2019 5th Experiment@ International Conference (exp.at'19)
June 12th – 14th, 2019, University of Madeira, Funchal, Madeira, Portugal

Object detection with Raspberry Pi3 and Movidius Neural Network Stick

Andreas Pester
Engineering&IT
Carinthia University of Applied Sciences
Villach, Austria
https://orcid.org/0000-0001-7278-7349

Michael Schrittesser
Engineering&IT
Carinthia University of Applied Sciences
Villach, Austria
m.schrittesser@fh-kaernten.alumni.at

***Abstract*—Object detection and classification is an increasingly important field of research in machine learning. Currently, powerful GPUs (Graphics Processing Units) are used to perform the computation-intensive operations in the shortest possible computing time. However, these systems are associated with high costs. In this paper a system for object detection and classification is developed, which gets by with less resources. This should minimize the costs while keeping the performance acceptable for the target application. To keep the costs low, a Raspberry Pi3 is used as development platform in connection with a Movidius stick for the outsourcing of the ANN. After explaining the theoretical basics of object detection and ANNs, this paper shows the implementation process of the selected hardware and software. For the evaluation of this system the algorithms YOLO and MobileNet are used and pre-trained models are used as basis. Based on the MSCOCO data set, both the quality of the object classification and the computing time are evaluated.**

Keywords— Object detection, classification, ANN (artificial neural network), CNN (convolutional neural network), Movidius, YOLO (You Only Look Once), MobileNetSSD

I. Introduction

Due to current technical advances, it is becoming increasingly important for machines to be able to interpret images. An example of this would be autonomous driving or quality control in industry 4.0. Identification and classification of objects on the images is done through object detection and classification systems that have evolved in recent years.

For such systems, computers with a lot of computing power and a strong GPU are used to recognize and classify objects in real time. Thus, such systems relate to high costs and are not usable in mass applications. A current approach to avoid this is a cloud-based solution. Central data centres are used to perform the calculations. This means, however, that the data from the own system must be sent to an external data centre, which creates quite a big data traffic and data security risks.

To improve this situation, the aim of this work is to develop a system that manages with less computing power and the same accuracy. To achieve this on a miniature computer like the Raspberry Pi3 used in the paper, the neural network used to classify the objects was not stored in a cloud but on a neural computer stick. For this implementation, a Movidius stick [1] was used.

Since the Movidius stick does not support training of neural networks, pre-trained models of the YOLO and MobileNet were used in this work. The YOLO net was downloaded from GitHub at [2]. For the MobileNet algorithm, Movidius provides a pre-trained network in ncappzoo which can be downloaded from [3].

II. Preparation of the Rasperry and the Movidius Stick

A. Implementation of the OpenCV library on Raspberry Pi3

Initially, the first tests were carried out on a virtual machine with Ubuntu 16.04 and then transferred to a Raspberry Pi3 with Raspbian Stretch (Release 2018-03-13, Debian 9.0). The system was programmed with Python 3.5 and the Open Source library OpenCV 3.4.4. The well-known computer vision library OpenCV is used for object detection in this work. OpenCV is an open source computer vision and machine learning software library. OpenCV was developed to provide a common infrastructure for computer vision applications and thus accelerate progress in the commercial field. [4]

The library has more than 2,500 optimized algorithms, including a comprehensive collection of both classical and modern algorithms for computer vision and machine learning. These algorithms can be used, for example, to recognize faces or identify objects. [4]. The implementation steps are:

- Installation of CMake, Image and Video I/O packages, and GTK-library
- Creation and activation of a virtual Python 3 environment
- Installation of NumPy
- Setting up Cmake to configure the OpenCV build process
- Compilation and installation of OpenCV
- Sym-link OpenCV dependencies into the virtual environment
- Test of the correct installation of OpenCV

B. Implementation of the Intel Movidius Stick on the Raspberry

The Intel Movidius Stick was designed based on a careful tuning of programmable vector processors, dedicated hardware accelerators and memory architecture for an optimized data flow. The Movidius Stick has a software-driven multi-core memory subsystem with multiple ports and caches that can be configured to allow a wide range of workloads. This technology enables exceptionally high instruction bandwidth to support 2 CPUs and high-performance video hardware accelerators.

To ensure consistently high computing performance and minimize power consumption, the Movidius stick's processor includes wide and deep register files in conjunction with a Variable Length Long Instruction Word (VLLIW) that controls multiple functional units. The SHAVE processor is a hybrid stream processor architecture that combines the best features of GPUs (Graphics Processing Unit), DSPs (Digital Signal Processor) and RISC (Reduced Instruction Set Computer). The architecture is designed to maximize performance per watt while ensuring programmability, especially with respect to computer vision and machine learning workloads. [5]

Fig. 1 Intel Movidius Neural Computer Stick

The Movidius stick provides several key elements for implementing deep learning applications on a low-power device such as the Raspberry Pi3:

- Performance: The raw performance of the Myriad SHAVE processor engines reaches hundreds of GFLOPS required for matrix multiplication calculations. [5]

- On-Chip-RAM: Deep neural networks generate large amounts of intermediate data, which are stored at the Movidius stick to avoid performance bottlenecks at the computer, like the Raspberry Pi3. [5]

- Flexible: The Movidius stick is hardware independent and can be used on different platforms like Ubuntu, Raspbian or Windows. [5]

- High Performance Libraries: The development package includes software libraries that work hand in hand with the architecture to support high and stable performance in matrix multiplication and multidimensional convolution. [5]

To be able to use the Movidius stick with all its functions on Raspberry Pi3, the SDK (Software Development Kit) provided by Intel must be installed

C. Implementation of the Neural Network on the Movidius Stick

The Movidius applications programmed in Python are realized by calling certain functions provided by Movidius. To be able to use the Movidius library, the Movidius API (Application Programming Interface) must be imported at the beginning of the program.

Before functions are executed on the Movidius stick, the number of detected devices is determined, and it is checked whether at least one stick is present. If this is not the case, the program is terminated.

```
from mvnc import mvncapi as mvnc

devices = mvnc.EnumerateDevices()
if len(devices) == 0:
    print('No devices found')
    quit()
```

If at least one device is detected, the first stick is used and stored in the variable device, thus activating the stick. Furthermore, the path or the name of the graph file is defined.

```
device = mvnc.Device(devices[0])

device.OpenDevice()

graph_file_name = 'graph'
```

Then the graph file is loaded into the memory and a graph instance of the API is created from the memory and stored in the variable graph. This variable can then be used to start the actual process.

```
with open(graph_file_name, mode='rb') as
f:
    graph_in_memory = f.read()

graph =
device.AllocateGraph(graph_in_memory)
```

After the execution of the program, the graph will be deleted, the memory of the stick will be cleared and the access to the stick will be deactivated.

```
graph.DeallocateGraph()
device.CloseDevice()
```

III. PERFORMANCE ANALYSIS OF THE YOLO AND THE MOBILENETSSD ALGORITHMS

For the performance analysis of object detection algorithms with a Movidius stick on Raspberry Pi3 two algorithms were tested, YOLO and MobileNetSSD. To train the two algorithms, the library for neural networks from MOVIDIUS was used. This library can be accessed by an own API. Additionally, INTEL provides a Neural Compute Engine on the stick, which increases the performance of deep neural networks. INTEL provides no information about the used deep learning frameworks, used in the MOVIDIUS API.

The "Train2014" data set from MSCOCO was used to evaluate the performance. This data set contains over 80,000 images and 90 categories. Since the algorithms MobileNetSSD and YOLO were trained on only twenty categories, only part of the data set is used for the evaluation. How the scaling of the data set was performed is described in part III.A. In addition, this subchapter contains all further preparation steps that are necessary to perform the analysis. In part III.B, the analysis of the algorithms is performed and interpreted using the key figures Precision, Recall and F-Measure. Finally, it is examined how the computing time behaves when using different hardware such as laptop and Raspberry Pi3.

YOLO ((You Only Look Once)) was developed for object detection. It is designed for finding and recognizing patterns on an image or in a video and so part of computer vision. YOLO algorithm differs from others in that it uses a single CNN network for both classification and localization of the object using boundary frames. [6]. This is like the Faster R-CNN algorithm [7].

The structure of the CNN network shown in Figure 2 was inspired by the network architecture of GoogLeNet [8] and consists of 24 convolution layers followed by two fully connected layers. Instead of the Inception modules used by GoogLeNet, the YOLO network uses a simple 1x1 reduction layer followed by a 3x3 convolution filter. The output is a 7x7x30 tensor of predictions. Furthermore, a fast YOLO network was developed which uses nine convolution layers instead of the 24 convolution layers.

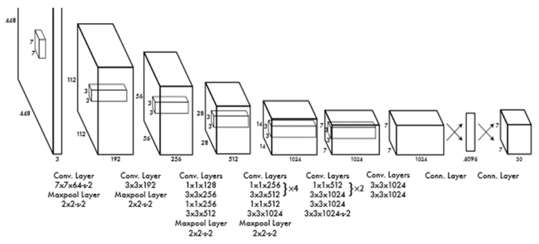
448
448
3
112
192
56
256
28
512
14
1024
7
1024
1024
4096
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Fig. 2 YOLO Architecture [6]

The MobilNetSSD algorithm was published in November 2016 by W. Liu et al. and achieved new performance and accuracy records in the field of object detection. With 74% mAP (mean average precision) and 59 frames per second on standard data sets such as MSCOCO, MobilNetSSD is one of the most powerful CNNs for object detection. [9]

As shown in Fig. 3, MobilNetSSD's architecture is like that of VGG-16 [10], but without the Fully Connected Layer. VGG-16 was used as the base network because it is known for its high-quality image classifications. Instead of the Fully Connected Layer, additional Convolution Layers have been added, allowing features in the image to be extracted and the input size for each subsequent layer to be progressively reduced. [9]

MobilNetSSD has undergone several optimizations to improve this network in the localization and classification of objects. Unlike the MultiBox method used in the YOLO algorithm, each feature map cell is linked to a set of standard boundary fields with different dimensions and aspect ratios. Unlike MultiBox, MobilNetSSD does not use a classification threshold, but uses multiple-scales maps to predict classes and boundary boxes. By using the multiple-scales maps, a higher mAP can be achieved because objects of different sizes can be better recognized on an image.

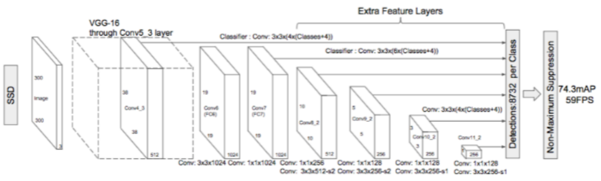
Extra Feature Layers
VGG-16 through Conv5_3 layer
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
SSD
300
Image
300
3
38
Conv4_3
38
512
19
Conv6 (FC6)
19
1024
19
Conv7 (FC7)
19
1024
10
Conv8_2
10
512
5
Conv9_2
5
256
3
Conv10_2
3
256
Conv11_2
256
Conv: 3x3x(4x(Classes+4))
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP 59FPS
Conv: 3x3x1024
Conv: 1x1x1024
Conv: 1x1x256 Conv: 3x3x512-s2
Conv: 1x1x128 Conv: 3x3x256-s2
Conv: 1x1x128 Conv: 3x3x256-s1
Conv: 1x1x128 Conv: 3x3x256-s1

Fig. 3 Architecture of MobilNetSSD [7]

A. Preparing data and algorithm

The MSCOCO data set consists of both: the images used for evaluation and the information about the images in the form of a JSON file. From this file, the sections Images, Annotations and Categories are required to assign the correct class (label) to the objects on the images. On one image can be more than one object.

The Categories section was used to compare the classes of the data record with the classes of the algorithms. This filtered out the relevant IDs and names of the classes that are listed in table 1. The filtered-out class labels were then assigned to the correct classes.

Table 1 ID's and related classes

ID	class
1	person
2	bicycle
3	car
4	motorcycle
5	airplane
6	bus
7	train
9	boat
16	bird
17	cat
18	dog
19	horse
20	sheep
21	cow
44	bottle
62	chair
63	couch
64	potted plant
67	dining table
72	tv

With the filtered classes, two hundred images from all classes were selected for evaluation. To separate the selected images from the 80,000-image dataset, a bash script was written whose code is shown below. This script copies the 200 images listed in a txt file to a separate folder.

```
@echo off
cd C:\Users\user\Dokumente\BA\train2014\
  for /F "delims=," %%i in (src.txt) do
                copy %%i
    C:\Users\user\Dokumente\BA\bilder\
```

From the annotation area the links of the images with the classes for the selected images are extracted. Since the annotations link the images with the IDs of the classes based on the image IDs, a list of the image names with the IDs of the images is filtered out of the JSON file. By linking the lists created so far, a list with picture name and class is created, which are displayed on the pictures. An extract from this list is shown in Table 2.

For subsequent analysis the recognized classes and the computing time per image are written into a csv file. The file containing the recognized classes has the same layout as the file with the correct assignments (see Table 2). For this the existing algorithm of YOLO and MobileNetSSD had to be adapted. The adapted algorithms can be found in [11]. Furthermore, by adapting the algorithms, the 200 images are automatically read, processed and saved.

Table 2 image and related class

image	class
bild (132).jpg	aeroplane

bild (133).jpg	person
bild (133).jpg	person
bild (133).jpg	bicycle
bild (133).jpg	bicycle
bild (134).jpg	person
bild (134).jpg	horse
bild (135).jpg	person
bild (135).jpg	person
bild (135).jpg	person
bild (135).jpg	person
bild (135).jpg	horse
bild (135).jpg	horse

B. Data analysis

Cross-evaluation is used to determine how many objects were correctly detected (TP), how many objects were not detected (FN) and how many objects were incorrectly detected (FP). Reference was made to each class and to all objects. (see Table 3). The parameter "True Negative" (TN) which describes the correct non-detection of a non-existent object, was not used, because in this evaluation not only one, but all objects present in an image must be classified.

Table 3 Analysis of the detected objects

		MSCOCO	MobilNetSSD			YOLO		
ID	**Object class**	**# Objects**	**TP**	**FN**	**FP**	**TP**	**FN**	**FP**
1	aero plane	21	18	3	2	12	9	5
2	bicycle	49	17	32	3	6	43	0
3	bird	46	15	31	7	10	36	7
4	boat	33	15	18	2	8	25	0
5	bottle	72	4	68	0	0	72	2
6	bus	30	17	13	3	11	19	0
7	car	101	34	67	7	21	80	6
8	cat	14	11	3	2	4	10	0
9	chair	133	44	89	8	24	109	7
10	cow	44	20	24	9	10	34	3
11	dining table	40	17	23	9	8	32	1
12	dog	27	14	13	7	5	22	4
13	horse	29	17	12	9	7	22	1
14	motorbike	33	17	16	3	8	25	2
15	person	534	284	250	17	222	312	40
16	potted plant	52	12	40	7	2	50	0
17	sheep	64	40	24	4	26	38	9
18	sofa	38	23	15	13	9	29	4
19	train	18	9	9	0	7	11	3
20	tv monitor	22	11	11	6	5	17	6
	Result	**1 400**	**639**	**761**	**118**	**405**	**995**	**100**

This table shows that the two algorithms correctly detected less than half of the existing objects. It is noticeable that YOLO did not recognize any objects of the class "bottle", although a total of 72 bottles are shown on the 200 images. Image 149, for example, shows two bottles, but, as shown in Fig. 4, the YOLO algorithm did not detect any of them.

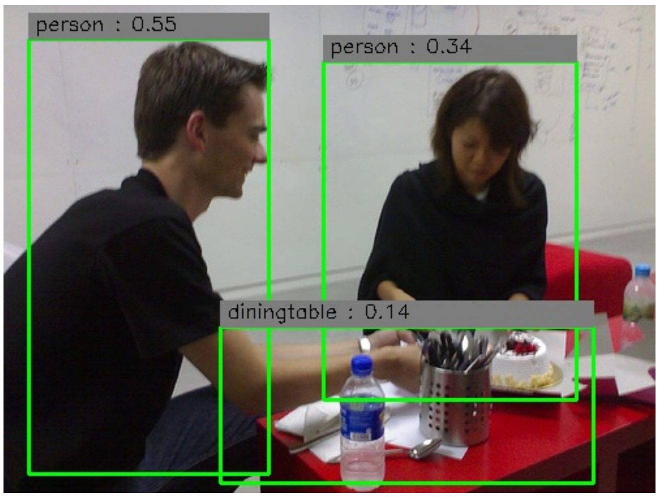

Fig. 3 Evaluation YOLO image 149 with two unrecognized bottles

Various indicators are used to determine the accuracy and correctness of the algorithms for each class and overall. Based on the evaluation of the recognized objects, the key figures Precision, Recall and F-measure were calculated. The key figure Precision shows the proportion of recognized objects that were recognized correctly. How many objects were recognized correctly in relation to all objects that should be recognized is expressed by the key figure Recall. F-Measure combines Precision and Recall and forms the harmonic mean of the two indicators.

In Table 4, the key figures calculated based on the entire analysis run are summarized. The key figure Recall is equivalent to the key figure Sensitivity. This overview shows that the MobilNetSSD performs slightly better overall than the YOLO algorithm. The precision of the algorithms is high at over 80%, but the recall is low at under 50%. As a result, the F-Measure is average at around 50%. When objects are detected, the MobilNetSSD algorithm with an F-Measure of 59.25% performs about 17% better than the YOLO algorithm with 42.52%.

One reason for the better performance of the MobileNetSSD algorithm is that MobileNetSSD automatically outputs all detected objects. The YOLO algorithm, on the other hand, requires a threshold that determines how high the probability must be that a detected object belongs to a certain class for the object to be output. In this evaluation, a threshold of 0.1 was chosen, since a lower value also increases the number of objects detected incorrectly.

Table 4 Overall evaluation of the object detection with MobilNetSSD and YOLO

	MobilNetSSD	**YOLO**
Precision	84,41%	80,20%
Recall	45,64%	28,93%
F-Measure	59,25%	42,52%

In addition to the evaluation of the object detection, the computing time was also analysed. First, the computing time of the MobilNetSSD algorithm was compared with the computing time of the YOLO algorithm. It was then determined whether the computing time behaved differently when different hardware was used. For this purpose, the analysis was performed on a virtual machine on the one hand and on the Raspberry Pi3 on the other. The respective computing times in seconds are shown in Table 5.

Table 5 Computing time for 200 images

	Virtual Machine		**Raspberry Pi3**	
	MobilNetSSD	YOLO	MobilNetSSD	YOLO
Computing Time (sec)	19,82	41,91	21,73	58,26

IV. Conclusions

The aim of this paper was to investigate a system for object detection and classification, which works despite low computing power with low computing time. For the implementation of this system a cost-effective Raspberry Pi3 with the operating system Raspbian Stretch was used. To significantly reduce the computing time for the classification of objects, the neural network was outsourced to a neural computer stick from the company Movidius. The costs of this system amount to about 120 Euro, which is significantly lower compared to other systems that use the computing power of GPUs. The required software, such as the OpenCV library for object detection or the Movidius SDK, is open source and therefore free of charge.

After the system was put together, the performance was examined in more detail. The two known algorithms YOLO and MobileNet were used and compared. The MSCOCO data set, which contains 80,000 images and 90 classes, was used for this purpose. First, the data set was scaled to the 20 classes to which the models of YOLO and MobileNet were trained, and then the two hundred images were randomly selected from the data set.

The analysis showed that the precision of both algorithms was high at over 80%, but the recall was low at under 50%. It should be noted that the MobilNetSSD algorithm with an F-Measure of 59.25% performed about 17% better than the YOLO algorithm.

In addition, the hardware independence of the Movidius stick was checked. In addition to the Raspberry Pi3, the algorithms were also executed on a virtual machine with Ubuntu 16.04. The algorithms were also tested on a virtual machine with Ubuntu 16.04. No differences could be found between the two systems in object detection and classification. In addition to the analysis of the object detection and classification, the computing time was also examined. The Raspberry Pi3 took about 9 seconds longer than the virtual machine with both algorithms. The possible reason for the time discrepancy could be the missing USB 3.0 interface on the Raspberry Pi3.

Following on from this work, it would make sense to train a model for individual classes yourself. For example, this could be a model that is specially trained to recognize vehicles to count them and draw conclusions about the traffic situation in cities.

References

[1] https://movidius.github.io/ncsdk/ncs.html, [last visited 12.02.2019]

[2] https://github.com/gudovskiy/yoloNCS, [last visited 12.02.2019]

[3] https://github.com/movidius/ncappzoo.git, [last visited 12.02.2019]

[4] OpenCV team, „OpenCV: About," [Online]. Available: https://opencv.org/about.html. [last visited 31.3.2018]

[5] Movidius, „Enabling Machine Intelligence at High Performance & Low Power," [Online]. Available: https://www.movidius.com/technology [last visitied 19.05. 018]

[6] J. Redmon, S. Divvala, R. Girshick und A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection," *CoRR,* 2016

[7] S. Ren, K. He, R. Girshick, J. Sun Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Available: https://arxiv.org/abs/1506.01497 [last visited 14.4.2019]

[8] C. Szegedy , W. Liu , Y. Jia , P. Sermanet , S. Reed , D. Anguelov , D. Erhan , V. Vanhoucke , A. Rabinovich. Going Deeper with Convolutions. Available: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43022.pdf [last visited 14.4.2019]

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu und A. Berg, „SSD: Single Shot MultiBox Detector," in *European conference on computer vision*, 2016

[10] K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Available: https://arxiv.org/abs/1409.1556. [Last visited 14.4.2019]

[11] M.Schrittesser. Objekterkennung mit Raspberry Pi3 und Movidius Neuronal Netzwerk Stick. Bachelor Arbeit 2, FH Kärnten. Juni 2018, unpublished