Ecovacs Robotics: the AI robotic vacuum cleaner powered by TensorFlow

2024년 8월 27일 화요일 오전 8:43

January 07, 2020

A guest post by Liang Bao, Chengqi Lv, from Ecovacs Robotics AI Department.

Translated by Ziqian Xu, from Ecovacs Robotics Technology Development Department.

Robotic vacuum cleaners are a main products of Ecovacs Robotics, which manufactures in-home robotic appliances. In this article, we share our journey of implementing and deploying a TensorFlow Lite model that helps the robot detect and avoid obstacles.

Technical background and aims

Fig. 1 Ecovacs robot vacuum cleaner

Robot vacuum cleaner is a smart home appliance which can clean the floor automatically. The main features of a robotic vacuum cleaner are environment mapping, anti-drop, obstacle climbing and auto-recharge. Technically speaking, it uses simultaneous localization and mapping (SLAM) using the laser distance sensor (LDS) on the top of its body, its bumpers, and an infrared sensor to detect and avoid obstacles in its path.

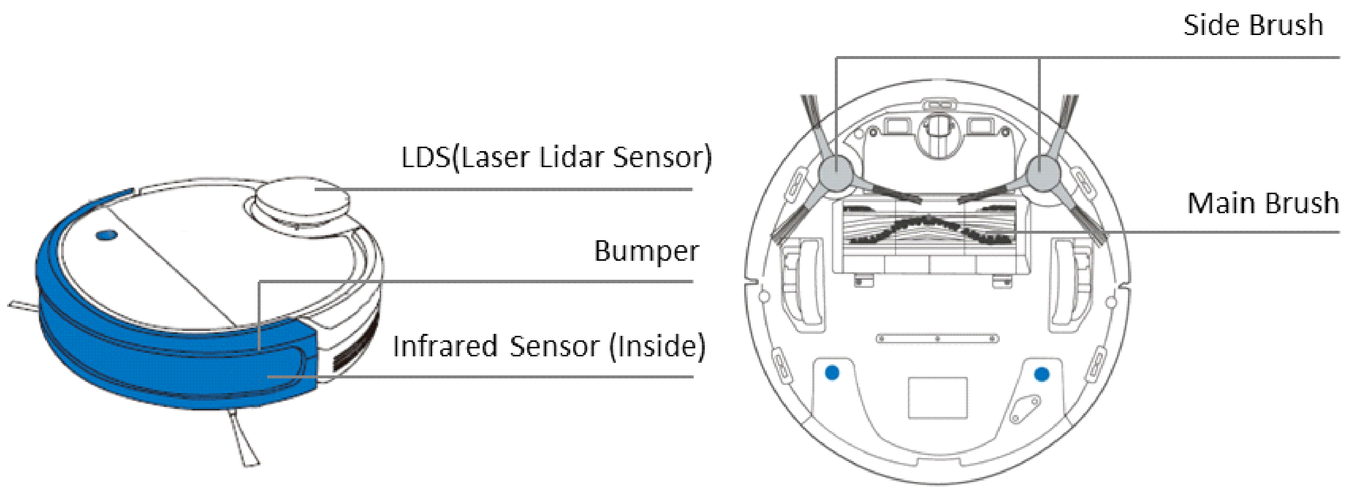


Fig. 2 Structure diagram of the robotic vacuum cleaner

Although SLAM technology for robotic vacuum cleaners is relatively mature, the robots cannot detect small objects on the floor. This is mainly because the LDS, bumpers and infrared sensors are not able to detect the small or soft obstacles effectively which causes three problems:

1. The robot pushes slippers and socks everywhere. When you wake up, you cannot find them.
2. Headphone cables and other wires can be entangled with the side brush causing the robot to get trapped easily.
3. Wipes and soft mats can be entangled with the main brush which causes the robot to not work properly.

The problems above force users to do the pre-cleaning (such as manually putting the shoes, socks and wires in order) before using the robotic vacuum cleaner, making the cleaning process a hassle. Therefore, we hope to use computer vision algorithms to detect these obstacles by adding a camera in front of the robot and find a way to avoid the obstacles and save users time.

Our main target detection objects are chargers, dustbins, shoes, slippers, dishcloths, mats, and wires. Here are the main problems we have come across during research: lack of data, complex indoor scenes, variety of object appearance, too many soft or small objects, and the limited computing power due to high production cost. These complex and varying problems make it impossible to detect the objects robustly with a rule-based algorithm. Inspired by the excellent performance of deep neural networks in ImageNet Large Scale Recognition Challenge, we decided to use deep neural networks to detect the objects.

Our team has developed an object detection model based on TensorFlow. The reasons that we chose TensorFlow among other deep learning frameworks are as follows.

1. It is a mature framework with a comprehensive ecosystem of documentations and

community resources, and it has attracted a large number of developers.

2. There is a library called Object Detection API that has implemented many classic object detection and segmentation models. Thus, it is very convenient to build new models based on this library.
3. TensorFlow-based models can be easily deployed to production. TensorFlow Lite makes it more convenient for developers to run the models on edge devices.

This article will generally introduce how we developed the model and how we implemented it into the robotic vacuum cleaners.

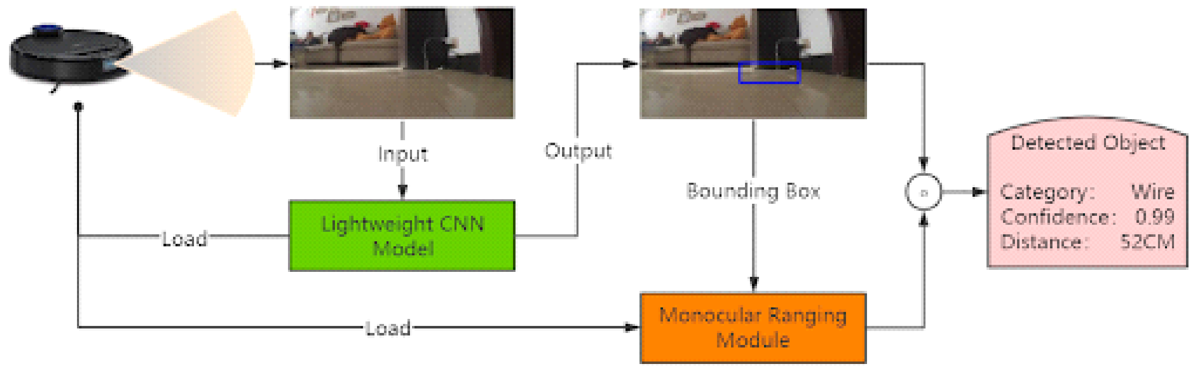


Fig. 3 Overall architecture diagram of the algorithm

Data collection and annotation

Building a deep learning model without large amounts of data is like making a house without blueprints. Due to the unique ground-view perspective and uncommon object categories, we cannot find any public dataset which fit our needs. Therefore, we first cooperated with many institutions to collect data from all over the world. Although manual data collection took up a lot of time and energy, it was worthwhile to create a robust and accurate dataset. During our data collection, we collected data in different seasons, at different times, and from different places. For example, overexposed or backlighting images can be collected during the daytime, while low light images are mostly collected at night.

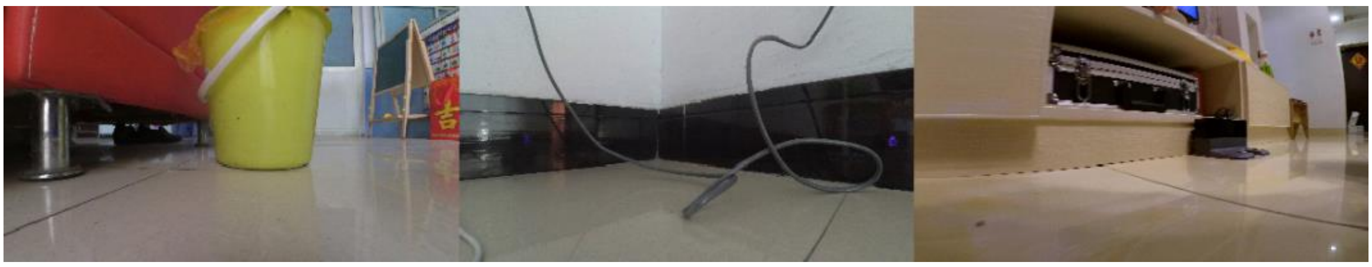


Fig. 4 Image examples of our data from ground view perspective

Data cleansing was then used to improve the quality of the collected data. This is necessary because many images are similar to each other since the camera takes photos continuously. Adding all of these images into our dataset will result in larger memory consumption and longer training process. It may also cause overfitting and higher generalization error. Deleting these images will not affect the accuracy of the model if applying data augmentation. After data cleansing, LabelImg was then used to annotate the data.

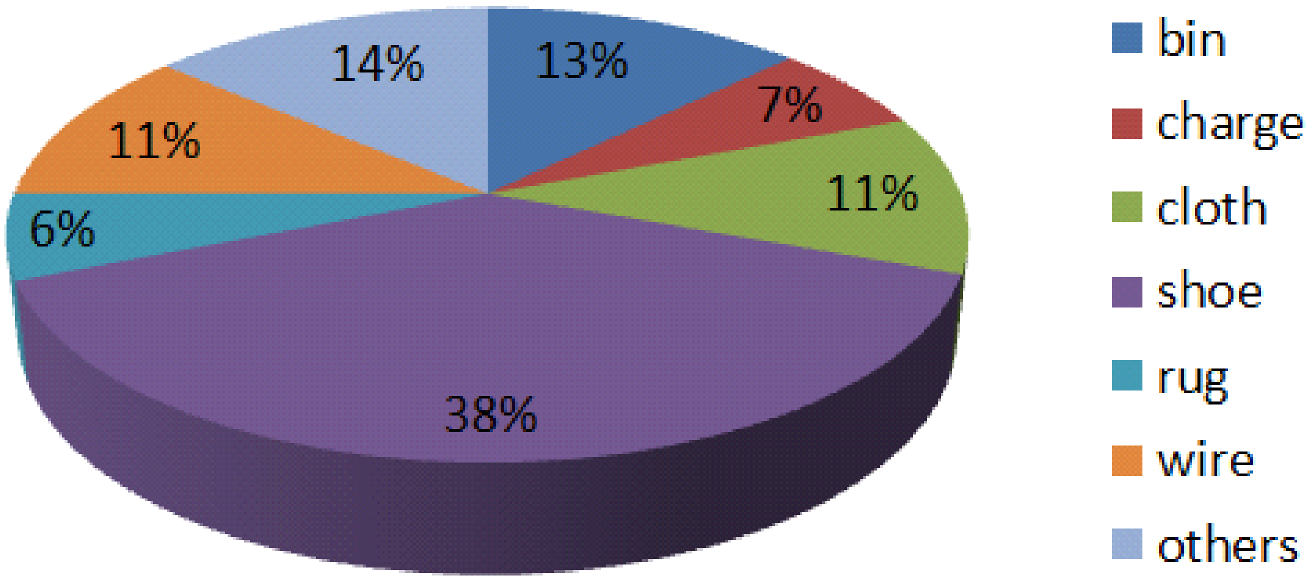
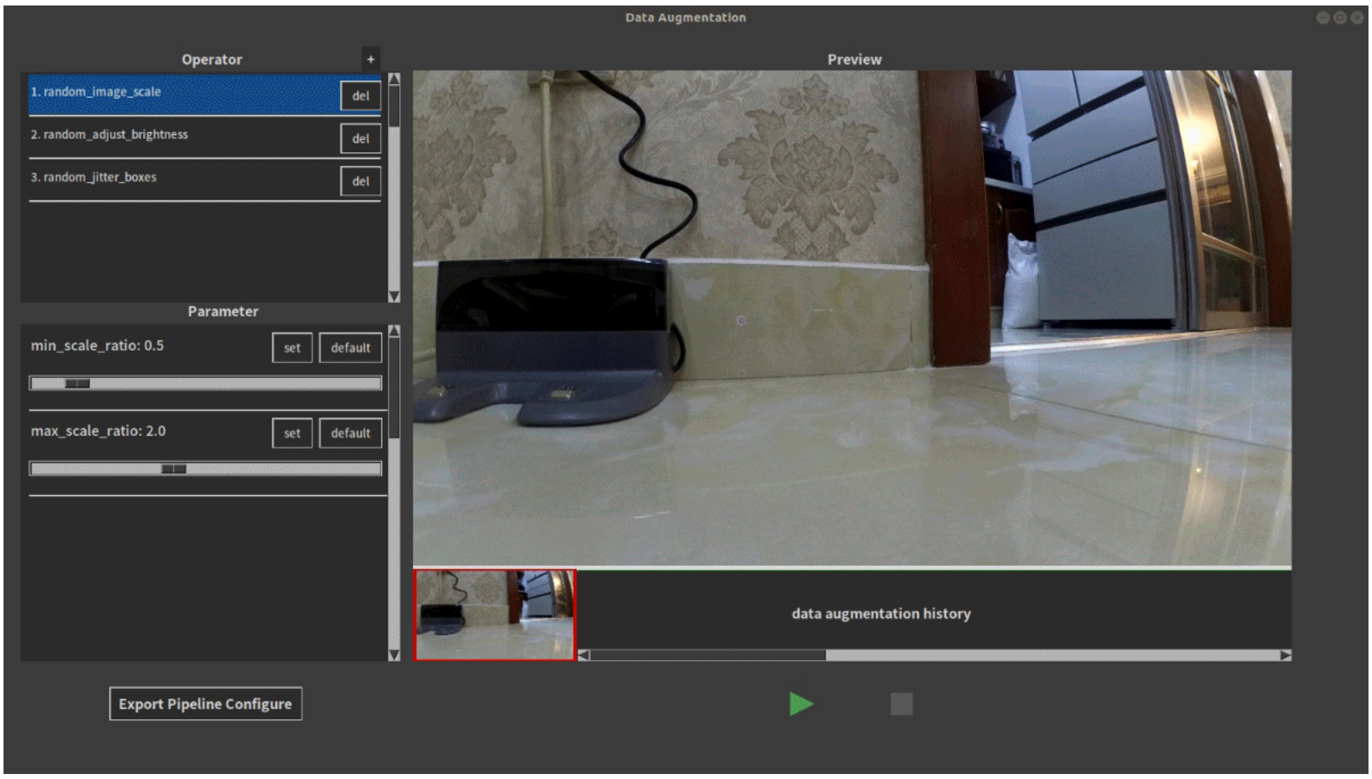


Fig. 5 Diagram of data distribution

Fig. 5 shows the distribution of the annotated data. Shoes accounts for the largest proportion and this category achieves a higher accuracy.

Data Augmentation

In developing the model, data augmentation played a significant role. For example, by changing the hue values of the image, we can simulate floors and mats of various colors. As mentioned before, we built our model based on the Object Detection API. This library includes a Python script which already implemented many common data augmentation methods and a corresponding protobuf file which has defined the hyperparameters for each method. Additionally, we also implemented some new methods based on our needs, and we developed a visualization tool where developers can choose different data augmentation functions and hyperparameters, preview their image, and export the modified configuration file with one single click.



1. random_image_scale
2. random_adjust_brightness
3. random_jitter_boxes

min_scale_ratio: 0.5

max_scale_ratio: 2.0

Fig. 6 Visualization tool of data augmentation for Object Detection API

Structure of our network

There are two common object detectors: the one-stage detector and the two-stage detector. Although the two-stage detector usually achieves higher accuracy, it requires massive computing power. In order to run our model on the robotic vacuum cleaner, we used the one-stage model which was similar to SSD detector[1] (this one-stage detector uses multi feature maps to predict objects at different scales) and replaced the VGG backbone network[2] in the original SSD detector by a customized lightweight network for feature extraction.

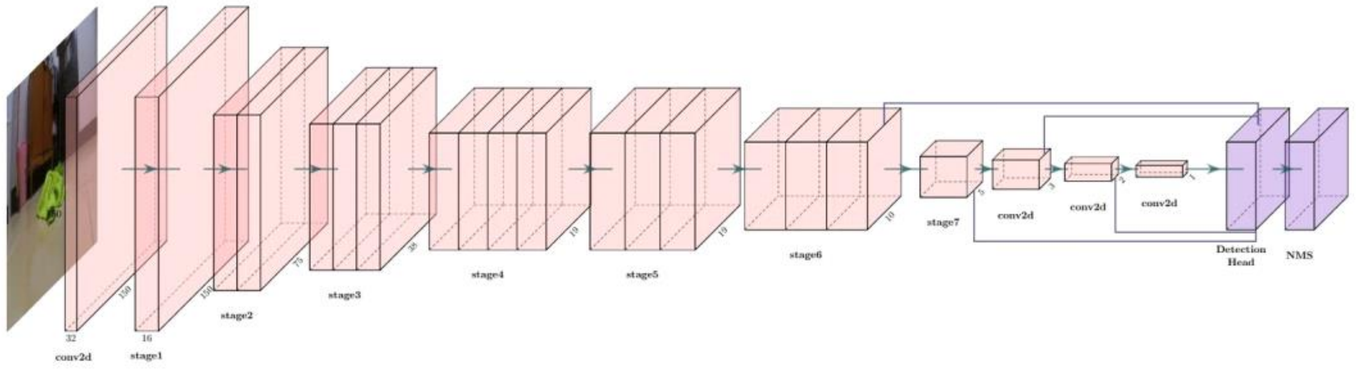


Fig. 7 Structure of the original SSD detector

We built the feature extraction backbone network based on the effective receptive field, pixel size of the object, and its size on the feature map. Fast downsampling was used to reduce the computation, and we made up for the loss of feature extraction capabilities through inverted residuals block used in MobileNetV2[3], which includes a spindle-shaped structure and a structure similar to the shortcut connection in

ResNet[4]. Also, in order to improve the recall of the small objects, we concatenated the features from shallow feature maps and deep feature maps.

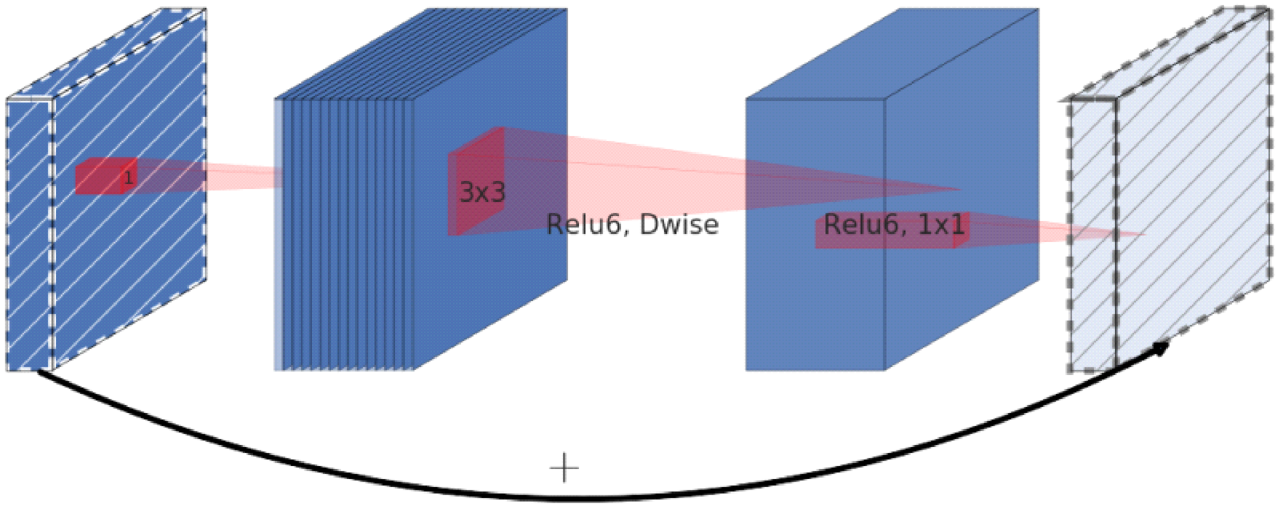


Fig. 8 Inverted residuals block used in MobileNetV2

In addition, standard convolutions are all replaced by depthwise separable convolutions in our model. Unlike standard convolutions, depthwise separable convolutions split the kernel according to the number of channels, and each channel has its own kernel. After that, the 1X1 convolution kernel is used to concatenate the features of all channels. For a 3X3 convolution kernel, the computation can be reduced to 1/9 if the standard convolution is replaced by the depthwise separable convolution. Our model uses ReLu6 as the activation function which does not require much computation. Also, ReLu6 can minimize the fixed-point constant precision loss caused by model conversion.

Anchor box optimization

When predicting bounding boxes, most traditional architectures use anchor boxes, and our model uses anchor boxes too. The two-stage detector mostly uses anchor boxes to extract region of interest (ROI) from the feature map, while the SSD detector generates the anchor box based on each pixel of the multi-feature maps in the backbone network. Every anchor is trained based on its width, height, the offset of its center, and the corresponding class.

During the development, we customized the anchor box to specific sizes for different categories. This allows the model to match more positive samples in average and to alleviate the problem of imbalance between positive and negative samples while training the model effectively. There are many ways to optimize the size and the proportion of anchor boxes.YOLO[5] used dimension clusters while we used genetic algorithm[6]. After optimization, the anchor box matches more positive samples, and the number of the anchor boxes decreases by two-thirds. Fig. 9 illustrates the average matched number of the anchor box of all categories. The number of matches for small objects (scale < 50) and large objects (scale > 250) dramatically increased after optimization.

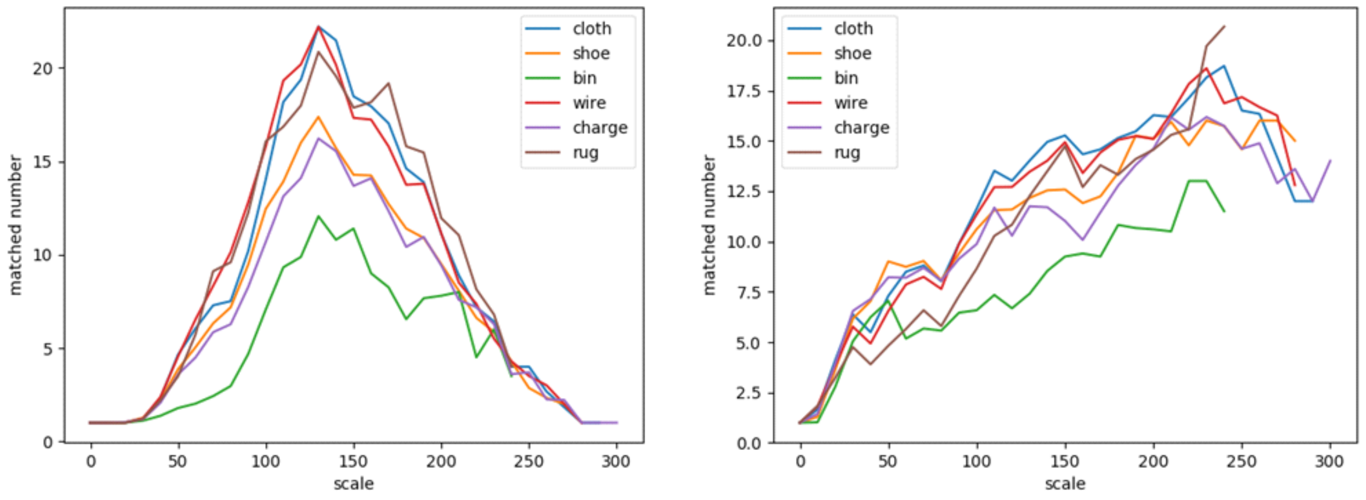


Fig. 9 Average matched number of the anchor box before and after optimization (the figure on the left hand side shows the matched number before optimization)

Loss function and model training

Based on the focal loss[7], we designed our loss function with weights to different anchor boxes. Unlike the origin focal loss, which only gives different weights to easy and hard samples, our model also weights the position. This way makes the model focuses more attention on the features of close objects and improves the accuracy. The structure of the model and training pipeline are defined by the Object Detection API: we define our model in the "/object_detection/models" module, the loss function in the "object_detection/core" module, and the anchor boxes in the "object_detection/anchor_generators" module. Then we used the "object_detection/model_main.py" script to train and test the models.

During training, we did a series of experiments and found that feature concatenation and optimization of the anchor box both positively contributed to accurate detection.

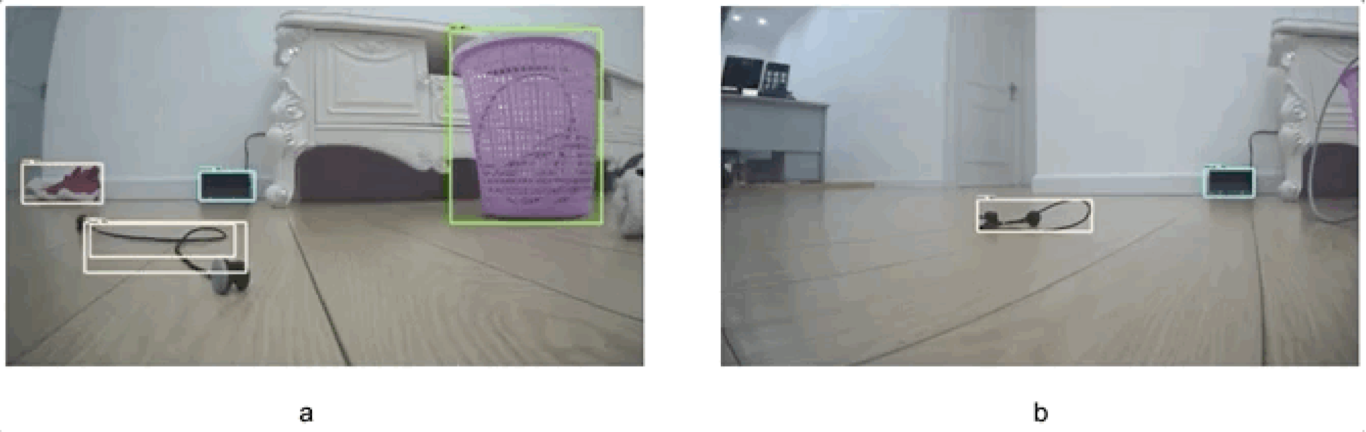


Fig. 10 How our model works (a: what our model sees; b: our model won't detect objects which are elevated from the floor)

Explanation of the model

When testing the model, usually it is hard to figure out what is happening inside the model if a misdetection occurs since the neural network is a black box. We hope to use some approaches to find the reason of the misdetection, and hence, chose Grad-CAM [8].

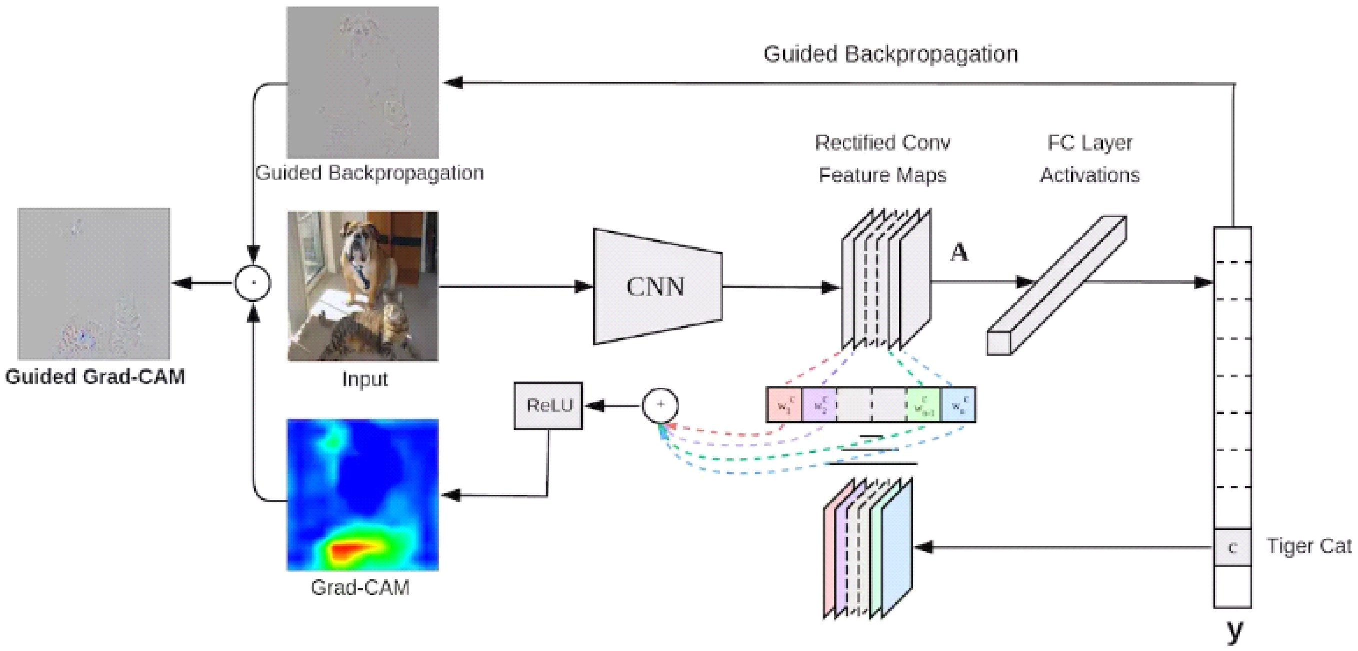


Fig. 11 Overall structure of Grad-CAM

The basic idea of Grad-CAM is to calculate the gradient of every pixel in the feature map and use it to show the contribution of this pixel to the classification. For each channel, the result of global average pooling shows the importance of the channel. The advantage of this method is that it does not require re-training. However, Grad-CAM needed to be adjusted for our detection model. For anchor based detector, global average pooling cannot be used to calculate the weight of each channel since each anchor box is multiplied by a 3X3 convolution kernel to predict its class. Instead, we use the value of the corresponding anchor box to avoid the influence from other anchor boxes.

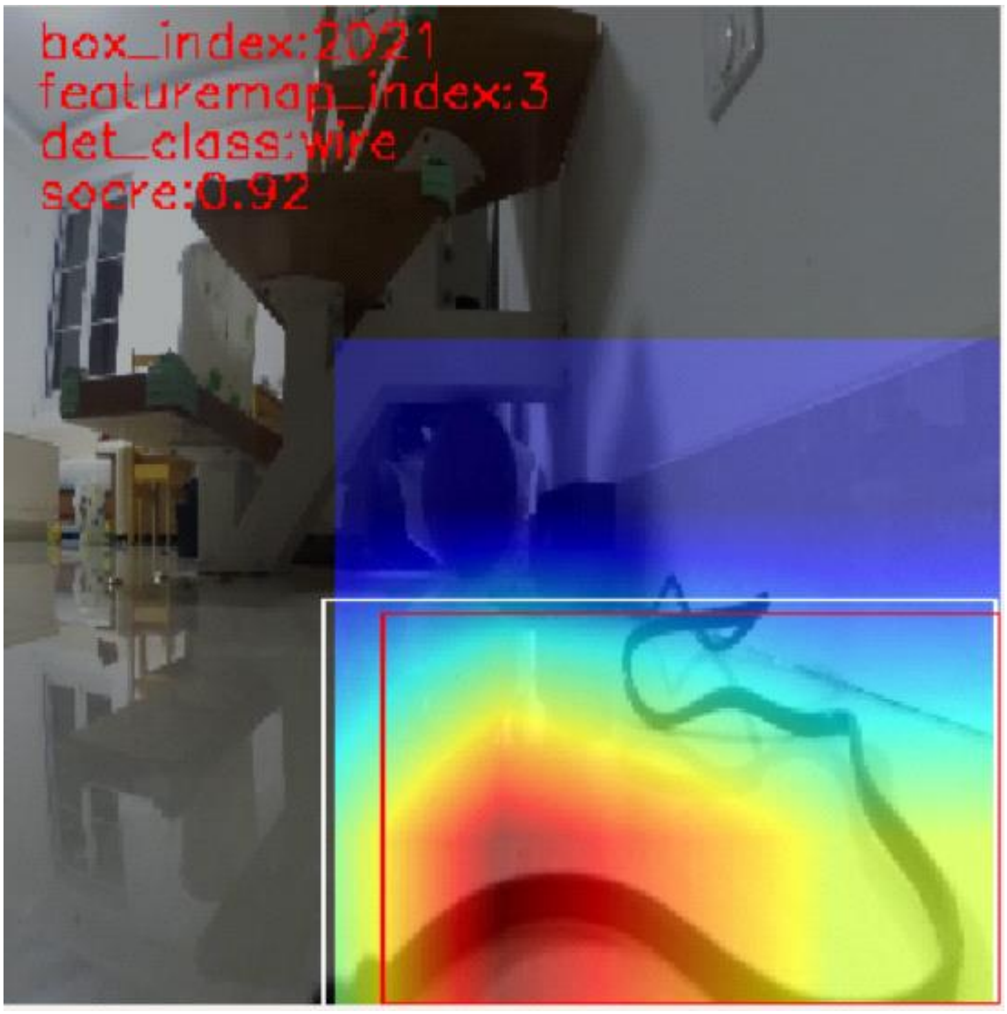


Fig. 12 Grad-CAM visual explanations

Fig. 12 shows the contribution of every pixel in the 2022nd anchor box (which is on the fourth feature map) to the class wire. Red regions indicate high score for the corresponding class, and blue regions indicate a low score for the corresponding class. In this figure, the part where the wire touches the ground is painted red, which means this region plays a key role in detecting the wire. On the contrary, although the part above the ground is within the sampling range of the 3X3 convolution sampling region, it is painted blue and indicates that this part contribute little to detecting the wire. According to the visual explanation of our model, we know that this seems very reasonable.

Model conversion and acceleration

The robotic vacuum cleaner works at home where privacy is a priority. In order to protect users' privacy and prevent data leakage, we need to deploy the neural network model on the robot in offline mode and use hardware accelerator to speed up inference as fast as we can.

After trained our model in TensorFlow, we converted it to TensorFlow Lite for deployment on the robot vacuum cleaner. The robot's motherboard runs Linux OS. It has 4*Cortex-A17 CPU cores (only one of them can be used for AI module) and a

Mali-700 GPU mainly for video processing.
We split the trained model into three parts: pre-processing, deep learning model and post-processing. Pre-processing applies several pre-processing techniques on the input image such as size normalization. The deep learning model consists CNN ops, and we used TensorFlow Lite to run the model on CPU. The post-processing step optimizes the output of the model, including non-maximum suppression (NMS).
In order to further increase the speed, we let the hardware in-camera ISP do the pre-processing step. Also, with the help of OpenCL, part of the CNN ops in the Lite model is deployed to the GPU. These two steps improved the inference speed by 30%, which enabled the robot to meet the minimum latency requirement for avoiding obstacles. In post-processing, we set different NMS IoU thresholds for different categories. For example, the IoU threshold for wire is set to be a bit larger in order to increase the number of the output boxes of wire and recall.

Ranging and obstacle avoidance

After the CNN model detects the objects in the image, we can get the accurate position of certain objects based on the bounding box. Given the position in the image plane and the parameters of the camera, the position of the object in the coordinate system can be calculated using the pinhole camera model.

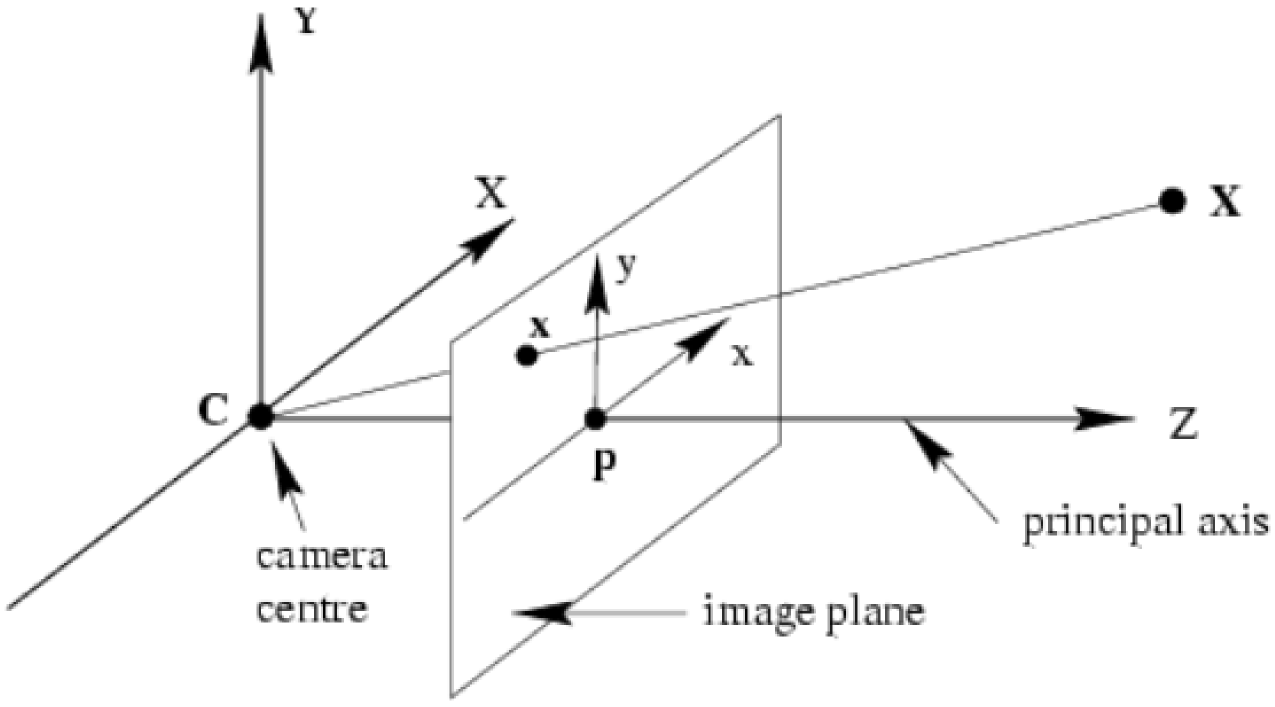


Fig. 13 Pinhole camera model

After camera calibration, we can measure the distance between the objects and the robot. Through this, the robot is able to smartly avoid obstacles.

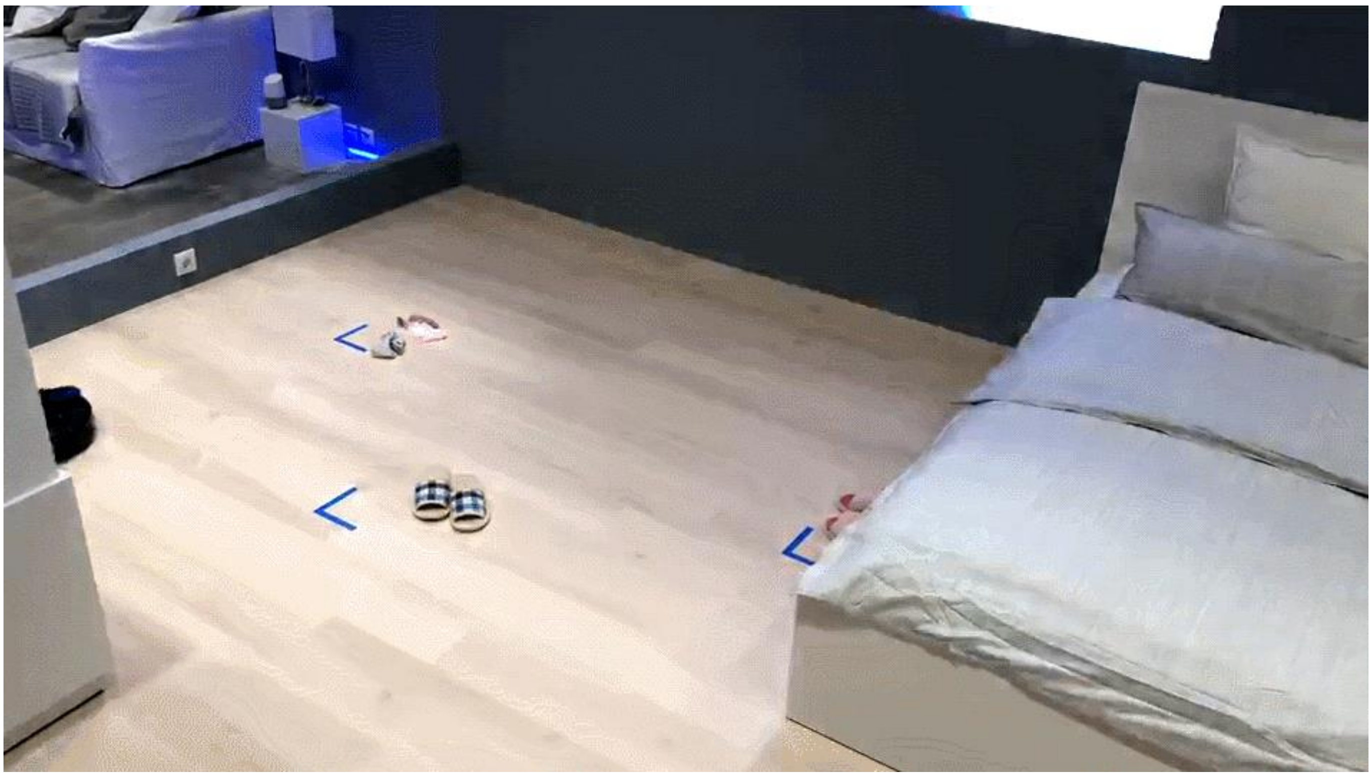


Fig. 14 DG70 at IFA 2018 in Germany

About Ecovacs

Ecovacs Robotics, as a global technology company, aims to help users live a better life with robots. The company has launched DEEBOT (floor robotic vacuum cleaner), WINBOT (window robotic vacuum cleaner), ATMOBOT (air purifier robot goes mobile), and BENEBOT (shopping assistance robot). Ecovacs robots are now available in over 60 countries, and have won good reputation around the world.

References

【1】Wei Liu, Dragomir Anguelov, (n.d.),SSD: Single Shot MultiBox Detector

【2】Karen Simonyan, Andrew Zisserman,VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

【3】Mark Sandler, Andrew Howard,(n.d.),MobileNetV2: Inverted Residuals and Linear Bottlenecks

【4】Kaiming He, Xiangyu Zhang,(n.d.),Deep Residual Learning for Image Recognition

【5】Redmon J., (2016), YOLO9000: Better, Faster, Stronger

【6】Sastry K., Goldberg D. and Kendall G., (n.d.), Genetic Algorithms

【7】Tsung-Yi Lin, Priya Goyal, (n.d.),Focal Loss for Dense Object Detection

【8】Ramprasaath R.Selvaraju, Michael Cogswell, (n.d.), Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization

출처: <https://blog.tensorflow.org/2020/01/ecovacs-robotics-ai-robotic-vacuum.html>