# Object Detection with TensorFlow on Hardware with Limited Resources for Low-power IoT Devices

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

Keywords: Object Detection, Convolutional Neural Network (ConvNet), Autonomous Driving, Simulator in Unity 3D, Sim-to-Real Transfer, Training Data Generation, Computational Intelligence, Computer Vision, TensorFlow.

Abstract: This paper presents several models for individual object detection with TensorFlow in a 2D image with Convolution Neural Networks (ConvNet). Here, we focus on an approach for hardware with limited resources in the field of the Internet of Things (IoT). Additionally, our selected models are trained and evaluated using image data from a Unity 3D simulator as well as 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. These annotated training data are automatically generated with the Unity 3D environment. The procedure for generating annotated training data is also presented in this paper. Furthermore, the different object detection models are compared to find a 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 of the trained models is presented. Also, a transfer learning approach in object detection is carried out in this paper. Finally, future research and work in this area are discussed.



## 1 INTRODUCTION

In the autonomous vehicle industry, vehicles can drive autonomously without a driver. To do this, these vehicles have to recognise the lane or determine and classify the objects in the environment. The safety of the occupants is at the top of the list. For this reason, the topic of computer vision has become very popular in recent years. 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. Additionally, object recognition is processed using Convolutional Neural Networks (ConvNets). These are known for the successful processing of 2D images. Furthermore, the ConvNets have been proven to be effective in object detection including contour finding. In this paper, we focus on an object detection for hardware with limited resources for low-power IoT devices. For this purpose, we have trained and evaluated different models of object detection with TensorFlow. The idea is to find a suitable system for model cars with non-high-computing hardware without a GPU. However, object recognition with limited resources is not only interesting in the field of model making. Additionally, this detection of objects can be useful in other areas. For example, consider postal drones that place the package in the garden or smart surveillance cameras for agriculture which notify when certain objects e.g. animal species are detected. Object detection is not new and has been researched in the vehicle industry for some time. For example, Tesla has already recognition of pedestrians, bicycles, or other vehicles since 2014 (Wikipedia, 2021). 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.

The 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. For this purpose, the model cars have to recognise the lane from a 2D image (Kuzmic and Rudolph, 2021) and determine and classify the objects on this lane.



Kuzmic, J. and Rudolph, G.
Object Detection with TensorFlow on Hardware with Limited Resources for Low-power IoT Devices.
DOI: 10.5220/0010653500003063
In *Proceedings of the 13th International Joint Conference on Computational Intelligence (IJCCI 2021)*, pages 302-309
ISBN: 978-989-758-534-0; ISSN: 2184-3236

## 2 RELATED WORK

Numerous scientific papers are dealing with object detection, e.g. (Fink, Liu, Engstle, Schneider, 2018) who have made a deep learning-based multi-scale multi-object detection and classification for autonomous driving or (Zaghari, Fathy, Jameii, Shahverdy, 2021) who have developed the improvement in obstacle detection in autonomous vehicles using YOLO non-maximum suppression fuzzy algorithm. Some scientific works introduce a shape detection framework for detecting objects in cluttered images (Zhu, Wang, Wu, Shi, 2008). Another approach for object recognition is to distinguish the objects by 3D information. For this purpose, LiDAR sensors (Beltrán et al., 2018) or 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, Shen, 2019). Also, some related scientific papers present the sim-to-real transfer (Tan et al., 2018) or (Kahn, Abbeel, Levine, 2020). Our approach is to develop further systems and procedures for object detection for hardware with limited resources in a real environment, e.g. in the field of model making or in the farm and forest business.

## 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. Table 1 shows the data sets we have created for our object detection.

Table 1: Our data sets for object detection with TensorFlow. The resolution of the images is 1280x720 (width x height) pixel. Count stands for the number of records.

| No. | Name | Classes | Class labels | Count |
|---|---|---|---|---|
| 1 | Sim 1 | 1 | SimCar | 300 |
| 2 | Sim 2 | 2 | SimCar, SimAnimal | 1000 |
| 3 | Mod 1 | 1 | PiCar | 111 |
| 4 | Mod 2 | 4 | PiCar, ModCar, ModAnimal, ModPerson | 200 |

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. Data sets 1 and 2 were created and automatically annotated with our simulator. Data set 1 contains a simulation car *SimCar* with various objects. Here, only the simulation car is labelled and not the other objects. So, the ConvNet can learn the difference to the other objects. Data set 2 has been expanded and contains the labels *SimCar* and *SimAnimal.* Data sets 3 and 4 were created with some objects from the model making area. We annotated this data manually. Data set 3 contains a model car *PiCar* and data set 4 additionally *ModCar*, *ModAnimal* and *ModPerson* from the real world. The following Figure 1 shows some images of the created training data from our data sets. The split of training and test data is 80/20. The test data was used as validation data.


Figure 1: Our several data sets for object detection with TensorFlow. First two rows: Data set from simulator. Last two rows: Data set from model making area.

Each of data sets 1 and 3 have just one class (*SimCar* and *PiCar*). So, results from simulation and model making area can be compared to find a suitable object detection system for hardware with limited resources. If object detection on real data has to be implemented, already published data sets MS COCO (Lin et al., 2014) or PASCAL VOC (Everingham et al., 2010) can be used. These already contain thousands of annotated real objects.

### 3.1 Automatic Labelling

With our simulator in Unity 3D (Kuzmic and Rudolph, 2020), thousands of annotated training data (input and output data) could be generated automatically. For the automatic generation of the required data sets, two virtual cameras were installed in the same place in a simulated car in the simulator. The first camera could see everything in the virtual environment (Fig. 2, left 1). The second camera only saw the object to be recognised (Fig. 2, left 2). These two images could be used for the further generation of the training data. The image from the first camera is the input image for the ConvNet. The image from

the second camera was converted into a greyscale image first. To get the annotation of the data (output data), a binary image was created from the greyscale image next (Fig. 2, right 1). From this binary image (black background, white 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 2) for the input image as coordinates for P (xMin, yMin) and Q (xMax, yMax). Then, these coordinates are stored in an XML file (Vuppala, 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.

Figure 2: Training data generation in our Unity 3D simulator. Left 1: Image from first camera. Left 2: Image from second camera. Right 1: Binary image. Right 2: Information for the position of the object P and Q.

### 3.2 Manual Labelling

For the data set with real data, some pictures from several models from model making were taken. Afterwards, these images were manually annotated with the *LabelImg* tool (Tzutalin, 2021). 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 are stored in an XML file with the same name as the image file.

## 4 OBJECT DETECTION

For the object detection with TensorFlow (Huang et al., 2017), pre-trained models, that were already established in object detection with real data, were investigated. The run time measurements of the pre-trained TensorFlow (TF) models were performed on the *NVIDIA GeForce GTX 1660 Ti* GPU. This makes it possible to get first differences for the run times of the models. Our pre-experiments were performed with the same 200 images to measure the run time of the models on our hardware and to make a small pre-selection. There are several ConvNet models for TensorFlow 2 available in the TensorFlow 2 detection model zoo (Chen, 2021). These models have already been pre-trained on the MS COCO 17 data set and can detect and classify 90 objects. The speed of detection is shown as frames per second (FPS) in table 2. The accuracy is given as COCO mean average precision (mAP) metric (Hui, 2018).

Table 2: Pre-experiment to determine the fastest TensorFlow model. SSD 1: *SSD MobileNet V2*. SSD 2: *SSD ResNet50 V1 FPN*. CenterNet: *CenterNet HourGlass104.*

| No. | Name | Resolution | mAP | FPS |
|---|---|---|---|---|
| 1 | SSD 1 | 320x320 | 20.2 | 16.4 |
| 2 | EfficientDet D0 | 512x512 | 33.6 | 5.7 |
| 3 | SSD 2 | 640x640 | 34.3 | 4.7 |
| 4 | SSD 2 | 1024x1024 | 38.3 | 2.3 |
| 5 | CenterNet | 1024x1024 | 44.5 | 1.7 |
| 6 | EfficientDet D4 | 1024x1024 | 48.5 | 1.5 |

As expected, this small preliminary experiment confirms: a smaller resolution of the model gives a faster processing of the images. After a comparison of the mAP and FPS it can be seen, that a higher resolution of the ConvNets gives a more accurate object detection. For this reason, a balance between the precision and the run time had to be found to use object detection on hardware with limited resources. To compare the different TensorFlow models, some models were trained and evaluated on the already presented data sets. The *base learning rate* was set to 0.008. The *warmup learning rate* is 0.0001. The *batch size* is 4. By default, these values are higher. The MS COCO data set contains much more training data. So in our scenario, we reduced these parameter values. Tables 3 and 4 below show our trained models. *Steps* denotes the value for the last training step. This parameter is important to avoid overtraining of the model.

Table 3: Overview of trained *EfficientDet D0* 512x512 models.

| No. | Steps | Data Set | mAP |
|---|---|---|---|
| 1 | 150 k | Sim 1 | 95.8 |
| 2 | 100 k | Sim 2 | 87.1 |
| 3 | 150 k | Mod 1 | 91.7 |
| 4 | 100 k | Mod 2 | 76.4 |

A comparison of the precisions shows that models with higher input image size are more accurate (comparison between table 3 no. 2 and table 4 no. 2).

Table 4: Overview of trained *SSD MobileNet V2* 320x320 models.

| No. | Steps | Data Set | mAP |
|---|---|---|---|
| 1 | 150 k | Sim 1 | 93.4 |
| 2 | 100 k | Sim 2 | 79.6 |
| 3 | 150 k | Mod 1 | 92.1 |
| 4 | 100 k | Mod 2 | 69.3 |

All trainings were carried out with the same settings of the parameter and training data. First, the models trained with the simulation data were evaluated. Sometimes these models detect objects in places where no objects can be seen. To solve this, more training data with the *PiCar* and many different shapes is needed. Afterwards, we started with the *EfficientDet D0* 512x512 model and one output class to get a model which detects individual objects from reality and needs few resources. Here, only the *PiCar* and no other objects are detected and classified. Figure 3 shows this recognition. Rectangles show the detection of the object by the ConvNet.

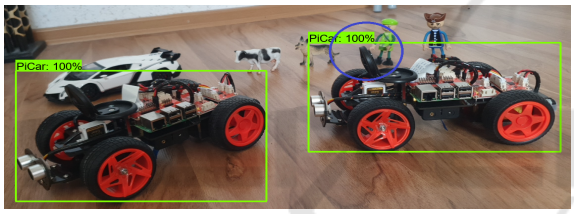


Figure 3: Object detection with *EfficientDet D0* 512x512 model (no. 3 in table 3). Class: *PiCar*.

To make a comparison, the *SSD MobileNet V2* 320x320 model was trained with the same output class (Fig. 4). As can be seen, this model is less accurate in recognising the *PiCar* (figures 3 and 4, marked in blue). But this is not reflected in the mAP (comparison between table 3 no. 3 and table 4 no. 3). These recognitions are completely acceptable for our purpose and sufficient for our application.

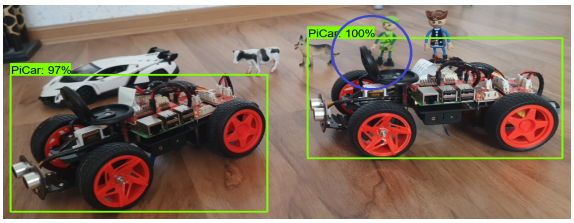


Figure 4: Object detection with *SSD MobileNet V2* 320x320 model (no. 3 in table 4). Class: *PiCar*.

Subsequently, these models were extended. Figure 5 shows the *EfficientDet D0* 512x512 model with four output classes. This model detects and classifies the objects very exactly. The detected position and size of the objects also match. Here, a precision of 76.4 % can be achieved. For comparison, figure 6 shows the detections of the *SSD MobileNet V2* 320x320 model with four output classes, too. Also, this model recognises very exactly the objects with four classes *PiCar, ModCar, ModAnimal, ModPerson* and achieves a lower precision of 69.3 %. But even this detection is completely sufficient for our purpose. Also, the never seen objects could be detected by our models. For example, a Lego person, a model horse and a blue model car are recognised by these two models (Fig. 7).

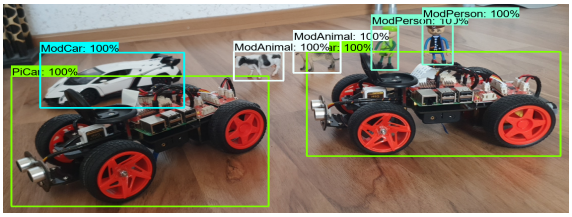


Figure 5: Object detection with *EfficientDet D0* 512x512 model (no. 4 in table 3). Classes: *PiCar, ModCar, ModAnimal, ModPerson*.

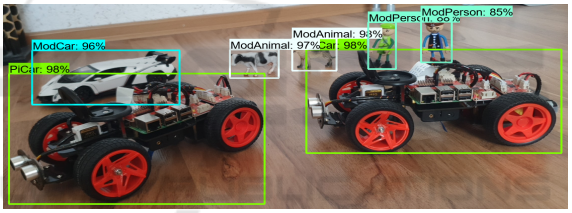


Figure 6: Object detection with *SSD MobileNet V2* 320x320 model (no. 4 in table 4). Classes: *PiCar, ModCar, ModAnimal, ModPerson*.

During training the ConvNets have already seen figures of model persons, model animals and model cars. These ConvNets learned the similar shapes of the objects and not only images from the training data set.

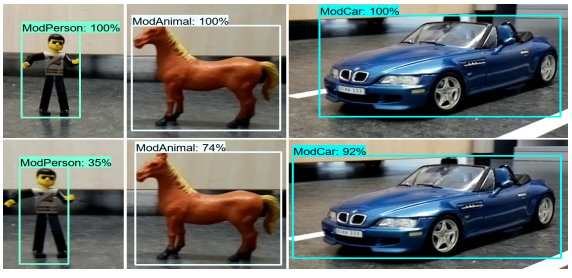


Figure 7: Object detection on never seen objects. First row: Detection with *EfficientDet D0* 512x512 model (no. 4 in table 3). Last row: Detection with *SSD MobileNet V2* 320x320 model (no. 4 in table 4).

Furthermore, every other object with an unknown shape is recognised 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. Additionally, we have converted the *SSD MobileNet V2* model with 320x320 pixels to a 224x224 pixel model. This model is not present in the TensorFlow 2 model zoo (Chen, 2021). Since, it is interesting to see how the models perform in precision and run time. Table 5 shows an overview of the accuracy of these models.

Table 5: Overview of trained *SSD MobileNet V2* 224x224 models.

| No. | Steps | Data Set | mAP |
|---|---|---|---|
| 1 | 150 k | Mod 1 | 90.9 |
| 2 | 100 k | Mod 2 | 60.6 |

With this model 60.6 % accuracy is achieved. For comparison, the *SSD MobileNet V2* 320x320 model achieves 69.3 % (no. 4 in table 4). The next figure 8 shows this object detection on the *Mod 2* data set.

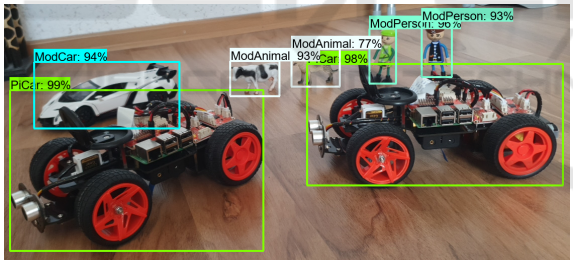


Figure 8: Object detection with *SSD MobileNet V2* 224x224 model. Classes: *PiCar, ModCar, ModAnimal, ModPerson*.

# 5 EXPERIMENTS

The following experiments were carried out to compare the functionality and the run time of our different TensorFlow models on the same hardware. The resolution is in the format width x height. Training of the ConvNets on Google Colab: *Intel Xeon 2.30 GHz CPU, 26 GB RAM, NVIDIA Tesla P100-PCIe-16GB GPU*. Run time measurements with GPU: *Intel i7-9750H 2.60 GHz CPU, 16 GB RAM, 256 GB SSD, NVIDIA GeForce GTX 1660 Ti GPU*. Run time measurements on Raspberry Pi 3 B for hardware with limited resources: *ARM Cortex-A53 1.2 GHz CPU, 1 GB RAM, 8 GB SD*. 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.

## 5.1 Run Time on GPU

As can be seen in previous section 4 very good results in object detection were already achieved with the *EfficientDet* and *SSD MobileNet* models. In this experiment we test the run time of these models on hardware with GPU. Therefore, we only test the run time of the models for the real data from model making area (Tab. 6). Afterwards, these results can be compared with the run times achieved on the hardware with limited resources. First column contains the number (id) of the experiment (Exp. No.).

Table 6: Run time overview of trained TensorFlow models on hardware with GPU. EfficientDet: *EfficientDet D0* 512x512 TF2. SSD 224: *SSD MobileNet V2* 224x224 TF2. SSD 320: *SSD MobileNet V2* 320x320 TF2.

| Exp. No. | Model | Steps | Data Set | Run Time [FPS] |
|---|---|---|---|---|
| 1 | EfficientDet | 150 k | Mod 1 | 14.6 |
| 2 | EfficientDet | 100 k | Mod 2 | 14.2 |
| 3 | EfficientDet | 150 k | Mod 2 | 14.1 |
| 4 | SSD 224 | 150 k | Mod 1 | 40.1 |
| 5 | SSD 224 | 100 k | Mod 2 | 39.2 |
| 6 | SSD 224 | 150 k | Mod 2 | 40.8 |
| 7 | SSD 320 | 150 k | Mod 1 | 39.8 |
| 8 | SSD 320 | 100 k | Mod 2 | 38.1 |
| 9 | SSD 320 | 150 k | Mod 2 | 38.3 |

As can be seen in table 6, the fastest model is *SSD MobileNet V2* 224x224 with 150 k steps and four output classes *PiCar*, *ModCar*, *ModPerson* and *ModAnimal*. This model achieves approx. 41 FPS. These measurements do not include loading and processing of the input images. With one class *PiCar* the same model accomplishes up to 40 FPS.

## 5.2 Run Time on Limited Resources

In the following experiment the run time measurements are carried out on hardware with limited resources. A Raspberry Pi 3 B was used for this test. To increase the run time of processing a quantised *SSD MobileNet V2* 224x224 model was created for this experiment. Quantisation converts the

weights of the model from *float* to *uint8* (TensorFlow Performance, 2021). *TFLite* represents a converted model from TensorFlow Core to TensorFlow Lite. Table 7 shows the run time measurements at a glance.

Table 7: Run time overview of trained TensorFlow models on hardware with limited resources. EfficientDet: *EfficientDet D0* 512x512. SSD 224 Q: *SSD MobileNet V2* Quantized 224x224. SSD 224: *SSD MobileNet V2* 224x224. SSD 320: *SSD MobileNet V2* 320x320.

| Exp. No. | Model | Steps | Data Set | Run Time [FPS] |
|---|---|---|---|---|
| 1 | EfficientDet | 150 k | Mod 1 | - |
| 2 | EfficientDet | 100 k | Mod 2 | - |
| 3 | EfficientDet | 150 k | Mod 2 | - |
| 4 | SSD 224 Q | 150 k | Mod 1 | 3.2 |
| 5 | SSD 224 Q | 100 k | Mod 2 | 2.8 |
| 6 | SSD 224 Q | 150 k | Mod 2 | 3.0 |
| 7 | SSD 224 | 150 k | Mod 1 | 1.4 |
| 8 | SSD 224 | 100 k | Mod 2 | 1.4 |
| 9 | SSD 224 | 150 k | Mod 2 | 1.4 |
| 10 | SSD 320 | 150 k | Mod 1 | 0.9 |
| 11 | SSD 320 | 100 k | Mod 2 | 0.9 |
| 12 | SSD 320 | 150 k | Mod 2 | 0.9 |

For the *EfficientDet D0* 512x512 models the measurement was automatically stopped by the Raspberry Pi after some time. No run time measurements could be carried out for these models on a Raspberry Pi. An out-of-memory error has occurred on this system. As can be seen after these experiments: the *SSD MobileNet V2* Quantised 224x224 model with 150 k steps and a *PiCar* output class is the fastest model. Here, the model accomplishes approx. 3.2 FPS. These measurements also do not include loading and processing of the input images. With four classes *PiCar*, *ModCar*, *ModPerson* and *ModAnimal* the same model achieves up to 3 FPS.

## 5.3 Evaluation of the Run Time

After the 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. For hardware with GPU: real-time object detection can be achieved with the models *SSD MobileNet V2* 224x224 and *SSD MobileNet V2* 320x320 (exp. no. 4 to 9 in table 6). For hardware without GPU: approx. 3.2 FPS can be achieved with the *SSD MobileNet V2* Quantized 224x224 model on the Raspberry Pi 3 B (exp. no. 4 in Table 7). To speed up this object detection, the *tflite runtime-2.5.0-cp37* library (TensorFlow Lite, 2021) and a quantised SSD TensorFlow 1 model were used (TensorFlow Performance, 2021). The run time measurements of the non-lite TensorFlow 2 models were performed with the *TensorFlow 2.4.0-rc2 library* (TensorFlow Core, 2021) on the Raspberry Pi 3 B. Additionally, object detection does not have to evaluate every frame. Object detection can be done as soon as anomalies are detected. In an autonomous vehicle it can be done, for example, with the radar sensor. In smart surveillance cameras it could be the motion sensor, for example.

## 5.4 Sim-to-Real Transfer

While training the different models with the simulation and the model data the following research question arose: *If the objects in the simulation look similar to model objects from the real world, which of our ConvNets can best recognise the real model objects?* To answer this question, we have trained two different models to perform a transfer learning approach. By creating the data in the simulation, many different data with many different models of an object can be created. Additionally, there are already a lot of modelled objects from the video game field. So, the modelling of own objects is not required. For this sim-to-real transfer we tested two models *EfficientDet D0* 512x512 and *SSD MobileNet V2* 320x320. These models were trained with the same data set *Sim 1* (includes *SimCar*) and 150 k steps to compare the results afterwards. Table 8 shows the comparison of the models evaluated on the test data from simulation and reality.

Table 8: Overview of trained TensorFlow sim-to-real models with object detection in simulation and real model data.

| Exp. No. | Model | Detect Sim | Detect Real |
|---|---|---|---|
| 1 | EfficientDet D0 512x512 | Yes | Yes |
| 2 | SSD MobileNet V2 320x320 | Yes | No |

As can be seen in figure 9, first row, the *SSD MobileNet V2* 320x320 model achieves accurate recognition on the simulation data. However, the

detection on the real model data does not work (Fig. 9, last row). Green rectangle shows the detection of the object by the ConvNet. Thus, the *SSD MobileNet V2* 320x320 model is only suitable for recognising already seen objects during the training. According to our experiments, this model is not suitable for the sim-to-real approach. We assume the resolution of this ConvNet is too small.

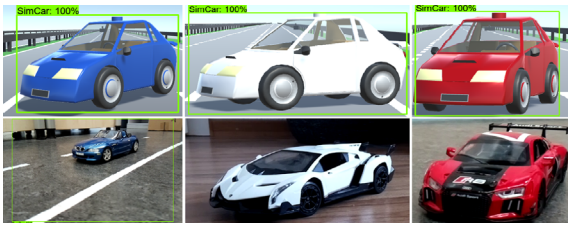


Figure 9: Sim-to-real transfer with *SSD MobileNet V2* 320x320. First row: Images from simulation. Last row: Model car images from the real world.

On the other hand, the *EfficientDet D0* 512x512 model is suitable for a sim-to-real approach. The next figure 10 shows the object detection with this model. The model car can be detected in simulation (Fig. 10, first row) and in real data from the real world (Fig. 10, last row). The images of the model car were never seen by this ConvNet before.

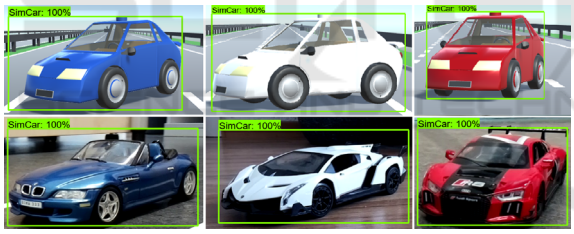


Figure 10: Sim-to-real transfer with *EfficientDet D0* 512x512. First row: Images from simulation. Last row: Model car images from the real world.

As can be seen, the *EfficientDet D0* 512x512 model detects the model cars very accurate. Conversely, some of the other objects (animals and persons) are also detected as *SimCar* (Fig. 11).

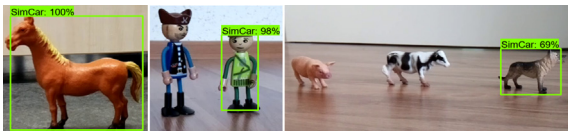


Figure 11: Object detection with *EfficientDet D0* 512x512 Model. Left: Detected model horse. Middle: Detected model person. Right: Detected model dog.

The reason is, such data was never seen by the ConvNet. So, the model has only learned the shape of the object *SimCar* and did not have other objects for comparison. However, this sim-to-real approach is worthy of improvement. We assume that these inaccuracies can be corrected with several training data and different models in simulation. But on the whole, from our experiments can be seen: the sim-to-real transfer can be successfully performed in object detection with the *EfficientDet D0* 512x512 Model.

## 6 CONCLUSIONS

This section summarizes once again the points that were introduced in this paper. For object detection with TensorFlow we have focused on hardware with limited resources for low-power IoT devices. The acquisition of automated annotated training data from the simulation is also presented. Furthermore, training data from individual objects in the field of the model making to compare the results afterwards were created. Additionally, several different TensorFlow models were trained to find a balance between the accuracy and the run time of these models. According to our experiments the *SSD MobileNet V2* with 224x224 pixels as well as the same model with 320x320 pixels resolution is suitable for object detection in real-time scenarios with GPU. A suitable model for object detection in hardware with limited resources for low-power IoT devices is the *SSD MobileNet V2* Quantized 224x224 model. This model achieves up to 3.2 FPS on a Raspberry Pi 3 B without hardware extension. The *SSD MobileNet V2* 224x224 models achieve an effective balance between accuracy and run time. Finally, a transfer learning approach in object detection was conducted in this work. With the *EfficientDet D0* 512x512 model this sim-to-real approach can be successfully carried out.

## 7 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. When developing software for hardware with limited resources for low-power IoT devices it is also interesting to see how the run time can be improving with a Raspberry Pi 4 B with hardware extension such as the Intel Neural Compute Stick 2 (CNET, 2018) or the Google Coral USB Accelerator (Coral, 2020). This approach will be explored in our future research.

# REFERENCES


Beltrán, J., Guindel, C., Moreno, F. M., Cruzado, D., García, F., De La Escalera, A., 2018. *BirdNet: A 3D Object Detection Framework from LiDAR Information.* 2018 21st International Conference on Intelligent Transportation Systems (ITSC), IEEE, ISBN 978-1-7281-0324-2.

Chen, Y. H., 2021. *TensorFlow 2 Detection Model Zoo*. Github.com. [online]. Available at: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. Accessed: 03/03/2021.

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.

Coral, 2020. *USB Accelerator datasheet*. Coral.ai. [online]. Available at: https://coral.ai/docs/accelerator/ datasheet/. 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, pp. 233-242. Springer, ISBN 978-3-658-23751-6.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., Murphy, K., 2017. *Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). ISBN 978-1-5386-0457-1.

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.

Kahn, G., Abbeel, P., Levine, S., 2020. *LaND: Learning to Navigate from Disengagements.* arXiv: 2010.04689.

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., 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, ISBN 978-989-758-504-3, pp. 148-155.

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.

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., Vanhoucke V., 2018. *Sim-To-Real: Learning Agile Locomotion For Quadruped Robots*. Proceedings of Robotics: Science and System XIV, ISBN 978-0-9923747-4-7.

TensorFlow Core, 2021. *TensorFlow Core v2.4.1, API Documentation*. TensorFlow.org. [online]. Available at: https://www.tensorflow.org/api_docs. Accessed: 07/05/2021.

TensorFlow Lite, 2021. *For Mobile & IoT*. TensorFlow.org. [online]. Available at: https://www.tensorflow.org/lite. Accessed: 09/05/2021.

TensorFlow Performance, 2021. *For Mobile & IoT, Post-training quantization*. TensorFlow.org. [online]. Available at: https://www.tensorflow.org/lite/performance/post_training_quantization. Accessed: 09/05/2021.

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.

Wikipedia, 2021. Tesla Autopilot, Driving features. Wikipedia.org. [online]. Available at: https://en.wikipedia.org/wiki/Tesla_Autopilot#cite_note-:2-70. Accessed: 05/03/2021.

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 (2021). DOI: 10.1007/s11227-021-03813-5.

Zhu, Q., Wang, L., Wu, Y., Shi, J. (2008) Contour Context Selection for Object Detection: A Set-to-Set Contour Matching Approach. In: Forsyth D., Torr P., Zisserman A. (eds) Computer Vision – ECCV 2008. ECCV 2008. Lecture Notes in Computer Science, vol 5303. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88688-4_57 ISBN 978-3-540-88685-3.