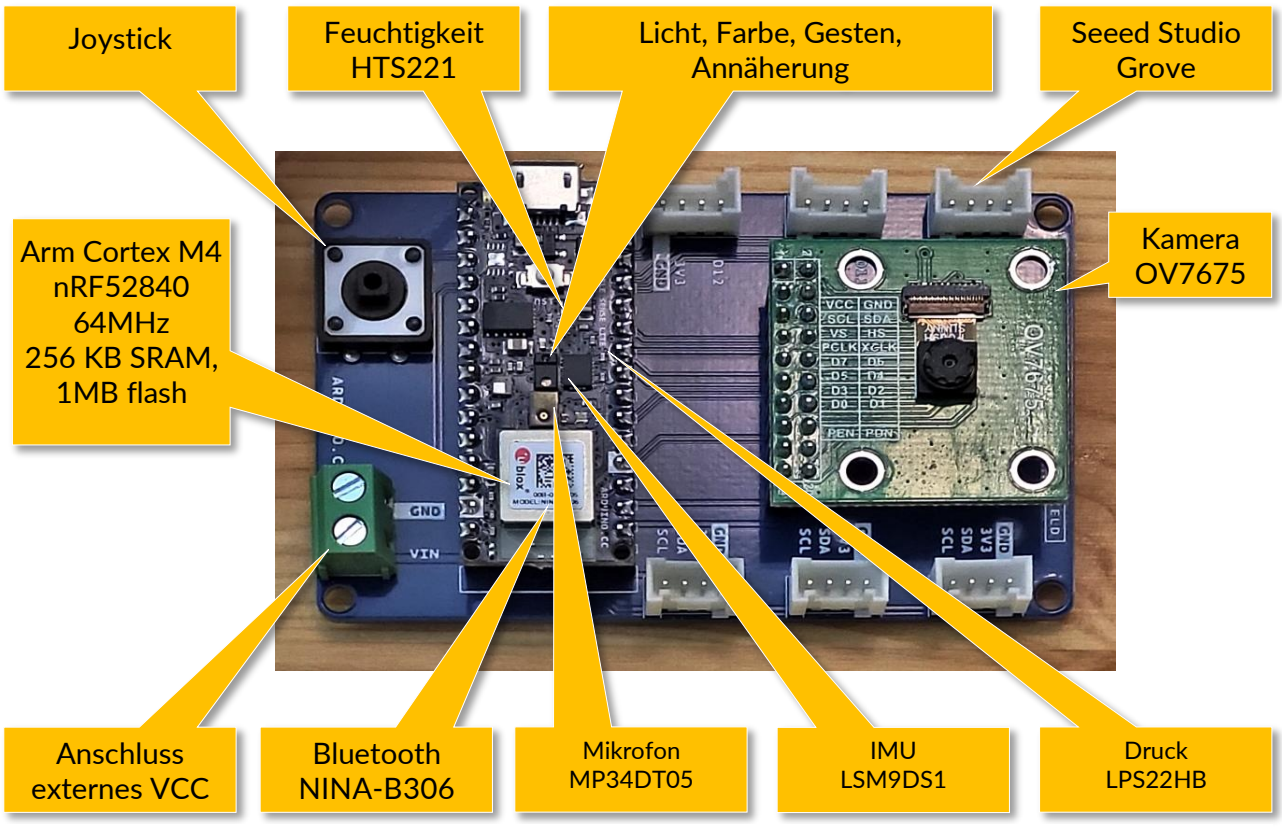


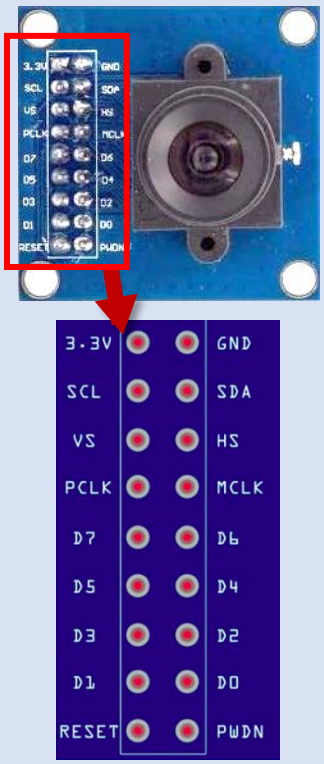
Abb. 6 Eigenes Foto & eigene Beschriftungen

In diesem Tutorial wird ausschließlich die OV7675 benötigt. Eigentlich ist die OV7670 die hauptsächlich unterstützte Variante dieser VGA-Kamera. Allerdings ist diese Kamera sehr schwierig zu verwenden, weil sie verkabelt werden muss. Die OV7675 ist, weil sie einen eigenen Connector auf dem Shield besitzt, deutlich einfacher in der Handhabung.

Wiring der OV7670 auf dem Breadboard

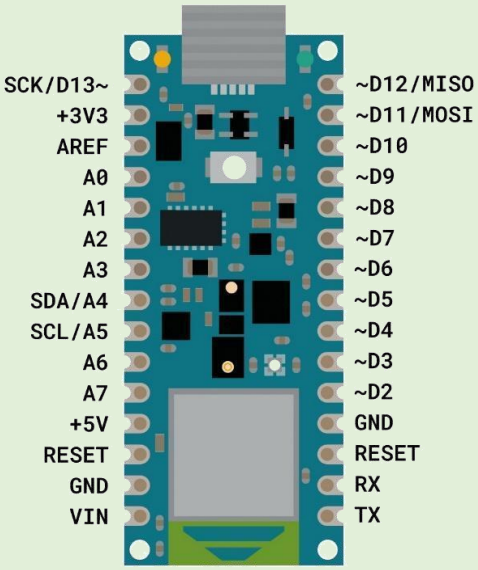
Hier ist beispielhaft das Wiring der OV7670 abgebildet. Sowohl Kamera als auch Arduino laufen auf 3,3Volt. Der Arduino ist dazu in der Lage, die Kamera mit 3,3V zu versorgen, aber es ist zu empfehlen, beide Geräte an eine externen 3,3 Volt-Spannungsquelle anzuschließen:

OV7670 PinOut



OV7670 Pin	Arduino Pin
GND	GND
SLC	A5
SDA	A4
VS/VSYNC	D8
HS/HREF	A1
PCLK	A0
MCLK	D9
D7	D4
D6	D6
D5	D5
D4	D3
D3	D2
D2	D0/Rx
D1	D1/Tx
D0	D10

PinOut Arduino 33 BLE Sense



Der TensorFlow Lite Micro Arbeitsfluss

0. Software-Setup:
Es werden installiert:
- Arduino-IDE mit diversen Libraries (in diesem Tutorial: V 2.01)
- Processing-IDE (in diesem Tutorial: V 4.01)
- Tensorflow/Python mit diversen Libraries (in diesem Tutorial verwendet: V 2.10)

1. Datenaufnahme
Benötigt Arduino-IDE und Processing

2. Datenaufbereitung
Einsortieren der aufgenommenen Bilddaten, benötigt lediglich den Explorer

3. und 4. Modell-Definition und Modell-Training
Installiertes Python 3.10, Tensorflow und im Optimalfall jupyter-Notebooks.

5. und 6. Modell-Deploy und Modell-Validierung
Wieder die Arduino-IDE mit der Tensorflow Lite Micro Bibliothek

Wichtiger Hinweis: Datenaufnahme und Inferenz benötigen gleiche Bedingungen

Datenaufnahme und Model-Deploy sollten auf demselben Board stattfinden, und es müssen die gleichen Skript-Varianten gewählt werden, denn sonst kann ein trainiertes Neuronales Netz nicht funktionieren.

Beispiel: In diesem Tutorial nutzt der ImageProvider – also die Ansteuerlogik für die Bildaufnahme – den QVGA-Modus, wobei eine Pixelauflösung des aufgenommenen Bildes von jeweils 72 Pixeln einprogrammiert ist. Dabei wird vom QVGA-Bild zunächst ein Bildausschnitt definiert und von diesem Ausschnitt wird jeweils der zweite oder dritte Pixel tatsächlich in das aufzubauende Bild übernommen. Dies führt zu einer besonderen Beschaffenheit des Bildes. In Anaconda/Tensorflow wird das Neuronale Netz auf diese Besonderheiten trainiert: Es trainiert zwar die Bildinhalte, allerdings ist der Inhalt und die Codierung eng miteinander verknüpft, und ein neuronales Netz kann dies nicht unterscheiden. Deshalb muss das fertige Neuronale Netz dieselbe Datenaufnahme-Logik sowohl bei Trainingsdaten als auch deployten Modell verwenden.

0. Software Setup:

Wir benötigen drei verschiedene Software-Pakete:

a. Die Arduino-IDE, um den Arduino Nano 33 BLE Sense zu programmieren
b. Anaconda für das Tensorflow-Environment & Jupyter-Notebooks
c. Processing entweder 3.5 oder mindestens 4.01 (sonst funktioniert die Serielle Kommunikation nicht)

0.1 Arduino IDE:

Download Arduino IDE von
https://www.arduino.cc/en/software

2. Erster Start, es werden viele Pakete nachinstalliert. Das ist deshalb wichtig, weil man das unter Administrator-Rechten machen muss. Das wird auf Schulrechnern oftmals vergessen.

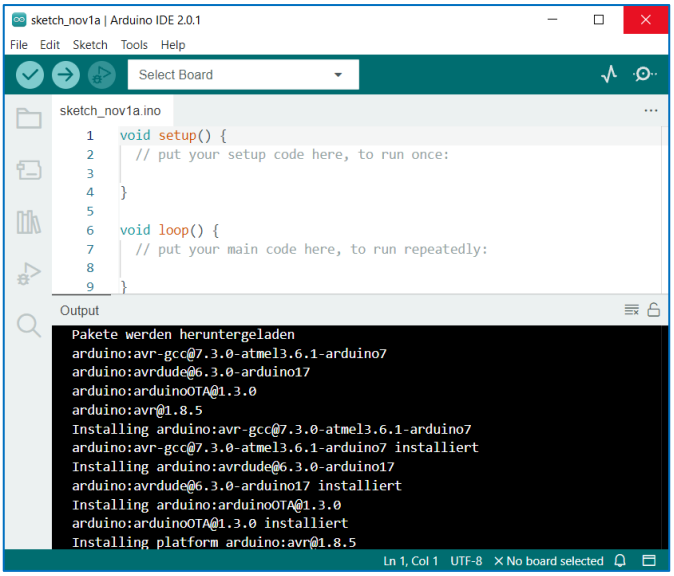


Abb. 7 Eigenes Screenshot Arduino-IDE

Setup für den Arduino 33 Nano BLE, Variante 1:
Geht man über „Select Board“ in der IDE-Menüleiste oben, dann kommt man zu diesem Fenster:

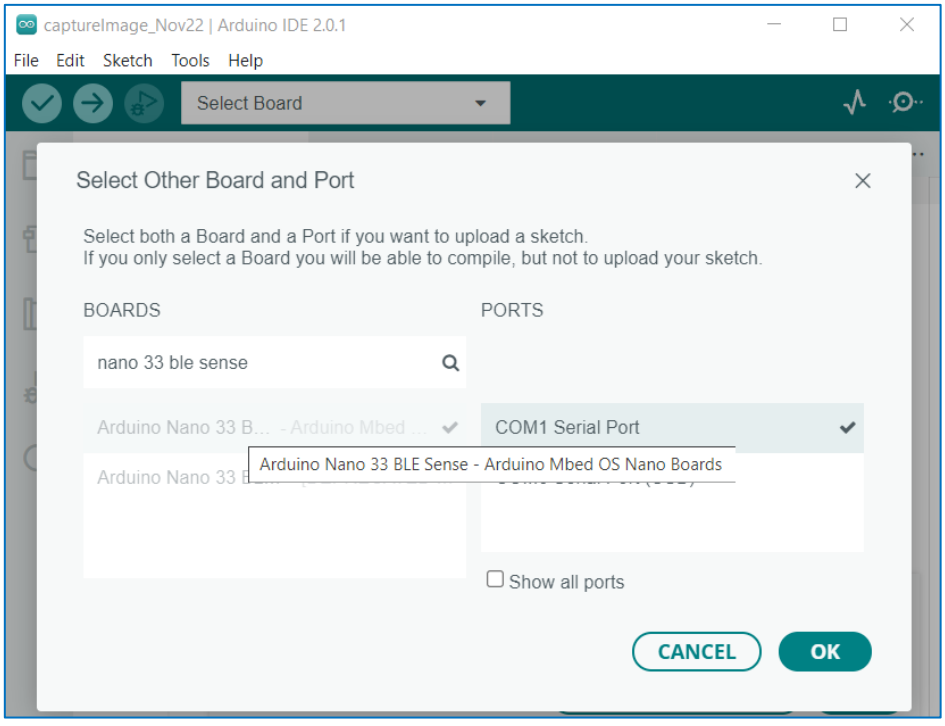


Abb. 8 Eigenes Screenshot Arduino DIE

Setup für den Arduino 33 Nano BLE, Variante 2:

Kompiliert man ein vorhandenes Skript, erkennt die IDE womöglich das Board und bietet eine Installation an:

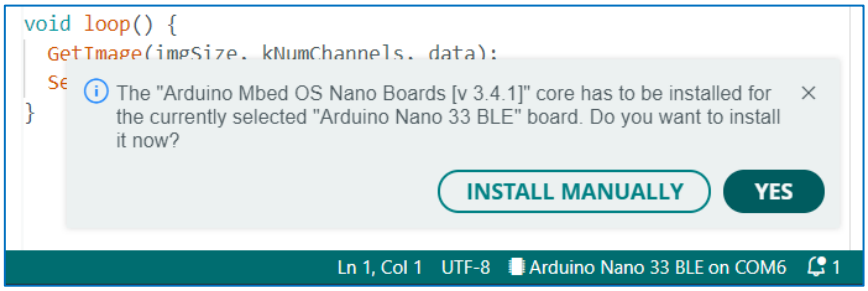


Abb. 9 Eigenes Screenshot Arduino IDE

Setup für den Arduino 33 Nano BLE, Variante 3:

Der 'klassische' Weg verläuft über den Boardmanager: Boards Manager öffnen, Suchbegriff „33 ble sense" eingeben und die aktuelle Bibliothek installieren:

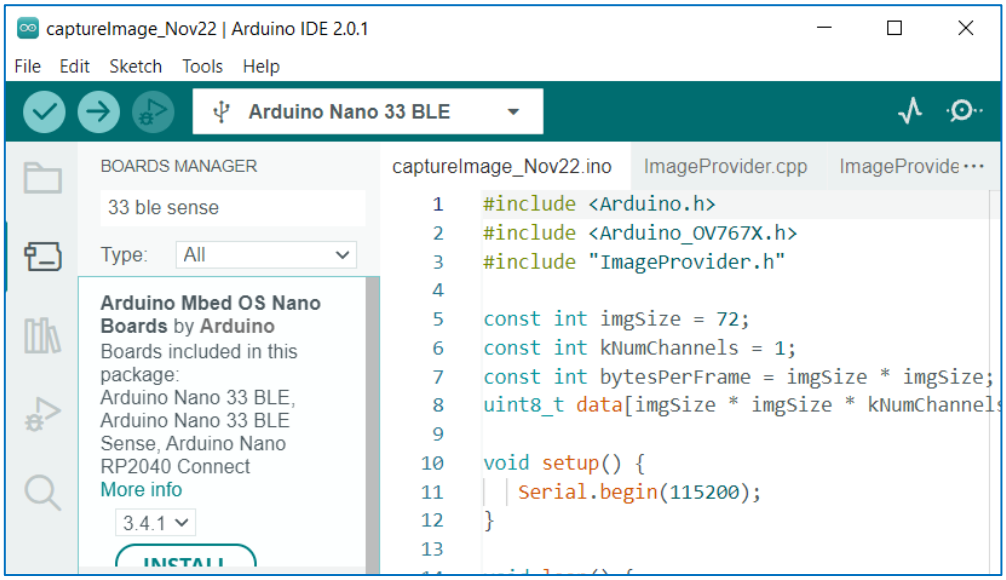


Abb. 10 Eigenes Screenshot Arduino IDE

Die Bibliothek für die Webcam OV7675

Die OV7675 wird eigentlich nicht nativ unterstützt, funktioniert aber fast genauso wie die OV7670. Deshalb wird diese Bibliothek installiert:

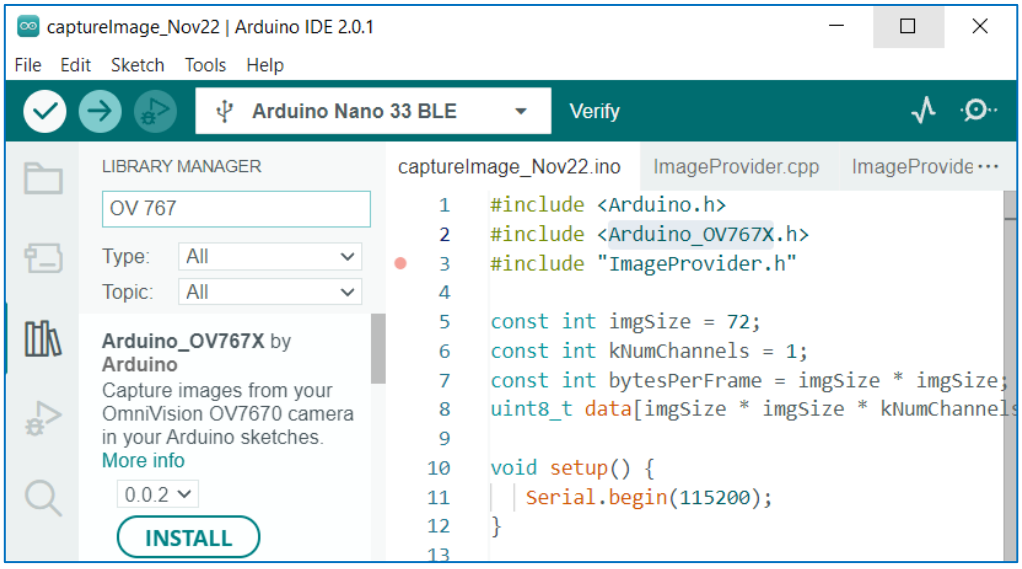


Abb. 11 Eigenes Screenshot Arduino IDE

Tensorflow Lite Micro-Installation

Früher – als alles besser war – konnte man die TFLM-Library über den Bibliotheksmanager installieren. Das geht seit Anfang 2022 nicht mehr, deshalb muss man die Library über Github herunterladen und altherkömmlich als zip-Bibliothek halbmanuell installieren:

https://github.com/tensorflow/tflite-micro-arduino-examples

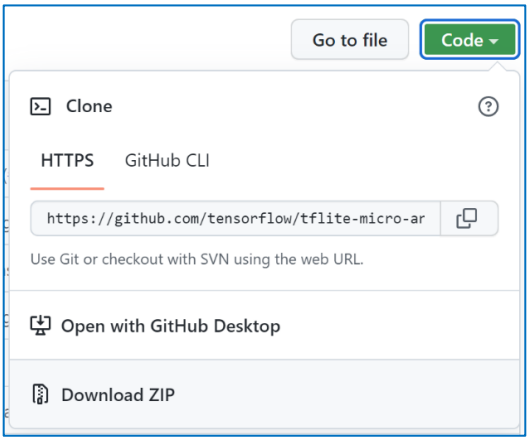


Abb. 12 Eigenes Screenshot Github Portal

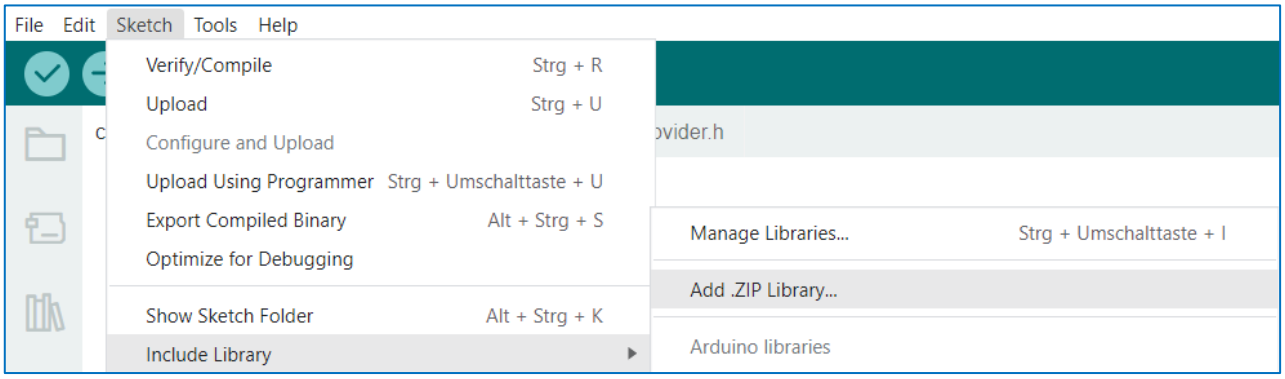


Abb. 13 Eigenes Screenshot Arduino IDE

0.2 Processing

Processing lädt man sich von der Original-Webseite herunter:

https://processing.org/

Die Software muss üblicherweise nicht installiert werden, sie wird ‚portable' ausgeliefert. Das heißt, man packt sie lediglich aus („entzippt sie"). Es wird der standardmäßige Java-Modus verwendet. Allerdings benötigt man noch zwei weitere Plugins, von denen einer nicht nachinstalliert, aber ‚kritisch' ist und eines tatsächlich nachinstalliert werden muss:

- Kann kritisch sein: Es wird die Serial-Bibliothek benötigt, um mit dem Arduino zu kommunizieren – also um die Bilddaten zu übertragen. In den frühen 4er-Versionen von Processing funktionierte diese Libary noch nicht. Alternativ kann die Version 3.54 verwendet werden.

- Weiterhin wird noch die ControlP5-Library benötigt, um die Dropdowns erzeugen und platzieren zu können. Dafür wird zunächst das Menü „JAVA -> Modes Verwalten" aufgerufen:

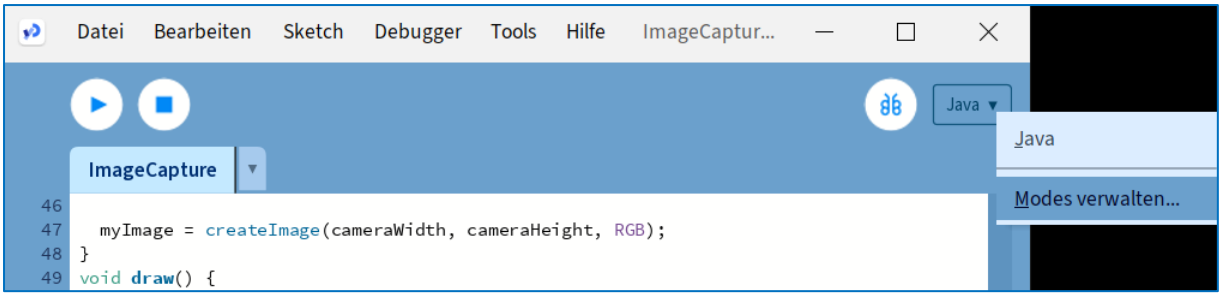


Abb. 14 Eigenes Screenshot Processing IDE

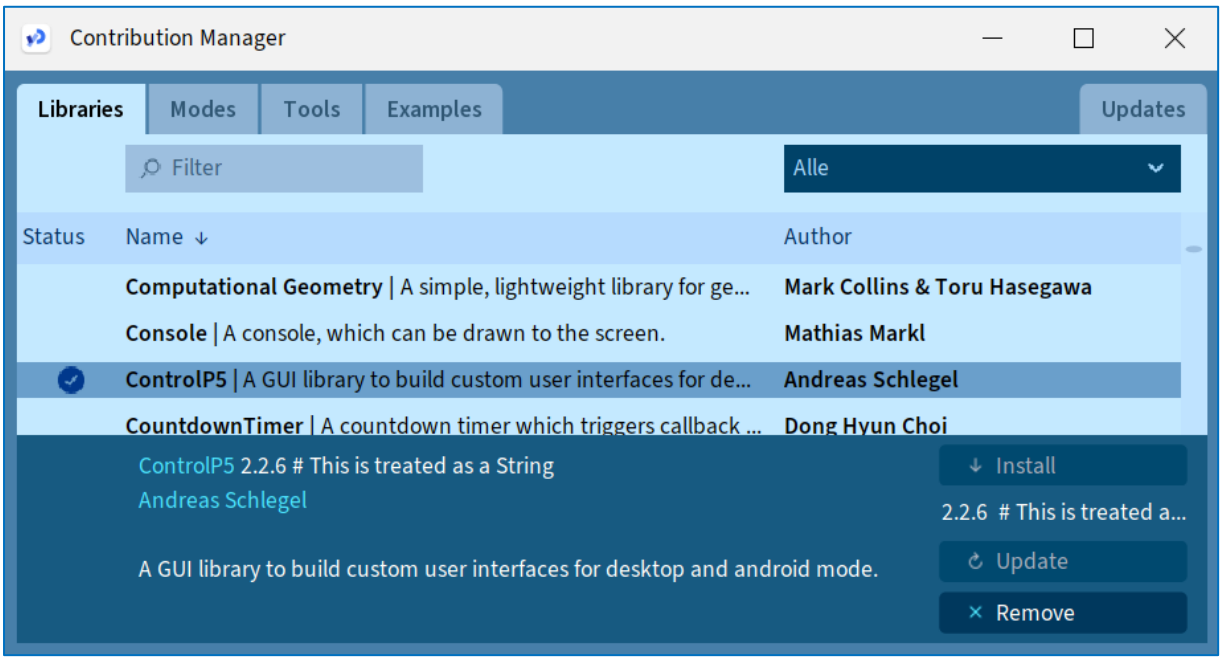


Abb. 15 Eigenes Screenshot Processing IDE

0.3 Variante A: Tensorflow-Setup in Anaconda

Anaconda lädt man sich von der Original-Webseite herunter:

https://www.anaconda.com/

Die Software stellt ein großes und mächtiges Framework zur Verfügung, das aus mehreren IDEs und Werkzeugen besteht. Es ist die sicherste und geradlinigste Methode, Tensorflow zum Laufen zu bekommen:

Installationsart

Anaconda Installation wird “Just Me” installiert.

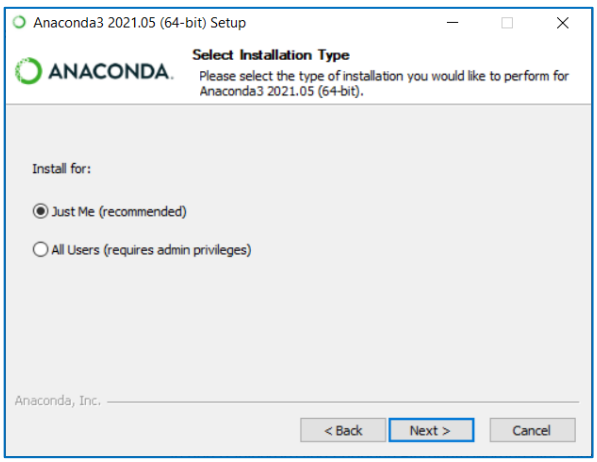


Abb. 16 Eigenes Screenshot Anaconda Setup

Anaconda Environment Setup:

Tensorflow packages werden ausgewählt. Es werden dabei nur ein Teil benötigt, die hier angewählt sind:

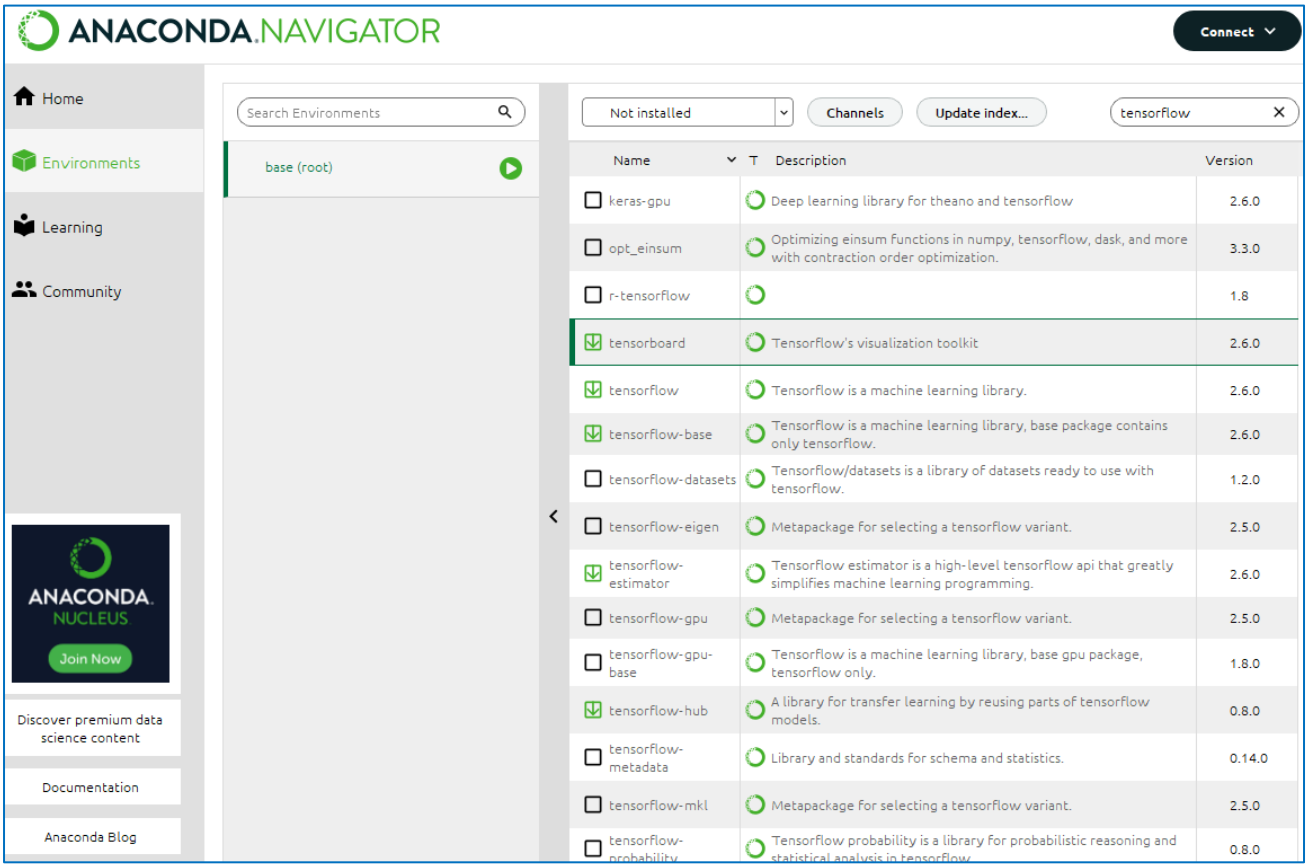


Abb. 17 Eigenes Screenshot Anaconda Environment

Keras Packages werden ausgewählt und installiert

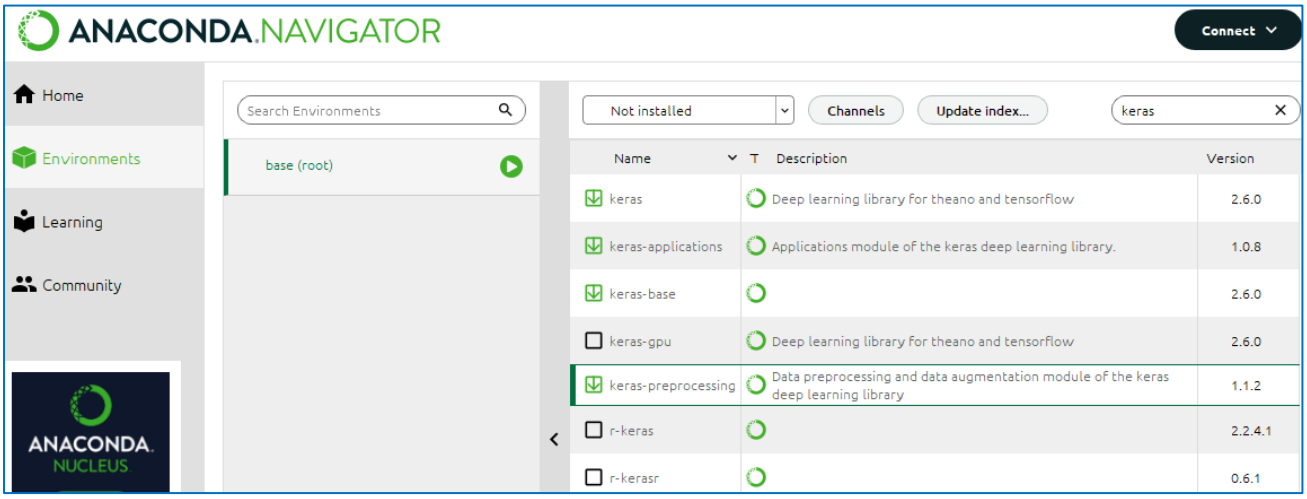


Abb. 18 Eigenes Screenshot Anaconda Environment

Überprüfen, welche Tensorflow-Version gerade aktuell installiert ist:

- innerhalb des Quelltextes eines Jupyter-Notebooks: `print (tf.__version__)`
- innerhalb der CMD-Konsole: `pip show tensorflow`

Upgrade der Tensorflow-Version mittels pip:
Anaconda als Admin starten, CMD Prompt im Navigator starten, folgenden Befehl eingeben:

`pip3 install tensorflow --upgrade --user`

0.3 Variante B: Tensorflow-Setup in Edupyter (portable Install)

Edupyter ist eine portable Variante für jupyter-Notebooks und kann im Kontext dieses Tutorials tatsächlich Anaconda ersetzen:

https://www.portabledevapps.net/edupyter.php

Die Installation sollte nicht in den von Edupyter vorgeschlagenen Standardordner erfolgen, sondern in einen selbst gewählten Ordner, wie zum Beispiel auf einem USB-Stick oder einem externen Laufwerk:

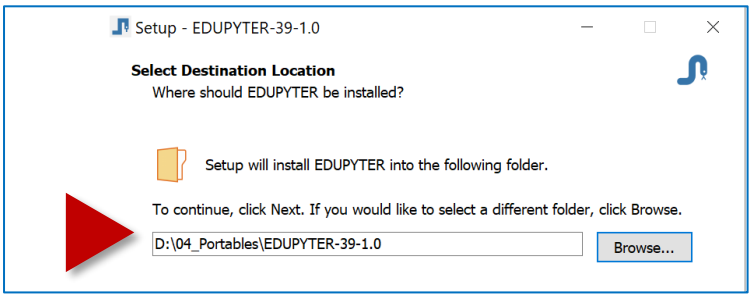


Abb. 19 Eigenes Screenshot Edupyter

Startet man die Entwicklungsumgebung, erscheint ein Icon im Systray:

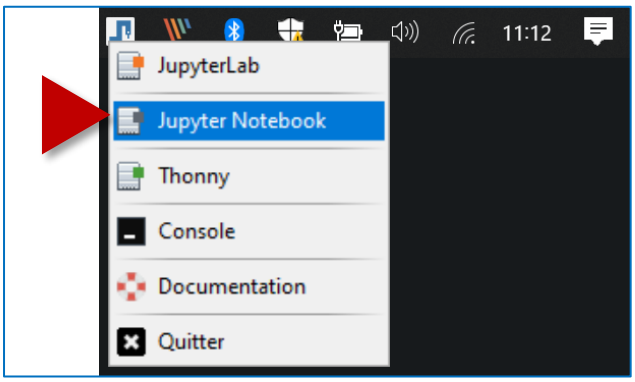


Abb. 20 Eigenes Screenshot Edupyter

Hier startet man ein Jupyter-Notebook; es sollte sich ein Browser öffnen, in dem das Notebook läuft:

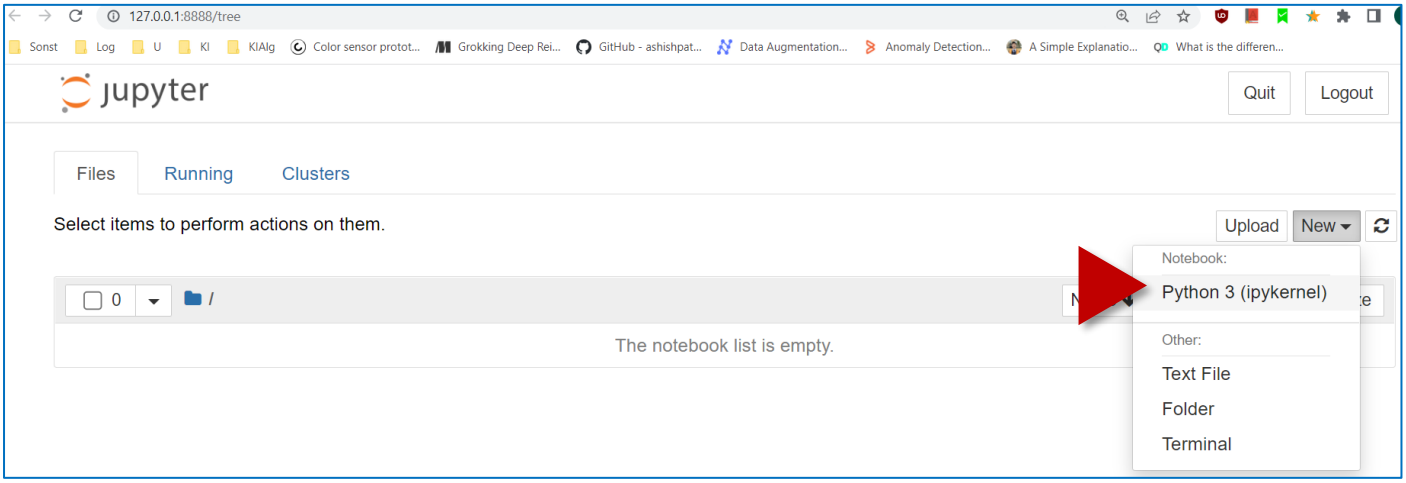


Abb. 21 Eigenes Screenshot Jupyter Notebook

Innerhalb des Notebooks müssen nun noch folgende Pakete installiert werden:

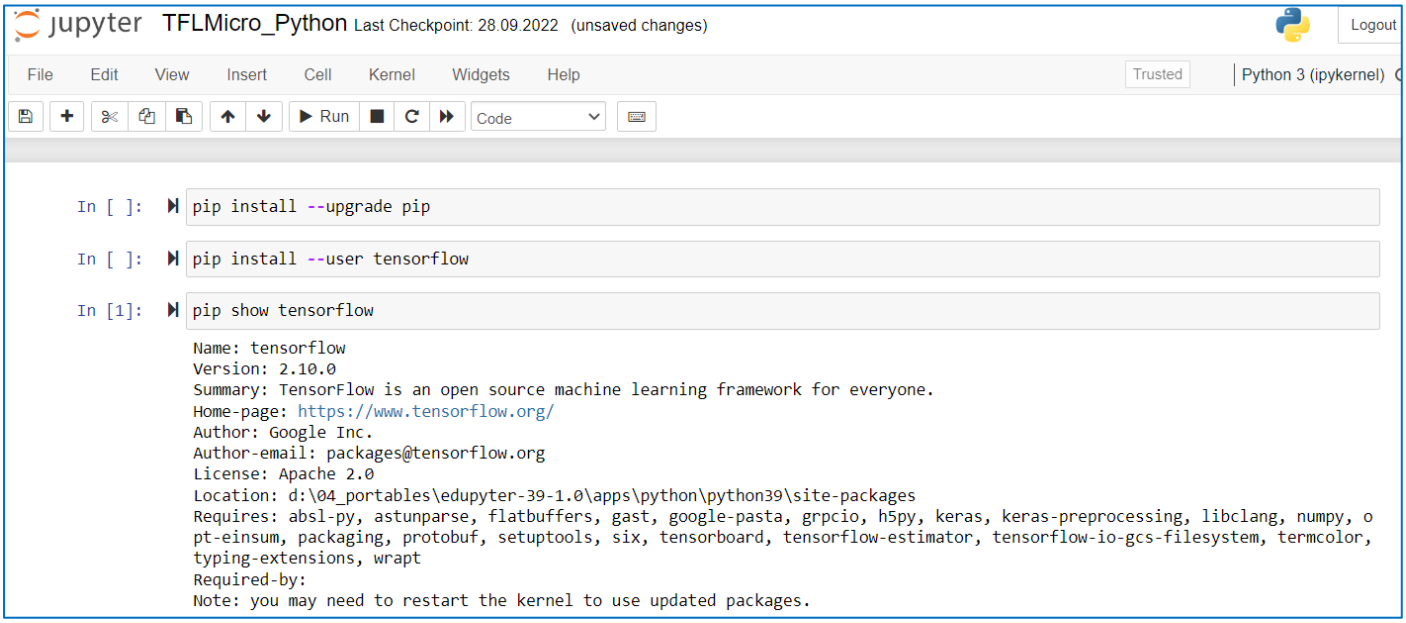


Abb. 22 Eigenes Screenshot Jupyter Notebook

1. Pip sollte sich immer auf dem aktuellen Stand befinden, deshalb zuallererst upgraden!

2. Tensorflow wird mit der „—user“-Option installiert, um keine Adminrechte zu benötigen wie im obigen Screenshot zu sehen, ist Keras bereits automatisch mitinstalliert.

3. Die neu installierte Tensorflow-Bibliothek sollte geprüft werden.

4. Danach den Kernel neu starten, vorher kann Tensorflow nicht geladen werden!

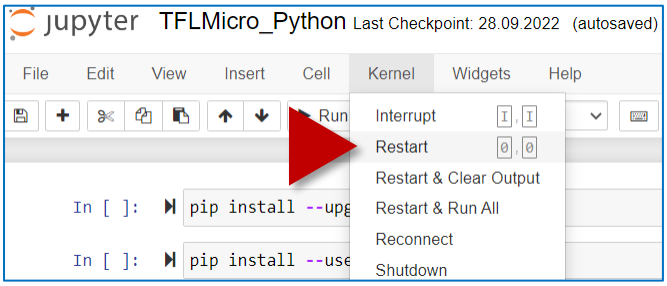


Abb. 23 Eigenes Screenshot Jupyter Notebook

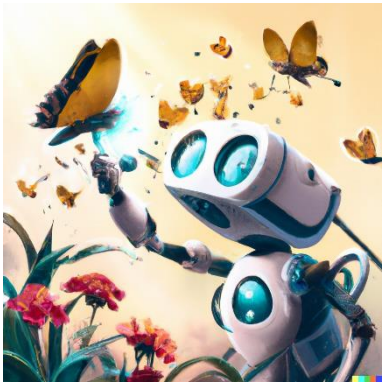


Abb. 24 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „a cute little robot capturing many butterflies, digital art“ von Jörg [CC BY-SA 4.0 International]

1. Datenaufnahme

1. Daten Aufnahme

Auf den BLE-Sense wird ein Bildaufnahmeskript aufgespielt, welches mit 5 Frames per Second Bilder zur Verfügung stellt und per Serieller Verbindung an den verbundenen PC schickt. Dort wird der serialisierte Datenstream wieder deserialisiert und zu einem Bild zusammengesetzt. Diese Aufgabe übernimmt ein Processing-Skript. Das übertragene Bild kann dann als „.png“ gespeichert werden.

Arduino Camera Capture:

Abgeleitet und stark vereinfacht von:

https://github.com/googlecreativelab/visual-alarm-clock/blob/main/Alarm/arduino_image_provider.cpp

3 Dateien miteinander in einem Ordner „captureImage“:

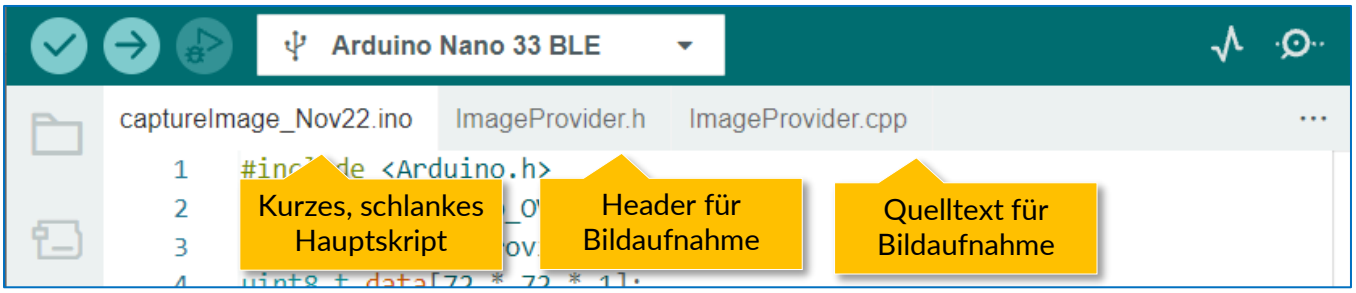


Abb. 25 Eigenes Screenshot Arduino IDE

Nützlich, um die Syntax von einigen Befehlen nachzuschauen:
https://github.com/arduino-libraries/Arduino_OV767X/blob/master/src/OV767X.cpp

CaptureImage_Nov22.ino

```
#include <Arduino.h>
#include <Arduino_OV767X.h>
#include "ImageProvider.h"

const int imgSize = 72;
```

Hier wird die Pixelauflösung gesetzt. Immer quadratische Bilder erzeugen!

```
const int kNumChannels = 1;
const int bytesPerFrame = imgSize * imgSize;
uint8_t data[imgSize * imgSize * kNumChannels];

void setup() {
   Serial.begin(115200);
}

void loop() {
  GetImage(imgSize, kNumChannels, data);
  Serial.write(data, bytesPerFrame);
}
```

ImageProvider.h

```
#ifndef IMAGE_PROVIDER_H_
#define IMAGE_PROVIDER_H_
#include <Arduino.h>
bool GetImage(int imgSize, int channels, uint8_t* image_data);
#endif
```

ImageProvider.cpp

```
#include "ImageProvider.h"
#include <Arduino.h>
#include <Arduino_OV767X.h>

byte captured_data[320 * 240 * 2];

boolean ProcessImage(int imgSize, uint8_t* image_data) {
  uint16_t color;
  for (int y = 0; y < imgSize; y++) {
    for (int x = 0; x < imgSize; x++) {
      int currX = floor(map(x, 0, imgSize, 100, 320-100));
      int currY = floor(map(y, 0, imgSize, 0, 240));

      int read_index = (currY * 320 + currX) * 2;
      uint8_t high_byte = captured_data[read_index];
      uint8_t low_byte = captured_data[read_index + 1];
      color = ((uint16_t)high_byte << 8) | low_byte;

      uint8_t r, g, b;
      r = ((color & 0xF800) >> 11) * 8;
      g = ((color & 0x07E0) >> 5) * 4;
      b = ((color & 0x001F) >> 0) * 8;
      float gray_value = (0.2126 * r) + (0.7152 * g) + (0.0722 * b);

      int index = y * imgSize + x;
      image_data[index] = static_cast<int8_t>(gray_value);
    }
  }
  return true;
}

boolean GetImage(int imgSize, int channels, uint8_t* image_data) {
  static bool g_is_camera_initialized = false;
  if (!g_is_camera_initialized) {
    if (!Camera.begin(QVGA, RGB565, 5)) {
      return false;
    }
    g_is_camera_initialized = true;
  }
  Camera.readFrame(captured_data);
  ProcessImage(imgSize, image_data);
}
```

Processing-Skript für die Datenaufnahme

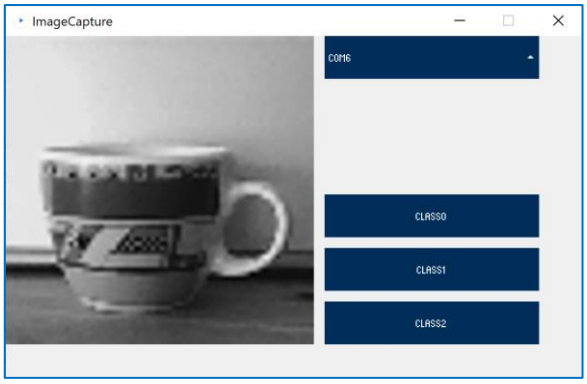


Abb. 26 Eigenes Screenshot ImageCapture

```
import processing.serial.*;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import controlP5.*;

Serial myPort;

final int imgSize = 72;
```

Hier wird die Pixelauflösung gesetzt. Wie im Arduino-Skript!

```
final int cameraBytesPerPixel = 1;
final int bytesPerFrame = imgSize * imgSize;
int counterA = 0;
int counterB = 0;
int counterC = 0;

PImage myImage;
byte[] frameBuffer = new byte[bytesPerFrame];
String[] portNames;
ControlP5 cp5;
ScrollableList portsList;

void setup() {
  size(540, 320);
  cp5 = new ControlP5(this);
  portNames = Serial.list();
  portNames = filteredPorts(portNames);
  portsList = cp5.addScrollableList("portSelect")
    .setPosition(4*imgSize + 10, 0)
    .setSize(200, 220)
    .setBarHeight(40)
    .setItemHeight(40)
    .addItems(portNames);
  portsList.close();

  cp5.addButton("Class0")
    .setPosition(4*imgSize + 10, 148)
    .setSize(200, 40)
    ;
  cp5.addButton("Class1")
    .setPosition(4*imgSize + 10, 198)
    .setSize(200, 40)
    ;
  cp5.addButton("Class2")
    .setPosition(4*imgSize + 10, 248)
    .setSize(200, 40)
    ;

  myImage = createImage(imgSize, imgSize, RGB);
}
void draw() {
  background(240);
  image(myImage, 0, 0, 4*imgSize, 4*imgSize);
}
void portSelect(int n) {
  String selectedPortName = (String) cp5.get(ScrollableList.class,
"portSelect").getItem(n).get("text");
  try {
    myPort = new Serial(this, selectedPortName, 115200);
```

```
    myPort.buffer(bytesPerFrame);
  }
  catch (Exception e) {
    println(e);
  }
}
String [] filteredPorts(String[] ports) {
  int n = 0;
  for (String portName : ports) n++;
  String[] retArray = new String[n];
  n = 0;
  for (String portName : ports) retArray[n++] = portName;
  return retArray;
}
void serialEvent(Serial myPort) {
  myPort.readBytes(frameBuffer);
  ByteBuffer bb = ByteBuffer.wrap(frameBuffer);
  bb.order(ByteOrder.BIG_ENDIAN);
  int i = 0;
  while (bb.hasRemaining()) {
    int r = (int) (bb.get() & 0xFF);
    myImage.pixels[i] = color(r, r, r);
    i++;
  }
  myImage.updatePixels();
  myPort.clear();
}
public void controlEvent(ControlEvent theEvent) {
  println(theEvent.getController().getName());
}
public void Class0() {
  String yourImageName = "pic"+counterA+"_0"; // Filename
  myImage.save("train/class0/" + yourImageName +".png");
  counterA++;
}
public void Class1() {
  String yourImageName = "pic"+counterB+"_1"; // Filename
  myImage.save("train/class1/" + yourImageName +".png");
  counterB++;
}
public void Class2() {
  String yourImageName = "pic"+counterC+"_2"; // Filename
  myImage.save("train/class2/" + yourImageName +".png");
  counterC++;
}
```

Abb. 27 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „a cute little robot sorting different vegetables, digital art“ von Jörg [CC BY-SA 4.0 International]

2. Datenaufbereitung

Die mit dem Processing-Script aufgenommenen Bilder werden im Ordner „train“ in den Unterordnern entsprechend der Klassen abgespeichert:

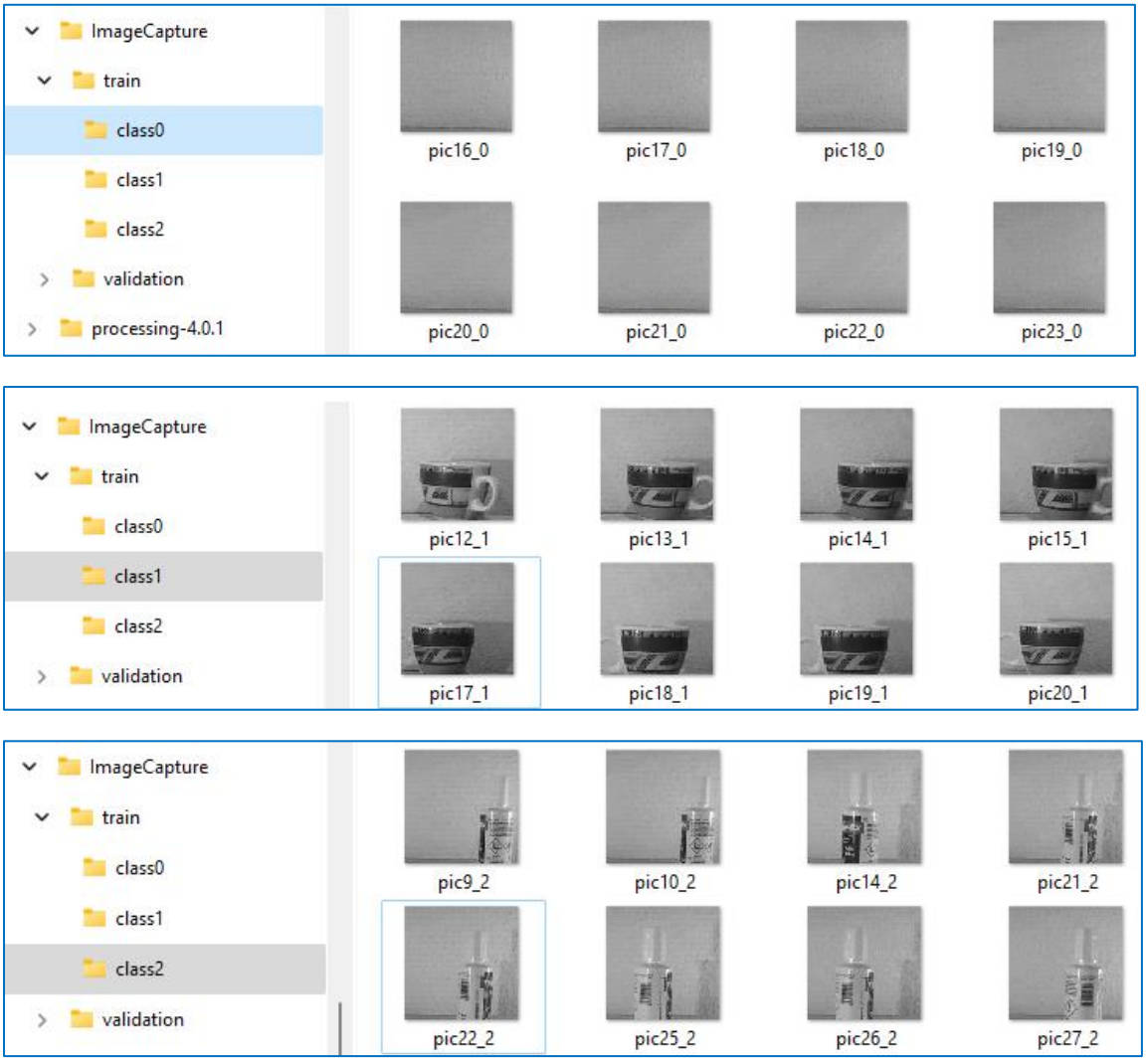


Abb. 28 Eigenes Screenshot Datensatz

Die Validierungsdateien müssen manuell aus den Trainingsdateien entnommen und in der gleichen Ordnerstruktur im Ordner „validation“ abgelegt werden.

3./4. Modellerstellung & Training

Abb. 29 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „a cute little robot staring at many different polaroid photos, cyberpunk style, digital art“ von Jörg [CC BY-SA 4.0 International]

3. Modell-Definition → 4. Model - Training

Das Training kann nicht auf dem Mikro-controller stattfinden, deswegen nutzt man auf dem PC die oben erwähnte Tensorflow-Installation. Alternativ kann man auch eine Google-Colab Entwicklungsumgebung nutzen.

TFLMicro_Python.ipyb

```
# Import aller Bibliotheken
import numpy as np
import matplotlib.pyplot as plt
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import preprocessing
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras.models import Sequential
print(tf.__version__)
```

```
# Trainings- und Validierungsdatensatz laden
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    'ImageCapture/train/',
    labels="inferred",
    label_mode="categorical",
    color_mode="grayscale",
    image_size=(72,72),
    batch_size=6
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    'ImageCapture/validation/',
    labels="inferred",
    label_mode="categorical",
    color_mode="grayscale",
    image_size=(72,72),
    batch_size=6
)
```

```
# Erzeugung des untrainierten Neuronalen Netzes, Variante nach Pete Wardens Blog
tf.keras.backend.clear_session()
class_names = ['nichts','tasse','box']

model = keras.models.Sequential()
model.add(layers.InputLayer(input_shape=(72, 72, 1)))
model.add(layers.Rescaling(1.0/255.0, offset=0.0)) # Normierung des Inputs
model.add(layers.Conv2D(filters = 2, kernel_size= (3,3), strides = (2,2), padding="same",
activation='relu'))
model.add(layers.Conv2D(filters = 2, kernel_size= (3,3), strides = (2,2), padding="same",
activation='relu' ))
model.add(layers.Flatten())
model.add(layers.Dropout(0.3))
model.add(layers.Dense(108, activation='relu'))
model.add(layers.Dense(3, activation='softmax'))
```

```
# Modell kompilieren
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
# Modell trainieren
epochs=70
history = model.fit(train_ds,validation_data=val_ds,epochs=epochs)
```

```
# Nachträgliche Diagrammdarstellung der Trainingsfortschritte
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

```
# Konvertierung in ein Tensorflow-Lite Modell
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
train_ds = train_ds.unbatch()

def representative_data_gen():
  for input_value, output_value in train_ds.batch(1).take(10):
    yield [input_value]

converter.representative_dataset = representative_data_gen
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8
tflite_model = converter.convert()

open("mymodel.tflite", "wb").write(tflite_model)
```

```
# Konvertieren in ein Tensorflow-Lite Micro Modell, Quantisierung
from tensorflow.lite.python.util import convert_bytes_to_c_source

source_text, header_text = convert_bytes_to_c_source(tflite_model,  "mymodel")
with  open('mymodel.h',  'w')  as  file:
    file.write(header_text)

with  open('mymodel.cpp',  'w')  as  file:
    file.write(source_text)
```

```
# Testphase - Prediction von einem Bild
img = keras.preprocessing.image.load_img("ImageCapture/pic0_2.png", target_size=(72,72,1),
color_mode="grayscale")
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)
score = predictions[0]
print(score[0], score[1], score[2])
```

```
# Nice to have: Heatmap-Darstellung eines Convolutional-Layers an einem Bild
import tensorflow.keras.backend as K

img = keras.preprocessing.image.load_img("ImageCapture/train/class1/pic10_1.png",
target_size=(72,72,1), color_mode="grayscale")
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)
score = predictions[0]
print(score[0], score[1], score[2])

with tf.GradientTape() as tape:
  last_conv_layer = model.get_layer('conv2d_1') ## Name aus model.summary()
  iterate = tf.keras.models.Model([model.inputs], [model.output, last_conv_layer.output])
  model_out, last_conv_layer = iterate(img_array) ## alle Layer zwischen Imput und
last_conv_layer durchgehen
  class_out = model_out[:, np.argmax(model_out[0])]
  grads = tape.gradient(class_out, last_conv_layer)

### Mittelwert des Tensors, entlang der definierten Achse.
pooled_grads = K.mean(grads, axis=(0))
heatmap = tf.reduce_mean(tf.multiply(pooled_grads, last_conv_layer), axis=-1)
### Normalisierung der Werte auf 0 bis 1
heatmap = np.maximum(heatmap, 0) / np.max(heatmap)
### Aus 1d macht 2d - auf die Größe des entspr. Layers
heatmap = heatmap.reshape((18, 18))
plt.matshow(heatmap)
plt.show()
```

Bemerkungen zum Aufbau des Neuronalen Netzes

Für die Bildbearbeitung ist ein Neuronales Netz das Modell der Wahl. Etwas spezifischer: Ein Convolutional Neural Network ist aktuell das beste Werkzeug, um solche Aufgaben zu lösen.

Da das Modell später auf dem Mikrocontroller gespeichert und für Inferenz genutzt wird, ist auf die Größe des Modells zu achten. Diese Größe lässt sich über die Anzahl der Parameter abschätzen, die beim Kompilieren des Models erzeugt werden. Erfahrungsgemäß soll diese im niedrigen 5-stelligen Bereich gehalten werden, damit das Modell klein genug für Arduino bleibt.

Dafür stehen folgende Optionen zu Verfügung, die unterschiedlich kombiniert werden können:

- Resizing einfügen – reduziert die Anzahl an Eingangsneuronen, hier auf die Hälfte)
- Strides auf (2,2) statt (1,1) setzen – Vergrößerung des Abstands bei der Faltung in einem Convolutional Layer)
- MaxPool2D Layer einfügen (reduziert die Größe des Bildes / des Layers auf die Hälfte bei (2,2))
- Neuronen im Dense-Layer (die doppelte Anzahl an Neuronen in diesem Layer bedeutet die Verdopplung der Parameter)

Standard-Aufbau eines CNN, mit dem man starten kann

```
# Klassischer Aufbau des untrainierten Neuronalen Netzes
tf.keras.backend.clear_session()
class_names = ['nichts','tasse','box']

model = keras.models.Sequential()
model.add(layers.InputLayer(input_shape=(72, 72, 1)))
model.add(layers.Resizing(36,36,interpolation='nearest'))
model.add(layers.Rescaling(1.0/255.0, offset=0.0))
model.add(layers.Conv2D(filters = 2, kernel_size= (3,3), strides = (1,1),
padding="same", activation='relu'))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Conv2D(filters = 2, kernel_size= (3,3), strides = (1,1),
padding="same", activation='relu'))
model.add(layers.MaxPool2D(2,2))
model.add(layers.Flatten())
model.add(layers.Dropout(0.3))
model.add(layers.Dense(108, activation='relu'))
model.add(layers.Dense(3, activation='softmax'))
```

Erläuterung der Parameter

„`input_shape=(72, 72, 1)`“

Das Input-Shape des Neuronalen Netzes entspricht der Pixelauflösung des Bildes: Jeder Pixel-Kanal des Bildes erhält ein Inputneuron. Im Falle eines Graustufenbildes mit der Auflösung 72px * 72px sind das 5184 Inputneuronen.

„`Resizing(36,36,interpolation='nearest')`“:

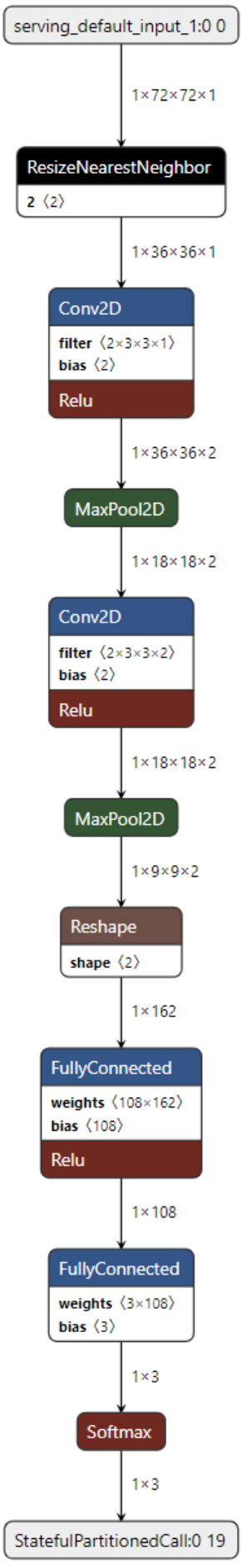
Um die Neuronenanzahl in den weiteren Layern zu reduzieren, wird das Bild herunterskaliert.

`layers.Rescaling(1.0/255.0, offset=0.0)`:

Die Inputwerte liegen im Bereich von 0 bis 255. Damit kann ein CNN nicht gut umgehen. Deshalb wird der Wertebereich normalisiert, auf einen Bereich zwischen 0 und 1. Dies muss vor dem eigentlichen Training geschehen.

`Conv2D(filters = 2, kernel_size= (3,3), strides = (1,1), padding="same", activation='relu')`:

Zu den 36*36 Bildwerten werden zwei Convolution-Filter angelegt. Diese haben die Eigenschaft, von jeweils 3*3 Bildwerten Convolutions zu berechnen. Dabei bewegt sich der Convolutionfilter sowohl in x- als auch in y-Richtung nach jeder Berechnung jeweils 1 Schritt weiter. Da es sich hierbei um trainierbare Parameter handelt, wird für die Backpropagation eine Aktivierungsfunktion bestimmt: die rectify linear unit (relu).



www.netron.app stellt das tflite-Modell dar.

Abb. 30 Darstellung eigenen Modells mit netron-App

`MaxPool2D(2,2):`
Nach jedem Convolutionschritt fasst man mittels Maxpooling 2*2=4 Bildwerte zusammen, indem von allen vier Bildwerten der jeweils höchstwertige Bildwert ‚überlebt'; die anderen drei Bildwerte werden verworfen.

`layers.Flatten():`
Die bisher entstandene 3-Dimensionalen Anordnung von Neuronen wird nun überführt in eine simple lineare Anordnung (Dense-Layer) eines Neuronalen Netzes.

`Dropout(0.3):`
Um Overfitting zu verhindern, werden pro Trainingszyklus ca. 30% der Verbindungen des neuronalen Netzes gelöscht.

`layers.Dense(108, activation='relu'):`
Der Hidden-Layer besitzt 108 Neuronen.

`layers.Dense(3, activation='softmax'):`
Das ist der Output-Layer. Er unterscheidet zwischen den drei unterschiedlichen Klassen „nichts", „Tasse" und „Box". Deshalb werden hier drei Outputneuronen verwendet – für jede Klasse eines – die jeweils mit der Softmax-Funktion aktiviert werden.

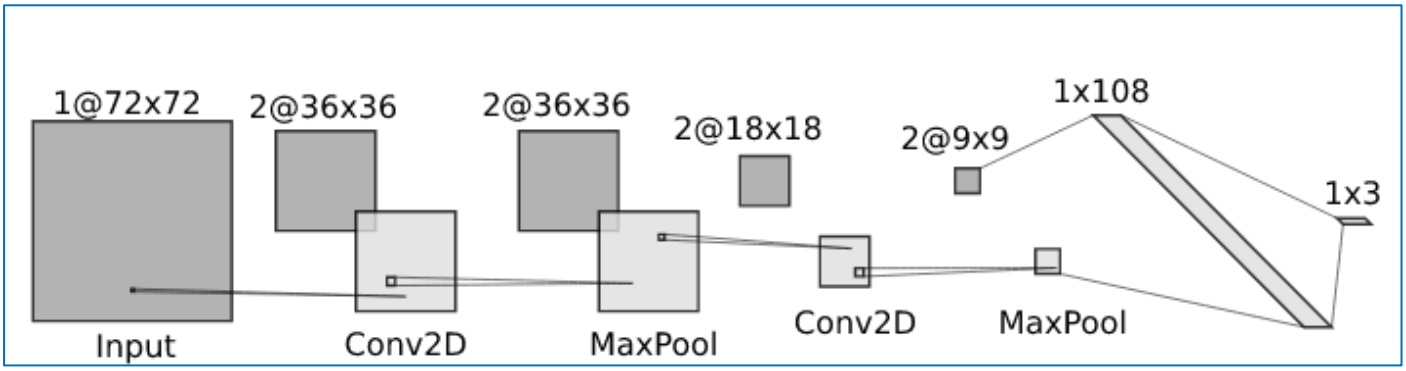


Abb. 31 Erstellt mit http://alexlenail.me/NN-SVG/LeNet.html

Wie werden die model.summary-Outputs berechnet:
https://towardsdatascience.com/how-to-calculate-the-number-of-parameters-in-keras-models-710683dae0ca

Bei der Modell-Erstellung ist darauf zu achten, dass die Daten auf signed 8-Bit (int8) quantisiert werden:
(https://www.tensorflow.org/lite/performance/quantization_spec)

Abb. 32 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „a cute little robot delivers many pizzas on a bicycle, digital art“ von Jörg [CC BY-SA 4.0 International]

5. Model-Deploy mit TFLM

Grundlegender Ablauf einer Tensorflow Lite Micro (TFLM) - Anwendung

TFLM läuft auf Geräten, die nur wenige Kilobyte Speicher besitzen. Deshalb wird keine Betriebssystem-Unterstützung benötigt. Es wird so viel Tensorflow-Funktionalität wie möglich genutzt. Der C-Code ist so geschrieben und konstruiert, dass der geringe Speicher effizient ausgenutzt wird.

Die folgenden Variablen werden benötigt:

1. Das **Modell**, in int8-quantisierter Form: das sind die Gewichtungen der Neuronen
2. Der **Error-Reporter**: Ensteht im Programmablauf irgendwo ein Fehler, so wird er hier registriert
3. Der **Input-Tensor**: Das ist hier das Datenarray des Graustufenbildes
4. Der **Interpreter**: Die Programmlogik, mit der das Neuronale Netz gesteuert wird
5. Der **Output-Tensor**: Das Datenarray, welches das Neuronale Netz ausgibt

Im Arduino-Setup finden sich die folgenden Funktionalitäten:

1. Die sensorspezifischen Funktionen, wie z.B. Initialisierungen etc.
2. Error-Reporter-Setup
3. Das Mapping des ML-Modells – also der Neuronalen Gewichtungen – mit GetModel()
4. Versionskontrolle: Zwischen verschiedenen Tensorflow-Versionen eventuelle Inkompatibilitäten?
5. Aufbau Op-Resolver: Die Operationen für die Layerarten wie z.B. Conv2D, MaxPool, Flatten …
6. Aufbau des Interpreters
7. Den Speicher für das Netz allozieren: Allocate_Tensors, alignas(8) etc.
8. Die Pointer für In- und Output-Arrays setzen

Im Arduino-Loop wird immer wieder ausgeführt:

1. Sensorspezifische Funktionen
2. Datenarray in den Input hineinladen
3. Inferenzieren, also die Daten mittels Interpreter durch das Neuronale Netz berechnen lassen
4. Den Output entgegennehmen und ausgeben.

Der Ops-Resolver

Abb. 33 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „a big toolbox with many tools, digital art“ von Jörg [CC BY-SA 4.0 International]

Baut man in Tensorflow ein Neuronales Netz auf – wie hier zum Beispiel den ‚Sequential'-Typ – dann benötigt man dazu unterschiedliche Layertypen und Layeroperationen. Dazu gehören Standard-Layer wie Dense, Flatten, Maxpooling, Resize, Conv2D usw.

Diese Typen und Operationen muss der Interpreter auf dem Mikrocontroller verstehen. Die notwendige Implementierung in TFLM ist der Op-Resolver. Derzeit sind, nach Stand 5. November 2022, insgesamt 97 (Siebenundneunzig) dieser Entitäten implementiert:

https://github.com/tensorflow/tflite-micro-arduino-examples/blob/main/src/tensorflow/lite/micro/all_ops_resolver.cpp

Demgegenüber stehen die Layer und Operationen in Tensorflow:

https://www.tensorflow.org/api_docs/python/tf/keras/layers

AllOpsResolver

Das Beispiel dieses Tutorials ist einfach gehalten: Es wird der sogenannte ‚AllOpsResolver' verwendet, der alle 97 verschiedenen Entitäten einbindet. Das benötigt allerdings viel Speicherplatz und sollte deshalb vermieden werden.

AllOps-Implementierung im Headerteil:

```
#include "tensorflow/lite/micro/all_ops_resolver.h"
```

AllOps-Implementierung in der Funktion “void setup”:

```
  static tflite::AllOpsResolver resolver;
  static tflite::MicroInterpreter static_interpreter(model, resolver, tensor_arena,
                                                     kTensorArenaSize, error_reporter);
```

Damit ist die Einbindung jeglicher Funktionalität vollständig und man muss sich nicht mehr weiter darum kümmern. Tritt jetzt allerdings über den Error-Reporter ein Fehler auf, dann weiß man automatisch, dass eine Inkompatibilität zwischen Tensorflow und TFLM besteht, und man muss das Tensorflow-Modell überarbeiten. Das ist oftmals der (frustrierende) Standardfehler zum Beispiel bei der Modellerstelltung auf dem ESP32.

MicroMutableOpsResolver

Um Speicherplatz zu sparen, wird von Google empfohlen, nur diejenigen Layer und Operationen zu implementieren, die auch tatsächlich benötigt werden. Dazu nutzt man den ‚MicroMutableOpResolver'

MicroMutableOps-Implementierung im Headerteil:

```
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
```

AllOps-Implementierung in der Funktion "void setup":

```
static tflite:: MicroMutableOpResolver<6> micro_op_resolver;
micro_op_resolver. AddResizeNearestNeighbor();
micro_op_resolver.AddMaxPool2D();
micro_op_resolver.AddConv2D();
micro_op_resolver.AddReshape();
micro_op_resolver.AddFullyConnected()
micro_op_resolver.AddSoftmax();

static tflite::MicroInterpreter static_interpreter(model, micro_op_resolver, tensor_arena,
                                                    kTensorArenaSize, error_reporter);
```

Erläuterung zu „<6>": Es werden insgesamt 6 verschiedene Operationen verwendet, deshalb wird diese Anzahl hier eingetragen.Weiterhin muss man aus dem Tensorflow-Pythonskript die unterschiedlichen Layertypen identifizieren und die zugehörigen Operationen in das Arduino-Skript einfügen:

`layers.InputLayer(…)`	*Benötigt keine eigene Funktionalität*
`layers.Resizing(interpolation='nearest')`	`micro_op_resolver. AddResizeNearestNeighbor()`
`layers.Rescaling()`	*Benötigt keine eigene Funktionalität*
`layers.MaxPool2D()`	`micro_op_resolver.AddMaxPool2D()`
`layers.Conv2D()`	`micro_op_resolver.AddConv2D()`
`layers.Flatten()`	`micro_op_resolver.AddReshape()`
`layers.Dense()`	`micro_op_resolver.AddFullyConnected()`
`layers.Dropout()`	
	`micro_op_resolver.AddSoftmax()` *(Weil der Output eine Softmax-A.F. besitzt)*

Tensorflow-Lite Micro Arduino-Projektdateien

Das gesamte Projekt besteht in diesem Fall aus 7 (sieben) Dateien, von denen zu jeder „.h“-Datei eine jeweilige „.cpp“-Datei gehört – also drei Paare von Dateien. Lediglich die „.ino“-Datei ist alleinstehend:

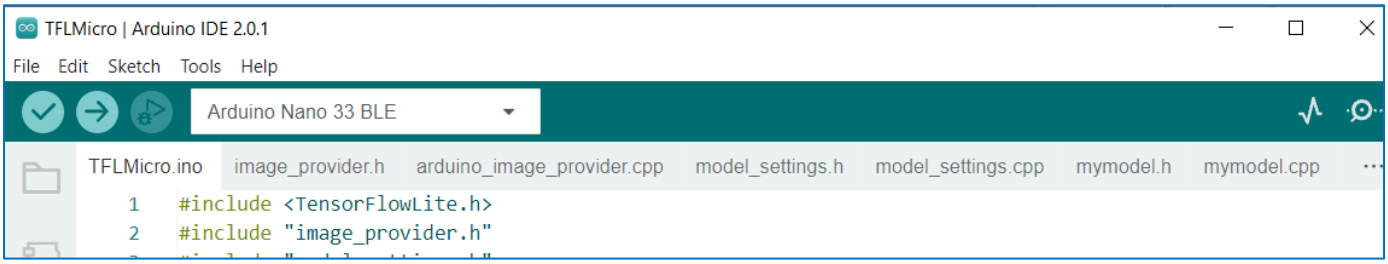


Abb. 34 Eigenes Screenshot Arduino IDE

Teil 1 von 4: TFLMicro.ino

Dies ist die Main-Datei, in der alle Header- und Librarydateien eingebunden werden:

```
#include <TensorFlowLite.h>
#include "image_provider.h"
#include "model_settings.h"
#include "mymodel.h"

#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/micro/system_setup.h"

namespace {
tflite::ErrorReporter* error_reporter = nullptr;
const tflite::Model* model = nullptr;
tflite::MicroInterpreter* interpreter = nullptr;
TfLiteTensor* input = nullptr;

constexpr int kTensorArenaSize = 22864;
uint8_t tensor_arena[kTensorArenaSize];
}

void setup() {
  Serial.begin(9600);
  tflite::InitializeTarget();

  static tflite::MicroErrorReporter micro_error_reporter;
  error_reporter = &micro_error_reporter;

  model = tflite::GetModel(mymodel_data);
  if (model->version() != TFLITE_SCHEMA_VERSION) {
    TF_LITE_REPORT_ERROR(error_reporter, "nope", model->version(), TFLITE_SCHEMA_VERSION);
    return;
```

```
  }
  static tflite::AllOpsResolver resolver;
  static tflite::MicroInterpreter static_interpreter(model, resolver, tensor_arena,
                                                     kTensorArenaSize, error_reporter);
  interpreter = &static_interpreter;
  TfLiteStatus allocate_status = interpreter->AllocateTensors();
  if (allocate_status != kTfLiteOk) {
    TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed");
    return;
  }
  input = interpreter->input(0);
}

void loop() {
  if (kTfLiteOk != GetImage(error_reporter,
                            kNumCols, kNumRows, kNumChannels, input->data.uint8)) {
    TF_LITE_REPORT_ERROR(error_reporter, "Image capture failed.");
  }

  if (kTfLiteOk != interpreter->Invoke()) {
    TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed.");
  }

  TfLiteTensor* output = interpreter->output(0);
  int8_t score0 = output->data.int8[0];
  int8_t score1 = output->data.int8[1];
  int8_t score2 = output->data.int8[2];
  Serial.print(score0);
  Serial.print("   ");
  Serial.print(score1);
  Serial.print("   ");
  Serial.println(score2);
}
```

Teil 2 von 4: ImageProvider

ImageProvider.h

```
#ifndef IMAGE_PROVIDER_H_
#define IMAGE_PROVIDER_H_

#include "tensorflow/lite/c/common.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
TfLiteStatus GetImage(tflite::ErrorReporter* error_reporter, int image_width,
                      int image_height, int channels, uint8_t* image_data);
#endif
```

ImageProvider.cpp

```
#include "image_provider.h"
#include <Arduino.h>
#include <Arduino_OV767X.h>

byte captured_data[320 * 240 * 2];

TfLiteStatus ProcessImage(tflite::ErrorReporter* error_reporter, int imgSize, uint8_t*
image_data) {
  uint16_t color;
  for (int y = 0; y < imgSize; y++) {
    for (int x = 0; x < imgSize; x++) {
      int currX = floor(map(x, 0, imgSize, 100, 320-100));
      int currY = floor(map(y, 0, imgSize, 0, 240));
      int read_index = (currY * 320 + currX) * 2;

      uint8_t high_byte = captured_data[read_index];
      uint8_t low_byte = captured_data[read_index + 1];
      color = ((uint16_t)high_byte << 8) | low_byte;
      uint8_t r, g, b;
      r = ((color & 0xF800) >> 11) * 8;
      g = ((color & 0x07E0) >> 5) * 4;
      b = ((color & 0x001F) >> 0) * 8;
      float gray_value = (0.2126 * r) + (0.7152 * g) + (0.0722 * b);

      int index = y * imgSize + x;
      image_data[index] = static_cast<int8_t>(gray_value-128);
    }
  }
  return kTfLiteOk;
}
TfLiteStatus GetImage(tflite::ErrorReporter* error_reporter, int image_width, int image_height,
int channels, uint8_t* image_data) {
  static bool g_is_camera_initialized = false;
  if (!g_is_camera_initialized) {
    if (!Camera.begin(QVGA, RGB565, 1)) {
      return kTfLiteError;
    }
    g_is_camera_initialized = true;
  }
  Camera.readFrame(captured_data);
  TfLiteStatus decode_status = ProcessImage(error_reporter, image_width, image_data);
  if (decode_status != kTfLiteOk) {
    TF_LITE_REPORT_ERROR(error_reporter, "DecodeAndProcessImage failed");
    return decode_status;
  }
  return kTfLiteOk;
}
```

Teil 3 von 4: ModelSettings

model_settings.h

```
#ifndef MODEL_SETTINGS_H_
#define MODEL_SETTINGS_H_

constexpr int kNumCols = 72;
constexpr int kNumRows = 72;
constexpr int kNumChannels = 1;

constexpr int kCategoryCount = 3;
extern const char* kCategoryLabels[kCategoryCount];
#endif
```

model_settings.cpp

```
#include "model_settings.h"
const char* kCategoryLabels[kCategoryCount] = {
    "nix", "tasse", "box",
};
```

Teil 4 von 4: Neuronales Netz-Modell

mymodel.h

```
#ifndef TENSORFLOW_LITE_UTIL_MYMODEL_DATA_H_
#define TENSORFLOW_LITE_UTIL_MYMODEL_DATA_H_

extern const unsigned char mymodel_data[];
extern const int mymodel_len;
#endif
```

mymodel.cpp

```
#include "mymodel.h"
alignas(8) const unsigned char mymodel_data[] = {
    0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00, 0x14,
    0x00, 0x20, 0x00, 0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c, 0x00,
    0x00, 0x00, 0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00,
...
};
const int mymodel_len = 22864;
```