Real-Time Object Detection with Intel NCS2 on Hardware with Limited Resources for Low-power IoT Devices

Jurij Kuzmic, Patrick Brinkmann and Günter Rudolph
Department of Computer Science, TU Dortmund University, Otto-Hahn-Str. 14, Dortmund, Germany

Keywords: Real-Time Object Detection, Convolutional Neural Network (Convnet), Autonomous Driving, Intel Neural Compute Stick 2 (NCS2), Computational Intelligence, Computer Vision, Tensorflow, Open Vino, YOLO.

Abstract: This paper presents several models for real-time object detection with a hardware extension on hardware with limited resources. Additionally, a comparison of two approaches for detecting individual objects with Single-Shot Multibox Detection (SSD) and You Only Look Once (YOLO) architecture in a 2D image with Convolution Neural Networks (ConvNet) is presented. Here, we focus on an approach to develop real-time object detection for hardware with limited resources in the field of the Internet of Things (IoT). Also, our selected models are trained and evaluated with real data from model making area. In the beginning, related work of this paper is discussed. As well known, a large amount of annotated training data for supervised learning of ConvNet is required. The data acquisition of the different real data sets is also discussed in this paper. Additionally, our dissimilar object detection models are compared in accuracy and run time to find the better and faster system for object detection on hardware with limited resources for low-power IoT devices. Through the experiments described in this paper, the comparison of the run time depending on different hardware is presented. Furthermore, the use of a hardware extension is analysed in this paper. For this purpose, we use the Intel Neural Compute Stick 2 (NCS2) to develop real-time object detection on hardware with limited resources. Finally, future research and work in this area are discussed.

1 INTRODUCTION

Nowadays it is possible to drive vehicles autonomously, without human intervention. These vehicles have a variety of sensors to interpret their environment and interact with it accordingly. For example, radar sensors can be used to determine the distance to the vehicle in front of these cars. But how does the vehicle behave if this system fails or provides incorrect distance measurements? Here, a control system that checks and compares the various measurements could be advantageous. This control system could be realized with distance measurements in a 2D image. For this purpose, the stereo camera can be used. This camera contains two cameras at a certain distance, similar to human eyes. This delivers two images. These both images can be used to determine the depth of the image to distinguish between roads, humans, cars, houses, etc. (Li, Chen and Shen, 2019). However, this various objects have to be recognized and classified. The autonomous vehicles have several cameras, including one that is directed at the road. With this camera, various objects, such as humans, vehicles or animals, can be delivered as a 2D image in the field of view of the camera on the road. These objects have to be recognized and classified in this 2D image. In addition, the position of an object on the motorway can be determined. Additionally, a lane detection (Kuzmic and Rudolph A1, 2021) have to be implemented. To evaluate this control system in real time a real-time object detection is a prerequisite. For this purpose, we focus in this paper on the real-time object detection of own objects on a hardware with limited resources for low-power IoT devices. Object detection requires usually hardware with high computational power, such as a graphics processing unit (GPU), because image processing is a highly intensive computing procedure. In addition, object detection is processed with Convolutional Neural Networks (ConvNets). These are known for their good processing of images. Also, the ConvNets have proven to be effective in object detection including contour finding. In this paper, we have trained and evaluated different models of object detection with different architectures such as Single-Shot Multibox

Kuzmic, J., Brinkmann, P. and Rudolph, G.
Real-Time Object Detection with Intel NCS2 on Hardware with Limited Resources for Low-power IoT Devices.
DOI: 10.5220/0010979900003194
In *Proceedings of the 7th International Conference on Internet of Things, Big Data and Security (IoTBDS 2022)*, pages 110-118
ISBN: 978-989-758-564-7; ISSN: 2184-4976

Detection (SSD) (Liu et al., 2016) and You Only Look Once (YOLO) (Redmon et al., 2016). The idea is to find a suitable system for model cars with non-high-computing hardware without a GPU. In addition, there are already low-cost hardware extensions to improve the run time of ConvNets on hardware with limited resources. Here, we use the Intel Neural Compute Stick 2 (NCS2) (CNET, 2018). This stick contains an Intel Movidius Myriad X Vision Processing Unit (VPU). The built-in microcontroller was specially developed for the use of Convolutional Neural Networks for applications with low-power consumption and real-time image processing (Intel, 2021).

Real-time object detection for low-power IoT hardware limited resources is not only interesting for the model making area. Additionally, this real-time object detection can be useful on hardware without internet connectivity. For example, consider postal drones that place the package in the garden or in smart surveillance cameras for agriculture which notify when certain objects e.g. animal species are detected. In addition, the route of an object can be tracked through a real-time evaluation, too. The problem in academic research is that these algorithms and procedures, which are already established in the autonomous vehicles industry, are kept under lock and key and are not freely accessible. For this reason, own algorithms and procedures have to be researched and developed in the academic field. Therefore, the topic of computer vision has become very popular in recent years.

The future goal of our work is to switch from the simulation we developed before (Kuzmic and Rudolph, 2020) to the real model cars. In case of a successful transfer of simulation to reality (sim-to-real transfer), the model car behaves exactly as before in the simulation. Here, the hardware of these model cars belongs to the low-power IoT devices with limited resources.

2 RELATED WORK

There are numerous scientific papers dealing with object detection, e.g. (Fink et al., 2019) who have made a deep learning-based multi-scale multi-object detection and classification for autonomous driving or (Zaghari et al., 2021) who have developed improvement in obstacle detection in autonomous vehicles using YOLO non-maximum suppression fuzzy algorithm. However, there are only a few scientific papers that are dealing with the detection of the objects in real time on hardware with limited resources for low-power IoT devices. For example, (Wang, Li and Ling, 2018) who have developed pelee: a real-time object detection system on mobile devices. This system reaches 23.6 FPS on an iPhone 8. In their work, the SSD and YOLO architectures were also analysed. However, the hardware of the iPhone 8 is quite expensive to use it, for example, in a model car or as a surveillance camera. Similarly, there are (Jose et al., 2019) who have researched real-time object detection on low power embedded platforms. This system operates at 22 FPS on low-power TDA2PX System on Chip (SoC) provided by Texas Instruments (TI). Similarly, there are scientific works that deal with YOLO real-time object detection for low-power hardware, such as (Huang, Pedoeem and Chen, 2018) who have developed YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers or (Jin, Wen and Liang, 2020) who implemented embedded real-time pedestrian detection system using YOLO optimized by LNN. These papers handle with an older YOLOv3 version. Furthermore, some scientific papers are also dealing with the Intel Neural Compute Stick 2 such as (Asmara et al., 2020) who developed prediction of traffic density using YOLO object detection and implemented in Raspberry Pi 3b + and Intel NCS 2.

Our approach is to develop a real-time object detection for low-power IoT hardware and to expand our previous development in this area (Kuzmic and Rudolph A2, 2021). Thus, it is possible to develop a low-cost real-time object detection e.g. for model making or a surveillance camera in a short time. For this purpose, we use the Raspberry Pi 3 B and Raspberry Pi 4 B with the Intel NCS2 as hardware extension.

3 DATA SET

Before training of the ConvNets, annotated training data have to be obtained for each specific use case. This training data is the basis for a successful object detection. The resolution is given in the format width x height. Some small pre-tests have shown: it is sufficient to take pictures of the object in a 360° view. The colour of the objects does not matter in object detection. The objects are distinguished by their different shapes. Our data sets were created with some objects from the model making area. We annotated this data manually. The procedure for this is described in subsection 3.2. Data set 1 contains a model car *PiCar* and data set 2 additionally *ModCar, ModAnimal* and *ModPerson* from the real world. So in data set 1 we have one class with 111 pictures. In data set 2 there are four classes with a total of 200 pictures. Figure 1 shows some images of the created training data from our data sets. In addition, the following can be seen on Figure 1, first two rows: Data set 1 contains a model car *PiCar* with various objects. Here, only the model car is labelled and not the other objects. So, the ConvNet can learn the difference to the other objects. Also, small pre-tests have shown that the ConvNet learns the differences between the various contours of objects. It is not enough to train the ConvNet, for example, with the model car on a white background. Any further unknown object would be interpreted as a model car at this point. The human brain, on the other hand, would learn it.

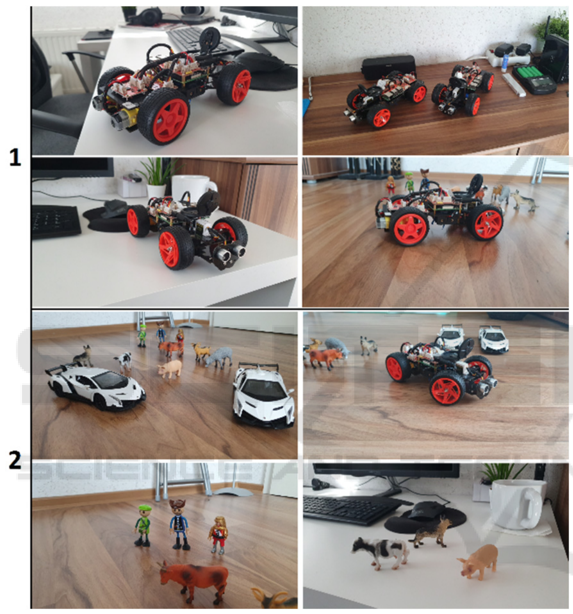

Figure 1: Our data sets for object detection. First two rows: Data set 1 from model making area with one class (*PiCar*). Last two rows: Data set 2 from model making area with four classes (*PiCar, ModCar, ModAnimal, ModPerson*).

If object detection on real data has to be implemented, already published data sets Microsoft Common Objects in Context (MS COCO) (Lin et al., 2014) or PASCAL Visual Object Classes (PASCAL VOC) (Everingham et al., 2010) can be used. These already contain thousands of annotated real objects. The split of our training and test data is 80/20. The test data was used as validation data.

3.1 Automatic Annotation

It is possible to annotate the objects to be classified manually. Unfortunately, this method is very time-consuming for large data sets. With the approach of contour finding in the HSV colour space, the coordinates of the objects can be found automatically. A distinction is made between the hue, the colour saturation and the colour value (brightness) (Ivanov and Skryshevsky, 2021). To determine the position of the object in a 2D image, the input image is converted from RGB format (Fig. 2, left) into HSV format first (OpenCV, 2021). Figure 2, middle shows this input image in a HSV format. Then, the HSV parameters for the contour finding and the binarization have to be found (Shermal, 2017). In the next step, a binary image can be created (Fig. 2, right, without the green rectangle). These parameters have only to be found once per object class. From this binary image (white background, black object), the information for the position of the object (top left and bottom right) could be extracted. This gives the position of the object (Fig. 2, right) for the input image as coordinates for P (xMin, yMin) and Q (xMax, yMax).

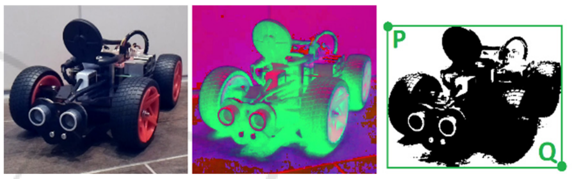

Figure 2: Automatic annotation with the HSV colour space. Left: Model car in RGB format. Middle: Model car in HSV format. Right: Model car as binary image with founded coordinates for P and Q.

Once these coordinates are available, the annotations can be created in a TensorFlow (Vuppala, 2020) or a YOLO format (Bochkovskiy, Wang and Liao 2020). The advantage of this approach: many annotated training data with different objects can be created in a short time. Several objects can also be created in one image. Since, the position of the objects is known, these objects can be moved or exchanged among each other. Additionally, to create many different training data the position and size of the objects can be changed. The exchange of the background image is also conceivable. In our case, there was not much training data. For this reason, we manually annotated our training data sets. This procedure is described in the following section 3.2. Nevertheless, we tried and tested the automatic annotation of the data sets afterwards.

3.2 Manual Annotation

For our data set, some pictures from several models from model making were taken (Kuzmic and Rudolph A2, 2021). Afterwards, these images were manually annotated with the LabelImg tool (Tzutalin, 2015).

This tool simplifies the drawing of the rectangle around an object and automatically determines the coordinates for P (xMin, yMin) and Q (xMax, yMax). Then, these coordinates can be stored in an XML file with the same name as the image file for training with TensorFlow as annotation. With this tool it is also possible to save the annotations in a YOLO-format.

4 EXPERIMENTS

The following experiments were carried out to compare the functionality and the run time of the different TensorFlow and YOLO models. The resolution is in the format width x height. The accuracy is given as COCO mean average precision (mAP) metric (Hui, 2018). All experiments are carried out on the same hardware. This gives the possibility to compare the results afterwards and to find the optimal object detection system. The test input images for the respective systems are also the same. For hardware with limited resources, a single-board Raspberry Pi 3 B and Raspberry Pi 4 B were used. The run time of detection is shown as frames per second (FPS). These measurements contain only the time for object detection and do not include loading and processing of the input images. Our Hardware for the experiments:

- Training of the ConvNets on Google Colab with Intel Xeon 2.30 GHz CPU, 26 GB RAM, NVIDIA Tesla P100-PCIE-16GB GPU.
- Raspberry Pi 3 B with ARM Cortex-A53 1.2 GHz CPU, 1 GB RAM, USB 2.0, 8 GB SD as hardware with limited resources.
- Raspberry Pi 4 B with ARM Cortex-A72 1.5 GHz CPU, 8 GB RAM, USB 3.0, 16 GB SD as hardware with limited resources.
- Intel Neural Compute Stick 2 (NCS2) with Intel Movidius Myriad X Vision Processing Unit 4 GB (VPU) as hardware extension for artificial neural networks.

4.1 Object Detection with SSD and Intel NCS2

In these experiments the models we have trained with TensorFlow are evaluated on the hardware with limited resources. The first step was to find some fast TensorFlow models. For TensorFlow some pre-trained models are available. These models have been already established in object detection with real data. In the TensorFlow 1 Detection Model Zoo (Shi, 2020), several ConvNet models are available for TensorFlow 1. These have already been pre-trained on the MS COCO 17 data set (Lin et al., 2014) and can detect and classify 90 different objects. The accuracy and the run time of these models is also specified. This gives first comparison of the ConvNet models. Furthermore, these models can be trained with several objects with the *TensorFlow Object Detection API* (Yu et al., 2020) in the next step. This library supports TensorFlow 1 and TensorFlow 2. The input resolution of these networks can also be adjusted. To evaluate these models on the Raspberry Pi and the Intel NCS2 for several objects, the models are first trained with *TensorFlow 1.15.5* with our training data. Subsequently, the trained models have to be exported as a frozen inference graph. Then, these models can be converted to OpenVINO models using the *OpenVINO 2021.3* library (OpenVINO, 2021) to run them on the Intel Neural Compute Stick 2. *Python 3.7.3* was used to execute the converted OpenVINO models on the hardware with limited resources. The *SSD MobileNet V2* and *SSD Lite MobileNet V2* models were used in our experiments.

Table 1: Run time overview of trained TensorFlow models on Raspberry Pi 3 B with Intel NCS2 and Raspberry Pi 4 B with Intel NCS2. First column contains the number (ID) of the experiment (Exp. No.).

Exp. No.	**Model**	**Data Set**	**mAP**	**Run time [FPS]**	
				RPI 3 B	**RPI 4 B**
1	SSD MobileNet V2 224x224	Mod 1	93.3	17.4	31.9
2	SSD MobileNet V2 224x224	Mod 2	80.8	17.2	31.7
3	SSD MobileNet V2 320x320	Mod 1	96.7	9.8	18.0
4	SSD MobileNet V2 320x320	Mod 2	88.5	9.7	17.9
5	SSD Lite MobileNet V2 224x224	Mod 1	99.9	16.5	29.5
6	SSD Lite MobileNet V2 224x224	Mod 2	83.3	16.3	29.1
7	SSD Lite MobileNet V2 320x320	Mod 1	99.8	9.4	16.7
8	SSD Lite MobileNet V2 320x320	Mod 2	93.6	9.3	16.7

It is interesting to note that the *SSD MobileNet V2* and *SSD Lite MobileNet V2* models are not listed as supported network architecture on the official Intel Movidius page (Movidius, 2019). At the time of our experiments and implementation, only the *SSD MobileNet V1* models and the *Inception* models are listed as supported networks on the official Intel Movidius page. Nevertheless, for the development of our real-time object detection system we decided to use the current *SSD MobileNet V2* and *SSD Lite MobileNet V2* models and to run them with OpenVINO on the Intel NCS2. According to the TensorFlow 1 Detection Model Zoo (Shi, 2020), the *SSD MobileNet V2* models are partly faster and more accurate than the *SSD MobileNet V1* models. Table 1 shows an overview of the accuracy and run time of the trained SSD models on the Raspberry Pi 3 B with Intel NCS2 and Raspberry Pi 4 B with Intel NCS2. Also, *SSD Lite MobileNet V2* models with one class as output can quickly overtrain (exp. no. 5 in table 1). Additionally, we can see that the run time on the Raspberry Pi 4 B with Intel NCS2 is almost twice as high (exp. no. 1 in table 1). Our assumption at this point: The Raspberry Pi 3 B only has a USB 2.0 interface. The Raspberry Pi 4 B, on the other hand, has a USB 3.0 interface. The Intel NCS2 hardware extension is also use a USB 3.0 interface. Thus, this is the bottleneck in terms of the run time. To check this, the *SSD MobileNet V2* 224x224 model was also run on the Raspberry Pi 4 B with the Intel NCS2 on the USB 2.0 interface. Table 2 shows that our assumption is correct. The run time bottleneck on the Raspberry Pi 3 B with the Intel NCS2 is the USB 2.0 interface (comparison between exp. no. 1 in table 1 and exp. no. 1 in table 2).

Table 2: Run time overview of USB 2.0 and USB 3.0 on Raspberry Pi 4 B with Intel NCS2. First column contains the number (ID) of the experiment (Exp. No.).

Exp. No.	**Model**	**Data Set**	**mAP**	**Run time [FPS]**	
				USB 2.0	**USB 3.0**
1	SSD MobileNet V2 224x224	Mod 1	93.3	18.9	31.9

Let now have a look to the evaluation of the test data from the worst SSD model (Fig. 3). This model achieves an accuracy of 80.8 % (exp. no. 2 in table 1). In Figure 3 can be seen that the detection of four classes *PiCar, ModCar, ModAnimal* and *ModPerson* is very accurate. Also, the classification and the position of the detected objects matches exactly (green rectangles). That is, the predicted object boundary overlaps exactly with the real object boundary. This detection is completely sufficient for our purpose. Also, the never seen objects could be detected by our trained SSD model. For example, a Lego person (Fig. 4, left), a model horse (Fig. 4, middle) and a blue model car (Fig. 4, right) are correctly recognized and classified by this model. During training the ConvNet has already seen figures of model persons, model animals and model cars. This ConvNet learned the similar shapes of the objects and not only images from the training data set. Furthermore, every other object with an unknown shape is recognized as a *PiCar*. The models have never seen any object with a similar shape (contour) during the training. Thus, the object cannot be clearly classified. As a result, the first output class *PiCar* is assigned to this unknown object. This problem can be solved by enlarging the data set. More training data with the *PiCar* and many different objects (different shapes) is needed. So, the difference to other shapes can be learned.

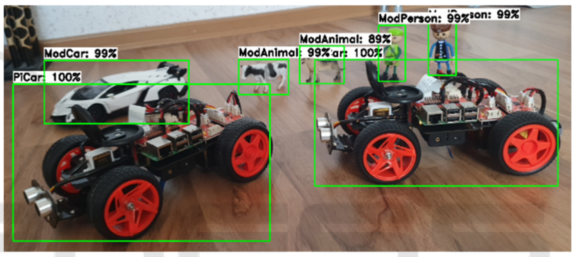

Figure 3: Object detection with *SSD MobileNet V2* 224x224 model trained with TensorFlow (exp. no. 2 in table 1). Green rectangle shows the detection of the object by the ConvNet. This model contains four classes as output (*PiCar, ModCar, ModAnimal, ModPerson*).

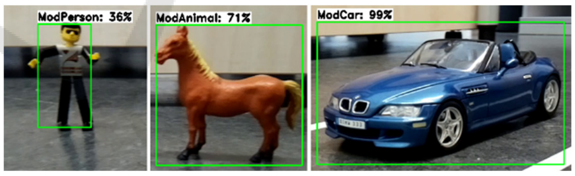

Figure 4: Object detection on never seen objects with *SSD MobileNet V2* 224x224 model trained with TensorFlow (exp. no. 2 in table 1). Green rectangle shows the detection of the object by the ConvNet. This model contains four classes as output (*PiCar, ModCar, ModAnimal, ModPerson*). Left: Detected model person. Middle: Detected model animal. Right: Detected model car.

4.2 Object Detection with YOLO and Intel NCS2

To implement a comparable system for object detection, some current *YOLO V4* models (Bochkovskiy et al., 2020) were trained and evaluated using the same data sets and the DarkNet framework (version: *CSPDarknet53*). For this purpose, the empty weights for *YOLO V4* (yolov4.conv.137) and *YOLO V4 Tiny* (yolov4-tiny.conv.29) were used (Bochkovskiy et al., 2020). Then, the trained weights of the models were converted to Protobuf models using *TensorFlow 1.12.0* and *Protobuf 3.15.6* (Tianwen, 2021). *Python 3.6.13* was used for this conversion. Notice, a conversion using *Python 3.7.3* did not work at the time of the experiments. Subsequently, the converted Protobuf models can be converted to OpenVINO using the *OpenVINO 2020.4* library and YOLO configuration files (Tianwen, 2021) in the next step. After the conversion to OpenVINO, several frozen DarkNet *YOLO V4* and *YOLO V4 Tiny* models are obtained to run them on the Intel Neural Compute Stick 2 hardware extension with *Python 3.7.3* (similar procedure to TensorFlow). Table 3 shows an overview of the accuracy and run time of the trained YOLO models on the Raspberry Pi 3 B with the Intel NCS2 and Raspberry Pi 4 B with the Intel NCS2.

Table 3: Run time overview of trained YOLO models on Raspberry Pi 3 B with Intel NCS2 and Raspberry Pi 4 B with Intel NCS2. First column contains the number (ID) of the experiment (Exp. No.).

Exp. No.	**Model**	**Data Set**	**mAP**	**Run time [FPS]**	
				RPI 3 B	**RPI 4 B**
1	YOLO V4 Tiny 224x224	Mod 1	93.3	9.9	15.5
2	YOLO V4 Tiny 224x224	Mod 2	89.5	9.8	15.4
3	YOLO V4 Tiny 320x320	Mod 1	98.6	5.3	8.5
4	YOLO V4 Tiny 320x320	Mod 2	94.5	5.2	8.3
5	YOLO V4 224x224	Mod 1	100	1.7	2.2
6	YOLO V4 224x224	Mod 2	99.2	1.6	2.2
7	YOLO V4 320x320	Mod 1	100	0.9	1.7
8	YOLO V4 320x320	Mod 2	98.8	0.9	1.7

It can be seen that the *YOLO V4* models with one class as output can quickly overtrain. Let now have also a look to the evaluation of the object detection on the same test image from the worst *YOLO V4 Tiny* model (Fig. 5). This model achieves an accuracy of 89.5 % (exp. no. 2 in table 3). As also can be seen in figure 5 the classification of the objects corresponds. However, the position of the detected objects does not match exactly (for example white model car in figure 5). That is, the predicted object boundary does not overlap exactly with the real object boundary. Also, the never seen objects could be detected by our trained YOLO model. For example, a Lego person (Fig. 6, left), a model horse (Fig. 6, middle) and a blue model car (Fig. 6, right) are correctly classified by this model. However, the position of the detected objects (green rectangles) does not match. Also, some objects are recognized twice (Fig. 6, middle. Therefore, the object detection with the YOLO architecture is not suitable for our use case.

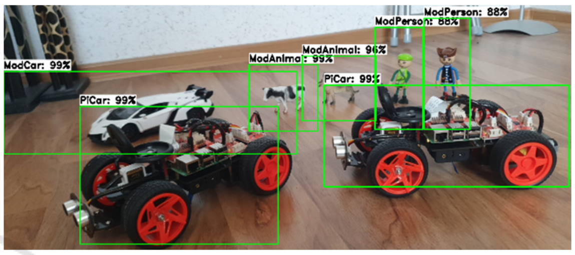

Figure 5: Object detection with *YOLO V4 Tiny* 224x224 model trained with DarkNet (exp. no. 2 in table 3). Green rectangle shows the detection of the object by the ConvNet. This model contains four classes as output (*PiCar, ModCar, ModAnimal, ModPerson*).

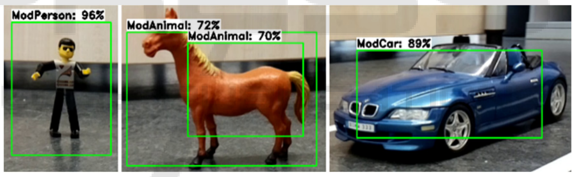

Figure 6: Object detection on never seen Objects with *YOLO V4 Tiny* 224x224 model trained with DarkNet (exp. no. 2 in table 3). Green rectangle shows the detection of the object by the ConvNet. This model contains four classes as output (*PiCar, ModCar, ModAnimal, ModPerson*). Left: Detected model person. Middle: Detected model animal. Right: Detected model car.

4.3 Evaluation of Run Time and Accuracy

After the experiments and performance tests of the models have been completed, evaluating of the run times of these different models could be started. Therefore, it is important to find a balance between sufficient accuracy and the run time of the models. Also, some experiments show longer training does not affect the run time. The run time depends only on the model architecture, on size of the input resolution and on size of the output classes of the ConvNet. With *YOLO V4* and *YOLO V4 Tiny* models, only a maximum of about 16 FPS could be achieved in our implementation. Furthermore, with *YOLO V4 Tiny* with 224x224 pixel resolution the size of the detected objects is partly incorrect (Fig. 5). For a distance measurement to a detected object in a 2D image, for example, the detected position and size of the objects have to match exactly. This is the case with the *SSD MobileNet V2* model with 224x224 pixel resolution (Fig. 3). A comparison between these two experiments (comparison between exp. no. 2 in table 1 and exp. no 2 in table 3) shows that it is not sufficient to select the models based on the COCO mAP (Hui, 2018). The *YOLO V4 Tiny* 224x224 model achieves a mAP of 89.5 % (exp. no. 2 in table 3) while the *SSD MobileNet V2* 224x224 model only achieves a mAP of 80.8 % (exp. no. 2 in table 1). However, if we examine the evaluated test images, we can see that the detection of the *SSD MobileNet V2* 224x224 model is much more accurate in spite of the smaller mAP. Thus, the results should be evaluated and examined for each specific use case.

As a standard, videos with a lot of motion are recorded at 30 FPS (Kuzmic and Rudolph A1, 2021). To analyse these videos and detect objects in real time, the two models *SSD MobileNet V2* and *SSD Lite MobileNet V2* trained with TensorFlow with a resolution of 224x224 pixels are suitable for a real-time application on the Raspberry Pi 4 B with Intel NCS2 (exp. no. 1, 2, 5 and 6 in table 1). Here, up to approx. 32 FPS could be achieved.

The SSD models with 320x320 pixel resolution, on the other hand, are slightly more accurate than the 224x224 pixel models. However, they are also slightly worse in terms of the run time (comparison between exp. no. 6 and 8 in table 1). These models achieve up to approx. 17 FPS. This approach does not achieve a real-time evaluation on our hardware.

5 CONCLUSIONS

This section summarizes once again the points that were introduced in this paper. In our research, we focused on real-time object detection for custom objects. The usage on hardware with limited resources for low-power IoT devices was our first priority. For this purpose, we also created our own data sets from the model making area. In addition, several different SSD and YOLO models were trained to find a balance between sufficient accuracy and the run time of the models. The dissimilar approaches allowed us to create a comparison of methods between SSD and YOLO on the Raspberry Pi 3 B and Raspberry Pi 4 B with Intel Neural Compute Stick 2. According to our experiments, the *SSD MobileNet V2* and *SSD Lite MobileNet V2* models trained with TensorFlow with 224x224 pixel resolution are suitable for real-time object detection on the Raspberry Pi 4 B with the Intel Neural Compute Stick 2 as a hardware extension. Here, up to approx. 32 FPS could be achieved with described hardware and SSD models. This is completely sufficient for a real-time application. As our experiments also show, the YOLO models trained with the DarkNet framework are significantly slower than the SSD models trained with TensorFlow on the Raspberry Pi 3 B and Raspberry Pi 4 B. These models are not suitable for real-time object detection on hardware with limited resources for low-power IoT devices.

In conclusion, we presented the *SSD MobileNet V2* and *SSD Lite MobileNet V2* models with 224x224 pixel resolution for real-time object detection with Raspberry Pi 4 and the Intel Neural Compute Stick 2 (NCS2). These models achieve an effective balance between accuracy and run time.

6 FUTURE WORK

As already announced, the goal of our future work is to successfully conduct a sim-to-real transfer, including our lane and object detection we have developed for the model making area. This means the simulated environment is completely applied to a real model vehicle. In this approach, we focus on developing software for hardware with limited resources for low-power IoT devices. Additionally, we want to set up a model test track like a real motorway for this experiment. Another important aspect on the motorways is the creation of an emergency corridor for the rescue vehicles in the case of an accident. Thus, the behaviour of the vehicles in the simulation can be compared with the behaviour of the model vehicles in reality. It is also conceivable to extend this object detection by a distance measurement to the detected objects on the lane. This can be used, for example, to protect the radar sensor in self-driving cars.

REFERENCES

Asmara, R. A., Syahputro, B., Supriyanto, D., Handayani, A. N., 2020. *Prediction of Traffic Density Using YOLO Object Detection and Implemented in Raspberry Pi 3b + and Intel NCS 2*. 4th International Conference on Vocational Education and Training (ICOVET), ISBN: 978-1-7281-8132-5.

Bochkovskiy, A., Wang, C. Y., Liao, H. Y. M., 2020. *YOLOv4: Optimal Speed and Accuracy of Object Detection.* arXiv:2004.10934.

CNET, 2018. *Faster new Intel AI brain sticks into the side of your PC for $99. The Neural Compute Stick 2 uses a Movidius Myriad X artificial intelligence chip and is geared for prototype projects.* Cnet.com. [online]. Available at: https://www.cnet.com/news/faster-new-intel-ai-brain-sticks-into-the-side-of-your-pc-for-99/. Accessed: 10/05/2021.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., Zisserman, A., 2010. *The PASCAL visual object classes (VOC) challenge.* International Journal of Computer Vision (IJCV), Volume 88, Issue 2, pp. 303-338.

Fink, M., Liu, Y., Engstle, A., Schneider, S. A., 2019. *Deep learning-based multi-scale multi-object detection and classification for autonomous driving.* In: Fahrerassistenzsysteme 2018, Springer, ISBN 978-3-658-23751-6, pp. 233-242.

Hui, J., 2018. *mAP (mean Average Precision) for Object Detection. COCO mAP.* Jonathan-hui.medium.com. [online]. Available at: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. Accessed: 10/05/2021.

Huang, R., Pedoeem, J., Chen, C., 2018. *YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers.* IEEE International Conference on Big Data (IEEE Big Data 2018), ISBN 978-1-5386-5036-3.

Intel, 2021. *Intel Movidius Vision Processing Units (VPUs)*. Intel.com. [online]. Available at: https://www.intel.com/ content/www/us/en/products/details/processors/movidius-vpu.html. Accessed: 24/06/2021.

Ivanov, I., Skryshevsky, V., 2021. *Porous Silicon Bragg Reflector Sensor: Applying HSV Color Space for Sensor Characterization*. IEEE 16th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), ISBN: 978-1-6654-4605-1, pp. 15-19.

Jin, Y., Wen, Y., Liang, J., 2020. *Embedded Real-Time Pedestrian Detection System Using YOLO Optimized by LNN.* International Conference on Electrical, Communication, and Computer Engineering (ICECCE), ISBN 978-1-7281-7117-3.

Jose, G., Kumar, A., Kruthiventi, S., Saha, S., Muralidhara, H., 2019. *Real-Time Object Detection On Low Power Embedded Platforms*. IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), ISBN: 978-1-7281-5024-6.

Kuzmic, J., Rudolph, G., 2020. *Unity 3D Simulator of Autonomous Motorway Traffic Applied to Emergency Corridor Building*. In Proceedings of the 5th International Conference on Internet of Things, Big Data and Security, ISBN 978-989-758-426-8, pp. 197-204.

Kuzmic, J., Rudolph, G., A1, 2021. *Comparison between Filtered Canny Edge Detector and Convolutional Neural Network for Real Time Lane Detection in a Unity 3D Simulator*. In Proceedings of the 6th International Conference on Internet of Things, Big Data and Security (IoTBDS), ISBN 978-989-758-504-3, pp. 148-155.

Kuzmic, J., Rudolph, G., A2, 2021. *Object Detection with TensorFlow on Hardware with Limited Resources for Low-Power IoT Devices*. 13th International Conference on Neural Computation Theory and Applications (NCTA).

Li, P., Chen, X., Shen, S., 2019. *Stereo R-CNN Based 3D Object Detection for Autonomous Driving*. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7644-7652.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C. L., 2014, *Microsoft COCO: common objects in context*. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T, editors, Computer Vision-ECCV 2014, Springer, ISBN 978-3-319-10602-1, pp. 740-755.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., Berg, A. C., 2016. *SSD: Single Shot Multibox Detector.* In European conference on computer vision (ECCV), Springer, 2016, pp. 21-37.

Movidius, 2019. *TensorFlow Support*. Movidius.github.io. [online]. Available: https://movidius.github.io/ncsdk/tensorflow.html. Accessed: 10/04/2021.

OpenCV, 2021. *Color conversions*. Opencv.org. [online]. Available at: https://docs.opencv.org/master/de/d25/imgproc_color_conversions.html#color_convert_rgb_hsv. Accessed: 09/03/2021.

OpenVINO, 2021. *Converting a TensorFlow* Model*. Openvinotoolkit.org. [online]. Available at: https://docs.openvinotoolkit.org/latest/openvino_docs_MO_DG_prepare_model_convert_model_Convert_Model_From_TensorFlow.html#Convert_From_TF. Accessed: 05/05/2021.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. *You only look once: Unified, real-time object detection*. In Conference on Computer Vision and Pattern Recognition (CVPR).

Shi, Y., 2020. *TensorFlow 1 Detection Model Zoo*. Github.com. [online]. Available at: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md. Accessed: 23/06/2021.

Shermal, F., 2017. *Color Detection & Object Tracking*. Opencv-srf.com. [online]. Available at: https://www.opencv-srf.com/2010/09/object-detection-using-color-seperation.html. Accessed: 09/03/2021.

Tianwen, W., 2021. *OpenVINO-YOLOV4*. Github.com. [online]. Available at: https://github.com/TNTWEN/OpenVINO-YOLOV4. Accessed: 10/07/2021.

Tzutalin, 2015. *LabelImg*. Github.com. [online]. Available at: https://github.com/tzutalin/labelImg. Accessed: 07/05/2021

Vuppala, S. R., 2020. *Getting data annotation format right for object detection tasks*. Medium.com. [online]. Available at: https://medium.com/analytics-vidhya/getting-data-annotation-format-right-for-object-detection-tasks-f41b07eebbf5. Accessed: 03/03/2021.

Wang, R. J., Li, X., Ling, C. X., 2018. *Pelee: A Real-Time Object Detection System on Mobile Devices*. 32nd Conference on Neural Information Processing Systems (NeurIPS).

Yu, H., Chen, C., Du, X., Li, Y., Rashwan, A., Hou, L., Jin, P., Yang, F., Liu, F., Kim, J., Li, J., 2020. *TensorFlow Model Garden*. Github.com. [online]. Available at: https://github.com/tensorflow/models. Accessed: 19/05/2021.

Zaghari, N., Fathy, M., Jameii, S. M., Shahverdy, M., 2021. *The improvement in obstacle detection in autonomous vehicles using YOLO non-maximum suppression fuzzy algorithm.* The Journal of Supercomputing, DOI: 10.1007/s11227-021-03813-5.