Recognition of Various Objects from a Certain Categorical Set in Real Time Using Deep Convolutional Neural Networks

Alexander Driaba
Bachelor's student.
Volgograd, Russia
casha.dryaba@mail.ru

Aleksei Gordeev
Supervisor.
Volgograd, Russia
alexurgor2008@gmail.com

Vladimir Klyachin
Ph.D. Physical and mathematical sciences.
Volgograd, Russia
klyachin.va@volsu.ru

Institute of Mathematics and Informational Technologies
Volgograd State University

Abstract

When creating a mobile autonomous robot, it became necessary to solve the problem of recognizing a certain categorical set of objects on the Raspberry Pi-3 Model B board with Intel Neural Compute Stick 2. The article discusses one of the approaches to solving this problem using a neural network based on MobileNet-SSD architecture.
The problem has a solution, subject to the availability of the necessary equipment.

1 Introduction

The main goal of the project is the implementation of computer vision systems in an autonomous mobile robot with a camera. When moving, the robot must be able to recognize people and other environmental objects in real time. As an approach to solving this problem, deep neural networks were chosen as one of the popular methods for solving recognition problems.

A similar project is the self-driving robot Ben, developed by Intel©. Their goal was to reduce the number of road accidents by creating a plausible simulation of traffic. In our work, first of all, we consider the problem of detecting various dangerous objects, such as explosives or gaps on the road.

2 Deep convolution neural network

The deep neural network we used for our detection and recognition task is MobileNet-SSD, which pre-trained dataset we can retrieve from internet.

In: S. Hölldobler, A. Malikov (eds.): Proceedings of the YSIP-3 Workshop, Stavropol and Arkhyz, Russian Federation, 17-09-2019–20-09-2019, published at http://ceur-ws.org

2.1 SSD

SSD (Single Shot MultiBox Detector) - framework which purposes is localization(tracking, bounding boxes) and classification at once. The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections [10]. Single Shot means that both localization and detection take place in one pass during recognition, the network simply "looks" once at the image.

SSD starts from the core network. This network is pre-trained on a large data set, such as ImageNet, which allows it to learn a rich set of different features. The core network is used to transfer training, spread the input image to a predetermined layer, obtain an object map, and then move this map forward to the object detection layers.

2.2 Multibox

The term MultiBox means that an SSD can recognize objects of different classes, even if their bounding boxes are overlapped. This is possible thanks to the priors system. Priors are fixed-size bounding boxes whose sizes are pre-calculated based on the size and position of the ground-truth bounding boxes.

The priors are selected such that their Intersection over Union (IoU) is greater than 50% with ground-truth boxes. In order not to need to create a MultiBox predictor, fixed priors are used. In other words, when the image is divided into cells when progresses through a convolutional network layers, for each cell tested several standard bounding boxes of different aspect ratios.

In the figure 1 you can see an example of the generated feature maps with different cell sizes. The objects successfully recognized within the fixed frames are highlighted in color. An object that cannot be recognized with a given cell size can be found with a lower number of partitions.

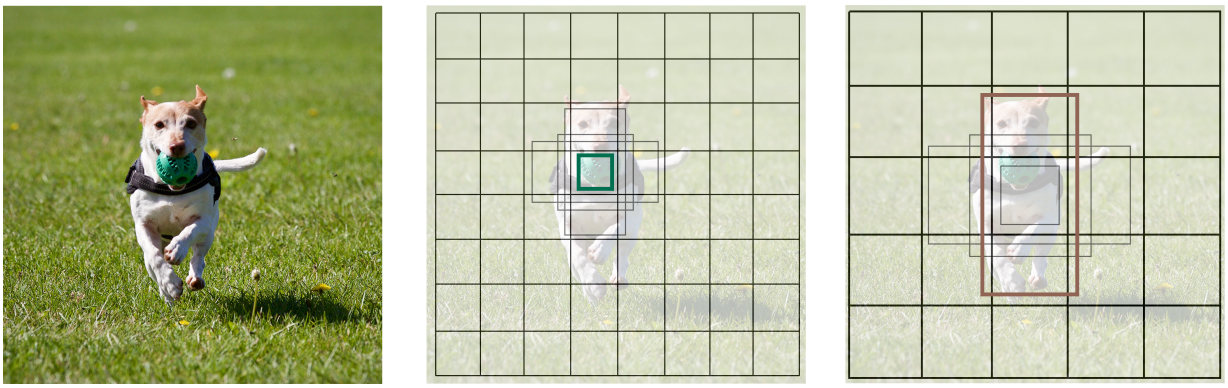


Figure 1: Feature maps example

Just like in Faster R-CNN the bounding box offsets are predicted. For each bounding box, the probability of all class labels within the region is also computing. The calculation of the probabilities of all the bounding boxes and a wide range of classes allows us to detect potentially overlapping objects.

When training SSD we use loss function of the MultiBox algorithm, which includes categorical cross-entropy loss and smooth L1-loss for location loss.

The SSD framework also includes a concept of hard-negative mining to increase training accuracy. During the training process, cells that have a low IoU with ground-truth objects are treated as negative examples.

2.3 MobileNet

As a base layer we used is MobileNet. As the base network, we could choose VGG or ResNet architecture, however, we stopped at MobileNet, as the fastest, albeit at the cost of accuracy. MobileNetV1 is an architecture from Google, which is known primarily for its smaller set of parameters and less network complexity, achieved by fewer addition and multiplication operations. MobileNet is a convolutional neural network architecture that applied on devices with limited computing power [8].

The overall architecture of the neural network is presented in the figure 2. We use MobileNet until conv_6, and then we separate all the other convolution layers. Each feature map is connected to the last recognition layer, which allows the detection and localization of objects of different scales.

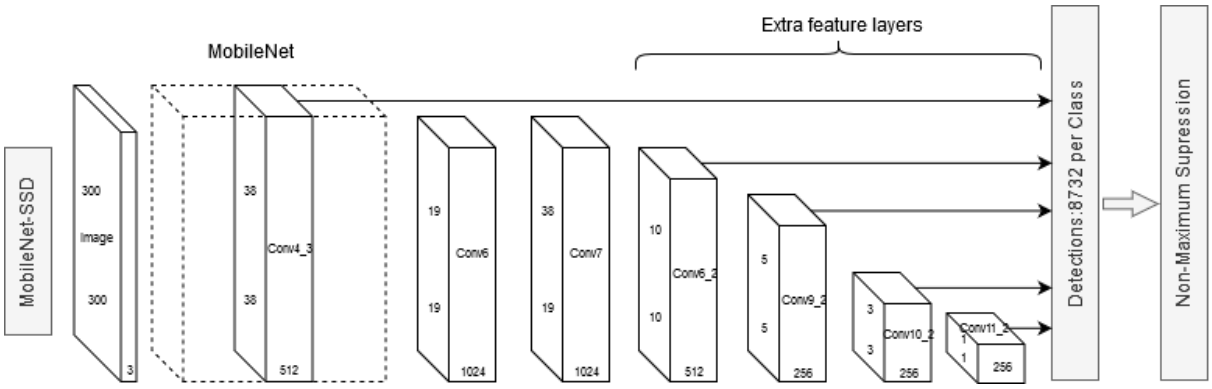


Figure 2: MobileNet-SSD Architecture

3 Preparing the dataset

In our work, the goal was to recognize the boundaries and labels of the following categories of objects:

- aeroplane, bicycle, boat, bus, car, motorbike, train
- bird, cat, cow, dog, horse, sheep
- bottle, chair, diningtable, pottedplant, sofa, tvmonitor
- person, background, face

The data set that we used was collected by us and tagged using the dlib library. Images were tagged with the imglab tool, which is included in this library. One of the advantages of this tool is the ability to mark some objects as "ignored". If we used dlib to train the detector, dlib would exclude such areas from the training set. Examples of such areas are too small objects or objects that cannot be separated from each other. In such cases, our model will not be able to isolate patterns and, accordingly, learn.

4 Training

To train the model on our dataset, we use the TensorFlow Object Detection API (or TFOD API). Since the TFOD API does not know how to separate the "ignored" areas from the usual ones, we converted the training and testing files to a format that API understands removing all ignored areas. As a result, we received several .record files and a file containing all our categories.

Also, besides the dataset, we needed a configuration file, which we took from the TensorFlow Detection Model Zoo, specifically the architecture "ssd_mobilenet_v1_coco.config"[11].

After that, we conducted the training procedure for our model, the preliminary results of which can be seen in the figure 3. The left graph shows the change in the loss function, while the right graph shows the accuracy of our model.

5 SSD issues and limits

It is worth noting that using the SSD framework, we encountered two main problems. The first is based on the foundations of the architecture itself - SSD does poorly with small objects. You can solve this problem by increasing the resolution of the input images, however, this will dramatically reduce the speed of our network.

The second problem can arise for similar objects, such as tables and chairs in the background, such objects can be confusing for the TFOD API due to hard-negative mining, in which cells with these objects can be marked as negative patterns. This is especially aggravated by the "ignored" areas that we discarded at the stage of preparation of the dataset.

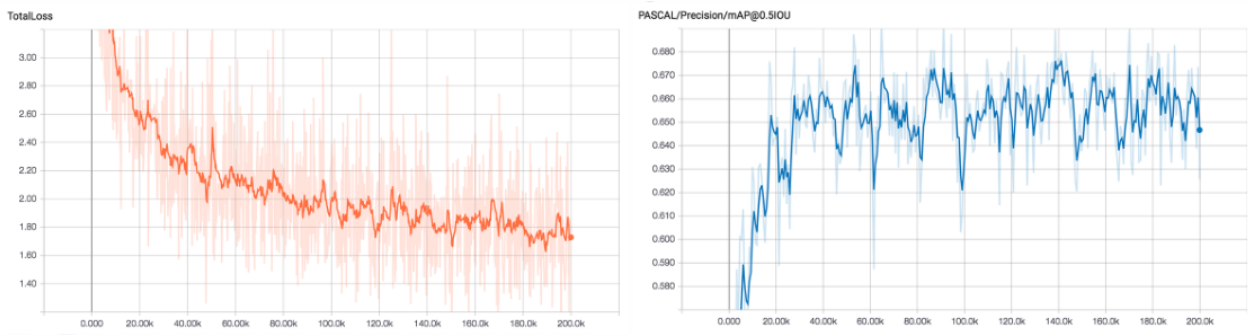


Figure 3: Loss and precision

6 Equipment

Due to the small size of the robot, the equipment was selected according to two criteria:
- It should be small.
- It should consume as little energy as possible.
These 2 criteria are fulfilled for the Raspberry Pi 3b board chosen by us. By itself, the Raspberry Pi 3b board has a 64-bit 4-core ARM processor with a clock frequency of 1.2 GHz [1].

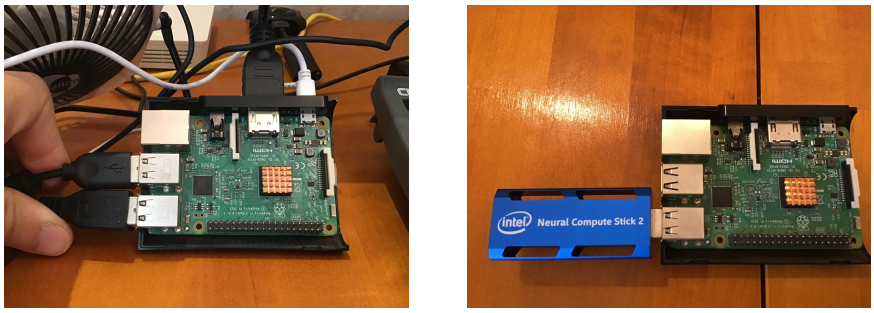
However, one board is not enough, if you try to deploy the network on a bare board, then the number of FPS will not be enough for the robot to recognize objects in real time (less than 1).

The lack of performance forces us to take Intel Neural Compute Stick 2 in addition to the board.

Intel Neural Compute Stick - this is what the people called it "neural stick". Using this device significantly improves the performance of the board and allows you to analyze incoming images from the camera in real time. The device is based on the Movidius Myriad X Vision Processing Unit (VPU), a specialized chip containing 16 general-purpose cores and components accelerating the process of image recognition by neural networks [2]. The choice fell on the Myriad chip (used in the stick) primarily because of its relative cheapness and, as already mentioned, low power consumption.

We will receive images using a Logitech C310 USB camera.

The figure 4 presents images of our equipment.



(a) Raspberry Pi 3 Model B (b) Neural Compute Stick

Figure 4: Equipment screenshots

7 Software

To work with a neural stick, we must install and configure the OpenVINO Toolkit. Raspberry Pi 3 Model B running the Raspbian 19 operating system. To ensure the interaction between Raspberry Pi and Intel Neural Compute Stick 2, we will use the OpenVINO toolkit, which provides an API for interacting with a VPU[3].

8 OpenVINO toolkit

The OpenVINO toolkit uses pre-trained models from various deep learning frameworks to optimize them for specific Intel© hardware (specifically for the VPU in our case).

To successfully use a pre-trained tensorflow or caffe models it is necessary to convert it from our legacy deep learning framework format into special OpenVINO format for optimization, which consists of bin and xml files:
-Bin file format is used to store frozen weights of all levels of network nets.
-An xml file stores a serialized map reflecting the structure of the model, which determines exactly how weights should be interact with each other [5].

Model Optimizer takes the finished model and form the resulting neural depth more shallow, throwing away extra layers, while maintaining relative accuracy. Further, this optimized model is loaded into the Inference Engine, which already has all the necessary tools and optimizations for execution on various types of devices.

Next, OpenVINO passes the optimized model to the Inference Engine.

Inference Engine is a framework with set of classes and APIs functions used for inference of neural networks operation results. The framework provides an API for reading intermediate presentation, setting
input and output formats, and executing the optimized model on specific devices, such as FP32 CPU's, Intel© HD Graphics, Myriad VPU etc. [6].

9 Application structure

The inference cycle is shown in the figure 5. The video signal comes from the camera and is divided into frames. Each frame is pre-processed using the OpenCV library before it gets into the plug-in. The Inference Engine plug-in allows us to make calculations on Myriad. After neural network processing (detection, presence or absence, tracking with the appropriate frame and signature), the data is used to overlay the frame on the original image, which is then displayed on the screen. The entire process of the application work can be called an "inference loop".

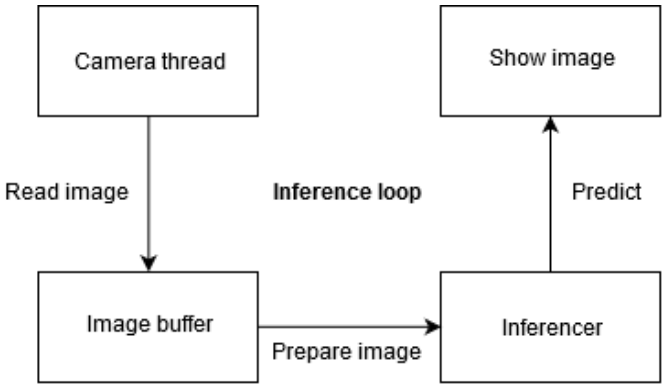


Figure 5: The general scheme of the application

10 Results

In our examples, the bottle was best recognized if it was located close to the camera, the chair at times disappeared from the objects of recognition, while the sofa all the time was determined (although it was partially in the frame), the monitor was recognized best of all, there were no problems with it.
At a resolution of 320x240 charge produces stable 30 FPS. By increasing the resolution to 1280x720 (which is HD) performance drops to 9 FPS.
For the normal functioning of the robot, with obtaining images in real time in good resolution, one INCS2 stick is not enough, it is best to use from 2 to 4 sticks. Only then will it be possible to achieve an acceptable speed in real time recognising [9].

In figures 6-8, you can see the result of the network. The network recognizes a plastic bottle, tablet, sofa and chair.

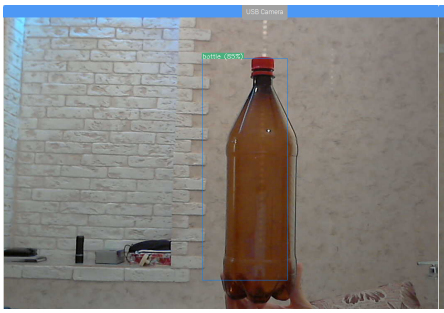


Figure 6: Bottle

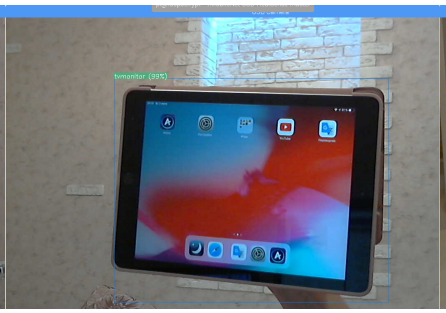


Figure 7: Tablet

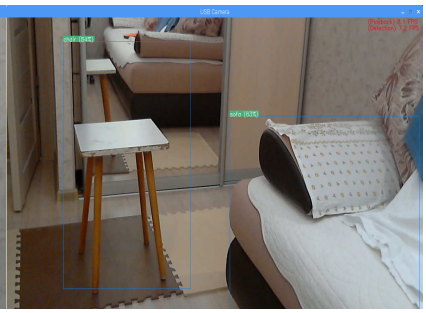


Figure 8: Chair and sofa

11 Discussion

By the results, we found that our object detection and recognition system used MobileNet-SSD on Raspberry Pi with INCS 2 allows us to recognize the most standard and typical objects with precision. And if we want to rise the processing speed then need to use to use at least 4 sticks, which we going to work with in the near future.

References

[1] Raspberry Pi 3: Specs, benchmarks & testing
URL: https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[2] Jean-Luc Aufranc. Intel Neural Compute Stick 2 with Myriad X VPU Finally Announced
URL: https://www.cnx-software.com/2018/11/14/intel-neural-compute-stick-2-myriad-x-vpu/

[3] Neal Smith. Get started with Neural Compute Stick
URL: https://software.intel.com/ru-ru/articles/get-started-with-neural-compute-stick

[4] Introduction to Intel Deep Learning Deployment Toolkit
URL: http://docs.openvinotoolkit.org/latest/_docs_IE_DG_Introduction.html

[5] Model Optimizer Developer Guide URL:
https://docs.openvinotoolkit.org/latest/_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html

[6] Inference Engine Developer Guide URL:
https://docs.openvinotoolkit.org/latest/_docs_IE_DG_Deep_Learning_Inference_Engine_DevGuide.html

[7] Wei Liu et al. "SSD: Single Shot MultiBox Detector"
URL: https://arxiv.org/abs/1512.02325

[8] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications"
URL: https://arxiv.org/abs/1704.04861

[9] Katsuya Hyodo. MobileNet-SSD-RealSense Library URL:
https://qiita.com/PINTO/items/94d5557fca9911cc892d#24-fps-boost-raspberrypi3-with-four-neu

[10] Adrian Rosebrock. Non-Maximum Suppression for Object Detection in Python
URL: https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-pyt

[11] Tensorflow detection model zoo URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md