Sistem automat de procesare de imagini pentru recunoașterea video a semnelor de circulație sau a culorilor de la semafor, folosind Raspberry PI și o cameră video atașată

Proiect - Sisteme Incorporate

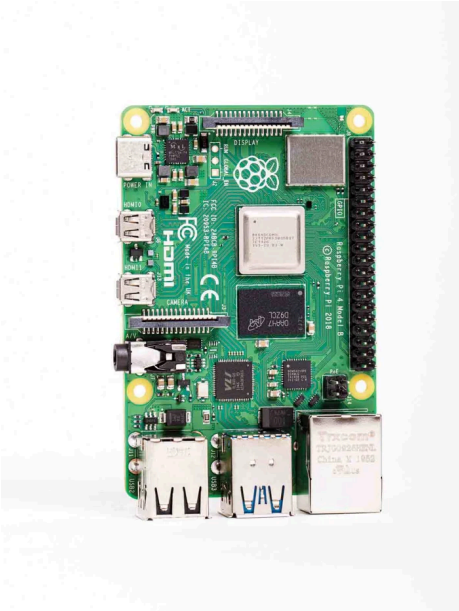
Hențiu Alexian-Tudor
Kondora Norbert
Vasile Ioana
2023-2024

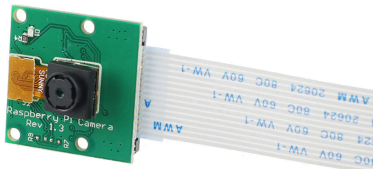
Cuprins

Desfășurarea lucrării

Echipamente utilizate:

- Raspberry Pi 4B
- Raspberry Pi Camera 1.3





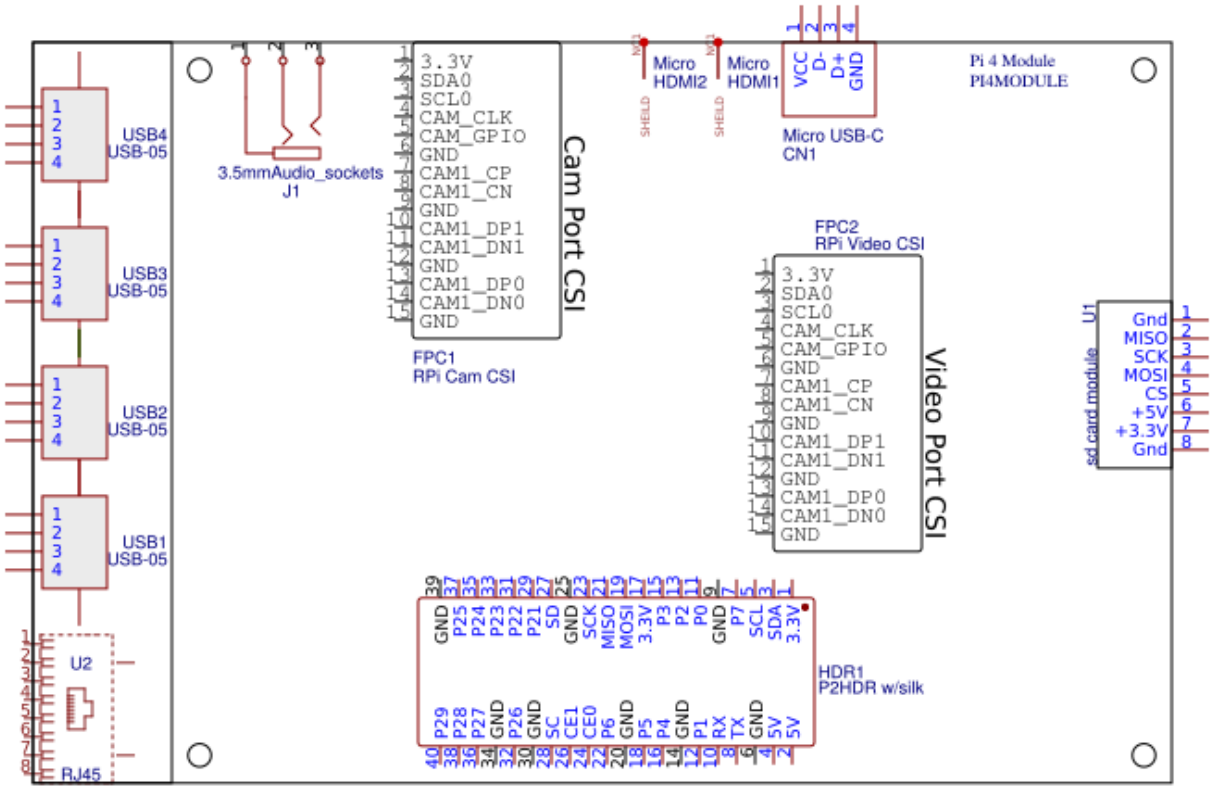
Mod de lucru:

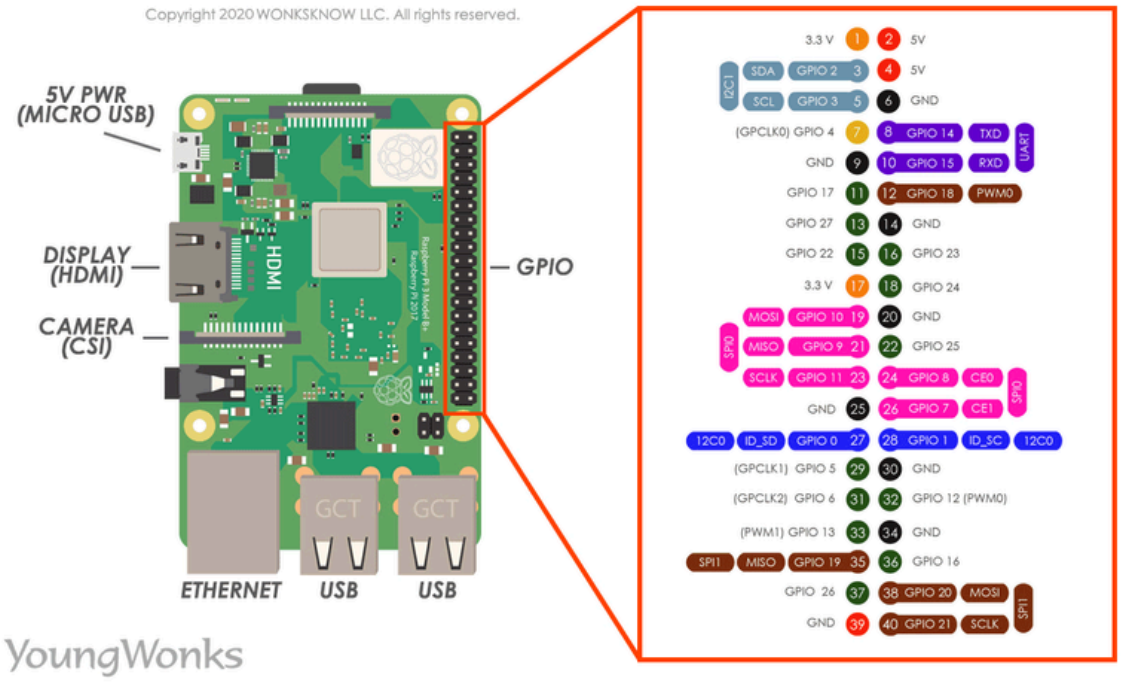
Pentru realizarea sistemului rutier de detecție, am antrenat doua modele Tensorflow, de tipul *ssd-mobilenet-v2-fpnlite-320*. Ambele modele au fost antrenate în Google Collab Pro, folosind GPU-ul T4. După antrenare, modele au fost convertite la Tensorflow Lite și cuantizate, pentru a putea rula cât mai eficient pe Raspberry Pi. Cu toate acestea, din cauza hardware-ului, performanța sistemului final este scăzută, detecția rulând cu 2-3 cadre pe secundă.

Primul model a fost antrenat pe un set de date realizat de noi, pentru 40.000 de iterații, în speranța că va fi capabil să identifice 49 de semne rutiere distincte. Însă, din cauza numărului redus de imagini din dataset (aproximativ 300), modelul este capabil să identifice doar semnele mai uzuale, precum: Limită de viteză, Stop, Cedează trecerea, Trecere de pietoni, Drum cu prioritate etc.

Cel de-al doilea model a fost antrenat pe un set de date preluat de pe internet, care conține aproximativ 900 de imagini, fiind capabil să distingă 4 clase diferite de obiecte: semne de stop, semne de limitare a vitezei, semne pentru trecere de pietoni și semafoare. Antrenarea acestui model a durat 20.000 de iterații.

Hardware


USB4
USB-05
USB3
USB-05
USB2
USB-05
USB1
USB-05
U2
RJ45
3.5mmAudio_sockets
J1
3.3V
SDA0
SCL0
CAM_CLK
CAM_GPIO
GND
CAM1_CP
CAM1_CN
GND
CAM1_DP1
CAM1_DN1
GND
CAM1_DP0
CAM1_DN0
GND
Cam Port CSI
FPC1
RPi Cam CSI
Micro
HDMI2
Micro
HDMI1
VCC
D-
D+
GND
Micro USB-C
CN1
Pi 4 Module
PI4MODULE
FPC2
RPi Video CSI
Video Port CSI
U1
sd card module
Gnd
MISO
SCK
MOSI
CS
+5V
+3.3V
Gnd
HDR1
P2HDR w/silk


Copyright 2020 WONKSKNOW LLC. All rights reserved.
5V PWR
(MICRO USB)
DISPLAY
(HDMI)
CAMERA
(CSI)
GPIO
ETHERNET
USB
USB
YoungWonks

Software

Script Python pentru detecție în timp real utilizând camera conectată la Raspberry Pi:

```
# Import packages
import os
import argparse
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util

# Define VideoStream class to handle streaming of video from webcam in separate processing
thread
# Source - Adrian Rosebrock, PyImageSearch:
https://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/
class VideoStream:
    """Camera object that controls video streaming from the Picamera"""
    def __init__(self,resolution=(640,480),framerate=30):
        # Initialize the PiCamera and the camera image stream
        self.stream = cv2.VideoCapture(0)
        ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
        ret = self.stream.set(3,resolution[0])
        ret = self.stream.set(4,resolution[1])

        # Read first frame from the stream
        (self.grabbed, self.frame) = self.stream.read()

       # Variable to control when the camera is stopped
        self.stopped = False

    def start(self):
       # Start the thread that reads frames from the video stream
        Thread(target=self.update,args=()).start()
        return self

    def update(self):
        # Keep looping indefinitely until the thread is stopped
        while True:
            # If the camera is stopped, stop the thread
            if self.stopped:
                # Close camera resources
                self.stream.release()
                return

            # Otherwise, grab the next frame from the stream
            (self.grabbed, self.frame) = self.stream.read()
```

```
    def read(self):
        # Return the most recent frame
        return self.frame

    def stop(self):
        # Indicate that the camera and thread should be stopped
        self.stopped = True

# Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    required=True)
parser.add_argument('--graph', help='Name of the .tflite file, if different than
detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than
labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying
detected objects',
                    default=0.5)
parser.add_argument('--resolution', help='Desired webcam resolution in WxH. If the webcam
does not support the resolution entered, errors may occur.',
                    default='1280x720')
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up
detection',
                    action='store_true')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
resW, resH = args.resolution.split('x')
imW, imH = int(resW), int(resH)
use_TPU = args.edgetpu

# Import TensorFlow libraries
# If tflite_runtime is installed, import interpreter from tflite_runtime, else import from
regular tensorflow
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate
```

```
# If using Edge TPU, assign filename for Edge TPU model
if use_TPU:
    # If user has specified the name of the .tflite file, use that name, otherwise use
default 'edgetpu.tflite'
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'edgetpu.tflite'

# Get path to current working directory
CWD_PATH = os.getcwd()

# Path to .tflite file, which contains the model that is used for object detection
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

# Load the label map
with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]

# Have to do a weird fix for label map if using the COCO "starter model" from
# https://www.tensorflow.org/lite/models/object_detection/overview
# First label is '???', which has to be removed.
if labels[0] == '???':
    del(labels[0])

# Load the Tensorflow Lite model.
# If using Edge TPU, use special load_delegate argument
if use_TPU:
    interpreter = Interpreter(model_path=PATH_TO_CKPT,
                              experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
    print(PATH_TO_CKPT)
else:
    interpreter = Interpreter(model_path=PATH_TO_CKPT)

interpreter.allocate_tensors()

# Get model details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

floating_model = (input_details[0]['dtype'] == np.float32)

input_mean = 127.5
input_std = 127.5

# Check output layer name to determine if this model was created with TF2 or TF1,
# because outputs are ordered differently for TF2 and TF1 models
outname = output_details[0]['name']
```

```
if ('StatefulPartitionedCall' in outname): # This is a TF2 model
    boxes_idx, classes_idx, scores_idx = 1, 3, 0
else: # This is a TF1 model
    boxes_idx, classes_idx, scores_idx = 0, 1, 2

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()

# Initialize video stream
videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
time.sleep(1)

#for frame1 in camera.capture_continuous(rawCapture, format="bgr",use_video_port=True):
while True:

    # Start timer (for calculating frame rate)
    t1 = cv2.getTickCount()

    # Grab frame from video stream
    frame1 = videostream.read()

    # Acquire frame and resize to expected shape [1xHxWx3]
    frame = frame1.copy()
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_resized = cv2.resize(frame_rgb, (width, height))
    input_data = np.expand_dims(frame_resized, axis=0)

    # Normalize pixel values if using a floating model (i.e. if model is non-quantized)
    if floating_model:
        input_data = (np.float32(input_data) - input_mean) / input_std

    # Perform the actual detection by running the model with the image as input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] # Bounding box
coordinates of detected objects
    classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0] # Class index
of detected objects
    scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] # Confidence of
detected objects

    # Loop over all detections and draw detection box if confidence is above minimum
threshold
    for i in range(len(scores)):
        if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):

            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image dimensions, need
to force them to be within image using max() and min()
```

```
            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))

            cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

            # Draw label
            object_name = labels[int(classes[i])] # Look up object name from "labels" array
using class index
            label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example: 'person: 72%'
            labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) #
Get font size
            label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too
close to top of window
            cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0],
label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in
            cv2.putText(frame, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 0), 2) # Draw label text

    # Draw framerate in corner of frame
    cv2.putText(frame,'FPS:
{0:.2f}'.format(frame_rate_calc),(30,50),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,0),2,cv2.LINE_A
A)

    # All the results have been drawn on the frame, so it's time to display it.
    cv2.imshow('Object detector', frame)

    # Calculate framerate
    t2 = cv2.getTickCount()
    time1 = (t2-t1)/freq
    frame_rate_calc= 1/time1

    # Press 'q' to quit
    if cv2.waitKey(1) == ord('q'):
        break

# Clean up
cv2.destroyAllWindows()
videostream.stop()
```

Bibliografie

1. Antrenare modele: https://colab.research.google.com/github/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/Train_TFLite2_Object_Detction_Model.ipynb#scrollTo=GSJ2wgGCixy2
2. Deployment pe RaspberryPi: https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/deploy_guides/Raspberry_Pi_Guide.md