Real robot place recognition using Convolutional Neural Network (CNN) and ROS

Andrea Bennati
USI - Università della Svizzera Italiana
Lugano, Switzerland
bennaa@usi.ch

Alessio Della Libera
USI - Università della Svizzera Italiana
Lugano, Switzerland
dellaa@usi.ch

***Abstract*—The ability of identify a known place is a simple task for a human being. Few years ago, this task would have been impossible to accomplish by a machine. Recently, machine and deep learning received a lot of attention both from Industry and Academic Research institutes. Using these technique also a machine can replicate that task with significant results. We decided to adapt one deep learning model, a Convolutional Neural Network (CNN), to let a simple robot, the Mighty Thymio, be able to recognize the place in which it is located.**

***Index Terms*—robotics, machine learning, deep learning, Convolutional Neural Network (CNN)**

I. Introduction

The ability of identify a place known is a simple task for a human being. A way to emulate this ability in a machine could be to train a Convolutional Neural Network (CNN) such that the machine can identify the place in which it is located through images. These kind of artificial neural networks are well designed for all the tasks that require the identification of entities in images, or visual media. As for a human it is simple to recognize two completely different rooms, the same will be for a CNN. In order to recognize two different rooms that don't have any or few object in common, is a straight forward task (we don't to need the CNN at all). For example, two rooms with different color of the floor, are easily distinguishable by such model, or room with lots of different objects. In order to make the recognition task more challenging we decided to arrange two rooms as similar as possible, having just few different objects or keeping the same objects in different positions in the two rooms. The objective of this paper is to show how the Mighty Thymio (Fig 1), is able to recognize different environments. The report is structured in the following way: first we describe the environments used for our experiments; then we focus on the algorithms used with a description on how we implemented them; a section is dedicated to all the problems we encountered and how we solved them and in the last section we present our results.

II. Architecture Overview

Our implementation is written in Python, using ROS (Robotic Operating System) as a framework. It is the standard de facto for robot programming and provides a plethora of Python APIs. Our architecture was composed by:

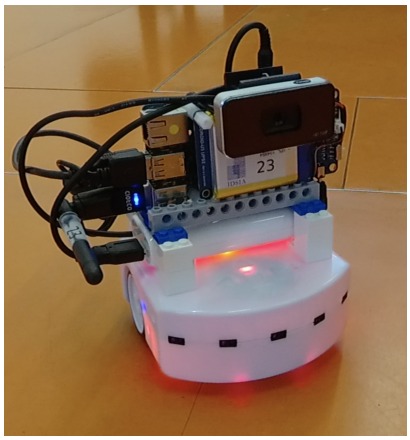

Fig. 1. Mighty Thymio

- A differential drive robot, the Mighty Thymio [1]; in our task, we used only the five front proximity sensors and the camera sensor;
- A ROS node that sends and reads messages from the Mighty Thymio, that enabled us to control the Thymio itself;
- A Convolutional Neural Network written in Python and using the Pytorch framework [2]

III. Implementation

We let the robot explore the environment and take images of it and then, once that we had enough images, we trained a CNN with them. Finally, we used this trained model to predict the place in which the Thymio is located. We did two different experiments: in the first one we used the same room (Fig 2) but with different objects from room 1 and room 2; in the second

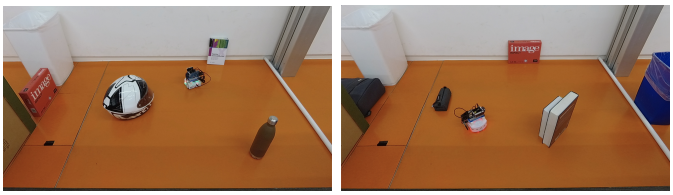

Fig. 2. Room 1 (left) and Room 2 (right)

Fig. 3. Room (left) and Entrance (right)

experiment, instead, we used two different rooms, the inside of a class room and the entrance (Fig 3). In the first experiment, we let the Thymio move randomly while predicting in which room it is. To test also a continuous prediction, in the second experiment, we implemented a function to control the robot via keyboard in order to guide it back and forth the two rooms and analyze its predictions. Once the Thymio predicts a room it changes its color (different for each room).

A. Random walking

To leave the Thymio free to explore the environment we decided to implement a random walking movement. In order to do so we created a state machine that rules the choice of the robot. The following Fig 4 represent the state machine.

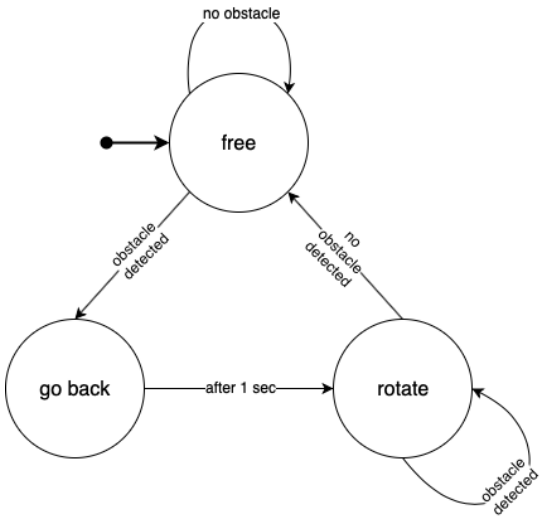

Fig. 4. State machine

Initially the robot is in the **free** state. If it has free space in front of it, it start moving straight. Through the use of the proximity sensors we are able to identify the presence of obstacles in front of the robot. When the robot detects an obstacle it enters the **go back** state in which it goes backward for one second (we chose to do this to avoid that in the rotation it may hits obstacles or walls). After one second in the **go back** state the robot enters the **rotate** state. In this state it randomly turns, both clockwise and counterclockwise depending on a random number, for a random number of seconds. The robot remain in the rotation state until the number of seconds have passed and the proximity sensors doesn't sense anything in front of the robot. Then it returns in the free state.

B. Data Collection

We collected 500 images per room in order to have enough material to train the CNN. Given the size of the rooms and the amount of objects in them we have a lot of similar images that can't allow the robot to discriminate between the two rooms, for this reason we incremented the number of images to 800. The Thymio collects and saves images every $\frac{1}{2}$ second.

C. Convolutional Neural Network

The architecture of Convolutional Neural Network used in our application is composed as follows:

- Convolutional layer
- Max Pooling layer
- Convolutional layer
- Four Fully Connected layers

We based our implementation on the official pytorch tutorial [2]; we adjust some hyperparameters and add another fully connected layer. Then we train our model, using `batch_size=16` for 10 epochs.

D. Prediction

Algorithm 1: How to predict a room

```
Data: raw pixels from camera sensor
Result: room prediction
image ← resize and convert pixels from BGR to RGB
threshold_probability
threshold_images
threshold_accuracy
limit_images
tot_predicted_images ← 0
predictions ← {'room1': 0, 'room2': 0}
while not Predicted do
    room_predicted, probability ← CNN.predict(image)
    if probability ≤ threshold_probability then
        skip image and do nothing
    else
        predictions[room_predicted] += 1
        tot_predicted_images += 1
        if tot_predicted_images > threshold_images then
            for room, count_room in predictions do
                if count_room / tot_predicted_images) >
                 threshold_accuracy then
                    return room
                end
            end
        end
        if tot_predicted_images > limit_images then
            tot_predicted_images ← 0
        end
    end
end
```

Algorithm 1 illustrates how the Thymio predicts a room. All the thresholds set in the code are based on the accuracy

of the network. If the network has high accuracy, then we increase the threshold, in order to maintain only the most accurate predictions. After a fixed number of images predicted, we compute the ratio of prediction for every room; if the ratio of predicted room over the total prediction is above a threshold (chosen based on the accuracy on the CNN, then the Thymio predicts that room. In the first experiment we discarded predictions that are predicted with less than 0.8 accuracy, while in the second experiment, since the CNN performs better, we discarded images that are predicted with less than 0.9 accuracy.

E. Human Control

In the second experiment, we control the Thymio using the keyword. Every keys (the arrow keys) is associated with a callback that change the linear and angular velocity. For example, pressing the top arrow key, increases the linear velocity, while pressing the bottom arrow key decreases it. In this way we were able to experiment a continuous prediction going back and forth between the room and the entrance. As opposed to the first experiment where the Thymio stops after having predicted a room, in this case, when a room is predicted, a led is turned on (the color is based on the room predicted) and the Thymio continues moving and predicting. In this way, every time we change a room, the Thymio changes its led color.

IV. Problems and Troubleshooting

After a first attempt to simulate our environments using Gazebo, we decided to use the real Thymio. Using the real Thymio introduced several issues that didn't happened using Gazebo. In the following sections we are going to explain the major issues we encountered and how we solved them.

A. Proximity Sensors

Every subscribers has a queue for storing the messages that it receives from a sensor. Setting the wrong size of the queue could led to delayed message or, in the worst case, to lose messages. If we set a queue size too large, then the queue will grow up quickly if the subscriber is not able to dispatch all the message received. This could lead to either delayed message or lost messages. Also, there is not a general rule for setting this value; it depends on the application [4]. In our first experiment, we first tested the random walking, without subscribing for the the camera sensor. The Thymio were able to receive all the message we sent without any delay. Instead, while we introduced the camera to save images, the Thymio started to not response to our messages, hitting multiple times obstacles or walls. Setting a small value for the queue size of the camera subscriber (we set to 1), solved this issue.

B. Prediction

Once we trained the model, we use our CNN to predict the room for each image given as input. Using the OpenCV library to handle images, both for saving and loading the images, we realized that before giving in input the image to the CNN we had to convert it from BGR to RGB. It was not so obvious at the beginning because, during training we loaded the images and then trained the model. When we had to predict images, obtained from camera pixels, we started receiving wrong predictions. The reason was that the OpenCV library uses BGR format to save images; instead matplotlib (that we used to show them) uses RGB. [5].

V. Results

In both the experiments, the Thymio was able to recognize the room in which it was placed. We present the result obtained, distinguishing the first and the second experiment.

A. Experiment 1

The CNN used in this experiment reached an accuracy of 84.7133 (Fig 5). The reason for the peaks in the loss plot (Fig 6) is that both rooms share some objects, for example walls, the floor of the same color, buckets, etc... So this means that images containing common objects are classified either to belong to room 1 or room 2, depending on how many pictures are collected for that room. If the Thymio collects a lot of images of the wall in room 1, during the data collection step, then it will predict room 1 every time it will see a wall. But if the same type of images are collected also for the room 2, than its predictions oscillates. As we can see from Fig. 7, the Thymio was able to recognize the room with high probability when the images contains object of that room, in other cases it predicts one of the room with less probability. This is the reason why in the prediction step (see Algorithm 1), we discard the labels predicted with less that 0.8 of accuracy, because that predictions are of images that contains objects common to both rooms.

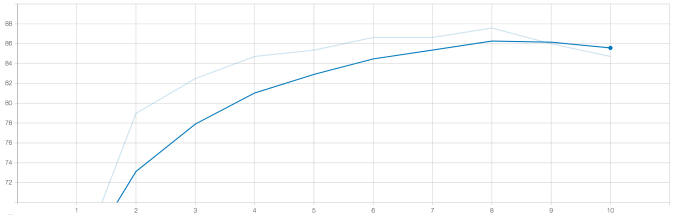

Fig. 5. Experiment 1: Accuracy - x axis epochs, y axis accuracy value

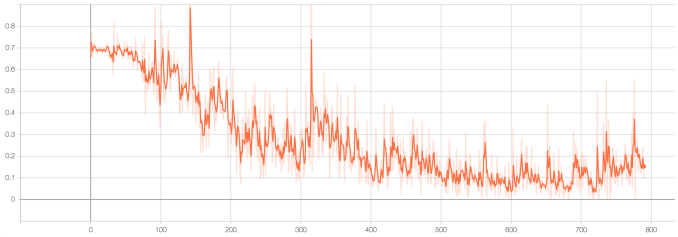

Fig. 6. Experiment 1: Loss - x axis batch, y axis loss value

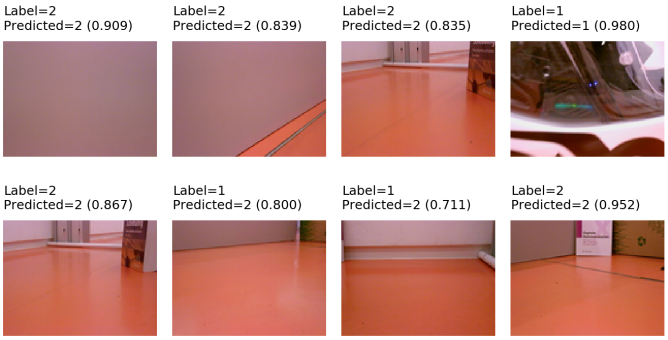

Fig. 7. Experiment 1: Example of Predictions

Fig. 10. Experiment 2: Example of Predictions

B. Experiment 2

In the second experiment, the CNN used for prediction reached an accuracy of 0.9975 (Fig 8). Its accuracy is higher than the CNN used in the first experiment because the images used don't have a lot of objects in common; they just have the same color of the floor. In this case, the images of both rooms

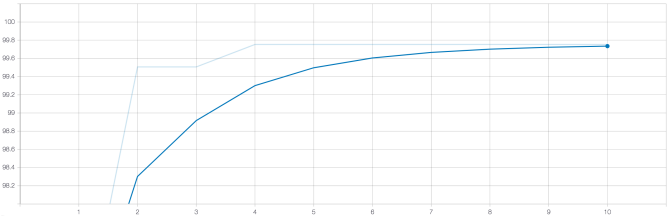

Fig. 8. Experiment 2: Accuracy - x axis epochs, y axis accuracy value

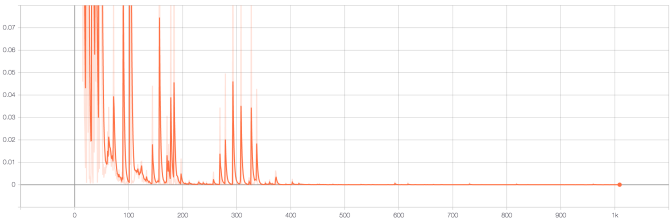

Fig. 9. Experiment 2: Loss - x axis batch, y axis loss value

are very different between each other: in one room we have a lots of repeating object like desks and chairs, while in the other we have few objects, like the vending machine. For this reason the Thymio is able to predict with few images where it is with high accuracy. Using also the keyboard to control the Thymio, and move it around in one of the room, it was able to correctly predict the room where it is even in places were it didn't take picture for training (Fig 10). Since the camera is pointing forward, rather that down, it was able to predict one room as soon as it enter in it.

VI. Conclusion

Working with the real Thymio introduces several issues that didn't happened using the Gazebo environment. Moving the robot required some extra work due to the delay introduced in receiving and sending messages. Despite the issues encountered, we were able to build a state machine for this kind of task and a CNN that predicts correctly where the robot is located, in both the experiments we did. In this work we investigate how can we enable a Thymio to predict a place using an artificial neural network (in our case a CNN). We also analyzed the issues that might rise when working with real robots rather that using a simulated environment.

VII. Future Works

Possible source of inspiration to extend this work, could be:

- train the Thymio in multiple rooms
- recognize if objects are moved in a room w.r.t to their original position

References

[1] Guzzi, Jérôme and Giusti, Alessandro and Di Caro, Gianni A. and Gambardella, Luca M., Mighty Thymio for Higher-Level Robotics Education, 2018.
[2] Pytorch, [online], An open source deep learning platform that provides a seamless path from research prototyping to production deployment, v1.1.
[3] Training a Classifier, [online], Tutorials > Deep Learning with PyTorch: A 60 Minute Blitz > Training a Classifier
[4] ROS Documentation, Publishers and Subscribers: Choosing a good queue_size
[5] OpenCV - Open Source Computer Vision, Getting Started with Images