Technical University of Denmark

Specialization: Artificial Intelligence and Computer Science

YOLACT application with Intel RealSense camera

RESEARCH PROJECT

Author:	Anastasia Ostapenko
Great supervisor:	Professor Thomas Bolander
Year:	2021

Declaration

I declare that I worked on my thesis on my own and that I used only the sources listed in the References and Internet sources

In ....
Anastasia Ostapenko

Title:
YOLACT application with Intel RealSense camera

Author:	Anastasia Ostapenko
Specialization:	Artificial Intelligence and Computer Science
Type of work:	Research project
Supervisor:	Professor Thomas Bolander Department of Applied Mathematics and Computer Science Technical University of Denmark
Consultants:	Lasse Dissing Hansen

Abstract: The work deals with training YOLACT on a custom dataset, processing visual information from RealSense cameras and using the obtained results for tracking whether the object was hidden into a box.
Key words: visual scenes, yolact, RealSense cameras, custom dataset

Contents

Introduction

This work is based on the results achieved in the previous project - "Description of visual scenes using natural language - implementation in the Pepper robot", however the objectives here are different, moreover instead of using cameras from the robot we are going to use the Intel RealSense cameras which provide better image quality and higher robustness.

The main objectives of the project are:

1. Make YOLACT work on RealSense cameras
2. Find a way to run YOLACT on a computer without NVIDIA GPU
3. Test robustness on moving objects
4. Train YOLACT on our own dataset
5. Handle objects that become hidden

Chapter 1

YOLACT

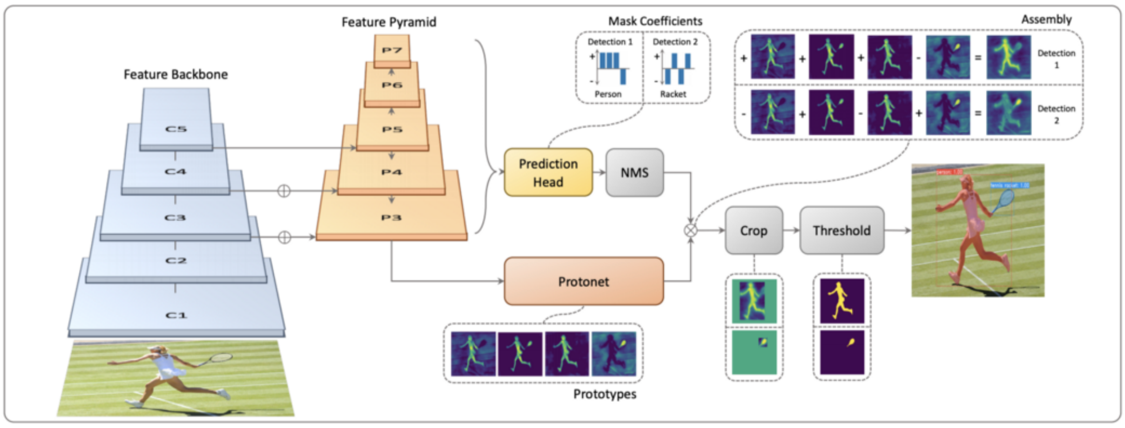
There is an increasing interest in the research area of Computer Vision nowadays and several great advances in object detection and instance segmentation were made during the last several years. It is good to note, that object detection is a process of detecting instances of some object (car, laptop, human) in images and video and labeling them with their associated class. And instance segmentation is separating all objects from the background image and assigning each of them a separate categorical pixel-level mask. Actually, instance segmentation can be viewed as a special kind of object detection. But instead of localizing an object by a bounding box, we prefer to use pixel-level localization.

Different approaches for object detection and instance segmentation exist. For our project - description of visual scenes and visual question answering in real time, we need to choose one which will be reliable enough, fast, simple to train and providing as much information about the objects as it is possible.

The first object detection algorithms (before deep learning) consisted of three steps: proposal generation, feature vector extraction and region classification. In the first stage areas where objects could have possibly been were proposed by scanning the whole image with sliding windows. Then on each area a feature vector was obtained from the siding window. The feature vector was encoded by low-level visual descriptors such as SIFT (Scale Invariant Feature Transform), Haar, HOG (Histogram of Gradients) or SURF (Speeded Up Robust Features) [1].

With the development of deep convolutional neural networks the was a significant progress in object recognition. Current detector frameworks based on deep learning can be divided into two categories: one-stage detectors and two-stage detectors. Two stage detectors such as Faster R-CNN (Region-based Convolutional Neural Networks) or Mask R-CNN firstly find on the image the "regions of interest" that have high probability of containing an object. Then the proposed regions are fed into the R-CNN and their classification score and the spatial offsets are obtained. One-stage object detectors make categorical prediction of objects on each location of the feature maps directly, without the cascaded region classification step [1]. Comparing to two-stage detectors one-stage are more efficient, however two-stage detectors are more precise. It can be said that one-stage detectors speed up two-stage detectors by simply removing the second stage, however such detectors as YOLO or SSD are able to fill the gap in performance in some other ways (e.g., strong data augmentation, anchor clustering, etc. [2])

An upgraded version of YOLO detector is YOLACT (You Only Look At Coefficients) which we are going to use in our project. YOLACT (also YOLACT++)[2][3] is a state of the art, real-time object segmentation algorithm, which can do object detection and segmentation with high accuracy and is much faster than two-stage detectors. YOLACT omits the localisation step and breaks the process of instance segmentation in two parallel tasks: generating a set of prototype masks and predicting per-instance mask coefficients [2]. Then for each instance the prototypes are linearly combined with the corresponding predicted coefficients and then cropped using a predicted bounding box. The prediction of prototype masks is critical to ensure high resolution of the final instance masks. The prototype masks depend only on input images and are independent of categories and specific instances.



YOLACT Architecture

For our work we have chosen to use exactly YOLACT but not any other two-stage or one-stage object detectors because of two main reasons: (i) in our project we should be able to process video from RealSense camera in real time and so - very fast and YOLACT is ideal for this (ii) we also need a detector which will be able to provide high accuracy and not all one-stage detectors (which are fast) are able to do this.

Chapter 2

Connecting RealSense Camera and YOLACT

The first objective of our project is to make YOLACT work on Intel RealSense camera and then investigate robustness of the achieved results.

2.1 Connection

The main part of this work was done in the previous project when we needed to connect the robot's camera and YOLACT. For this project we have replaced it with Intel RealSense depth camera. We have also changed the whole structure of the code in order to make it more objective oriented and also to simplify the process of choosing different cameras.

2.2 YOLACT on a computer without NVIDIA GPU

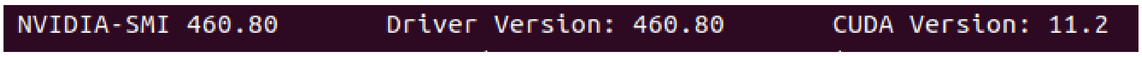
Our next goal is to run YOLACT on a computer without NVIDIA GPU in order to test algorithm on the local computer. We have chosen to use remote server for this purpose as it was relatively simple to implement. However, in such case the program will be much slower because of the need to transfer the computations and images between local and remote devices.

The video is processed by splitting into single frames and YOLACT is applied on each frame individually. In our solution with the use of remote server we save each such frame, taken by RealSense camera on the local machine. Then it is processed on the remote server, saved there with all the extracted information and at the end send back to the local machine. We have also added additional checking whether the frame is fully processed and saved safely in order to prevent bugs caused by two different processes accessing one file at the same time. However, this results in the whole process being too slow. So as a conclusion for this part we would say that such way is suitable for troubleshooting and testing but is not sufficient enough for real "purposes" and working on a local machine with NVIDIA GPU is preferable in such case.

2.3 Robustness on moving objects

We have tested robustness with Intel RealSense camera, on moving objects.

Figure 2.1: Information about NVIDIA



```
NVIDIA-SMI 460.80        Driver Version: 460.80        CUDA Version: 11.2
```

The results of course depend on the object itself and initial probability with which it was recognised (before moving), in other words, how well the object corresponds to the data on which YOLACT was trained. We would say the robustness on moving objects is high. We would also mention that good lightning and camera with high resolution are important for increasing the robustness. You can find videos of two good examples below.

1 example, 2 example.

However, as it was said earlier not all objects from the COCO dataset can be recognised well while moving. Here you can find an example of YOLACT trying to recognise a moving sports ball 3 example. This can also motivate one to create a custom dataset if a particular object (such as sports ball which is recognised badly) should be used for the project.

Chapter 3

Handling objects that become hidden

One of the main objectives of the project is to handle objects that become hidden either by being put into containers or behind them.

3.1 Training YOLACT with a custom dataset

We have decided to train YOLACT on our own dataset for this purpose in order to increase robustness and to decrease False Positive rate on recognised objects. This part is based on the following tutorial [4].

We have chosen two types of objects for our purpose: box as a container and tennis ball as an object which will be hidden into container.

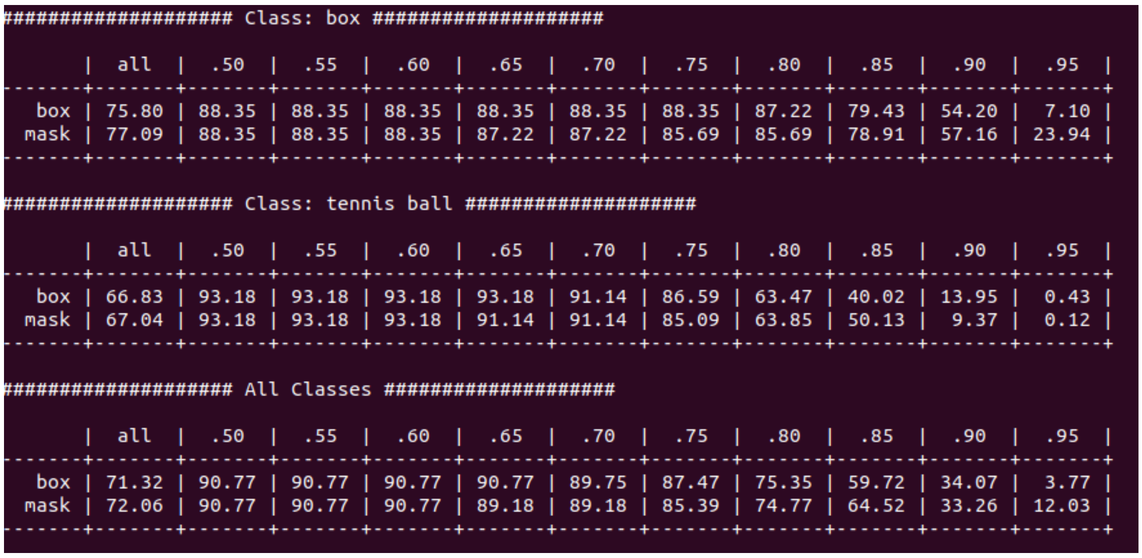
First step was to create a COCO style dataset. We have found two datasets of boxes and tennis balls on Internet and also added our own photos there. This was particularly important in the box dataset. The reason is as follows: box from one side is a simple object, so not a lot of features can be extracted and from the other side there are a lot of different types of boxes so we wanted to be sure that our particular boxes which we use in the project will be recognised.

Then we have annotated all the photos with the help of coco-annotator tool. More information about the whole process can be found here [5] and here [6].

We didn't train completely from scratch and have used the resnet101_reducedfc.pth pretraind model.

Time required for training - 4 days.

Figure 3.1: Achieved accuracy



```
#################### Class: box ####################

       |  all  |  .50  |  .55  |  .60  |  .65  |  .70  |  .75  |  .80  |  .85  |  .90  |  .95  |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
   box | 75.80 | 88.35 | 88.35 | 88.35 | 88.35 | 88.35 | 88.35 | 87.22 | 79.43 | 54.20 |  7.10 |
  mask | 77.09 | 88.35 | 88.35 | 88.35 | 87.22 | 87.22 | 85.69 | 85.69 | 78.91 | 57.16 | 23.94 |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+

#################### Class: tennis ball ####################

       |  all  |  .50  |  .55  |  .60  |  .65  |  .70  |  .75  |  .80  |  .85  |  .90  |  .95  |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
   box | 66.83 | 93.18 | 93.18 | 93.18 | 93.18 | 91.14 | 86.59 | 63.47 | 40.02 | 13.95 |  0.43 |
  mask | 67.04 | 93.18 | 93.18 | 93.18 | 91.14 | 91.14 | 85.09 | 63.85 | 50.13 |  9.37 |  0.12 |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+

#################### All Classes ####################

       |  all  |  .50  |  .55  |  .60  |  .65  |  .70  |  .75  |  .80  |  .85  |  .90  |  .95  |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
   box | 71.32 | 90.77 | 90.77 | 90.77 | 90.77 | 89.75 | 87.47 | 75.35 | 59.72 | 34.07 |  3.77 |
  mask | 72.06 | 90.77 | 90.77 | 90.77 | 89.18 | 89.18 | 85.39 | 74.77 | 64.52 | 33.26 | 12.03 |
-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
```

You can find a video example here. The results are sufficient for our needs but for more accurate predictions a larger dataset is required.

3.2 Detecting objects which were hidden into container

The first step is to detect which objects were removed from the frame and if the object we need - tennis ball is among the removed objects. Then we should understand on how many frames the object misses. If the number is larger than 5, we check other conditions.

The second condition is that container must be present in the frame. In our case we use class "box" from the custom dataset as a container.

And at the end we compare the coordinates of container and removed object (coordinates are saved from the last frame), if the difference between them by x and y is not "big" (the object was right above the container) we make a conclusion that the object was probably hidden into a box.

You can find a demo video here. Script output is on the right screen.

One big improvement we can see here is to find out how the depth information from RealSense camera corresponds with the YOLACT centroids of the objects and also use the third "depth axis".

Chapter 4

What has been tried but not achieved

1) Reducing the number of classes from pre-trained coco-dataset. It is advised to "remove the COCO annotations in the annotation JSON that aren't part of the classes you want" or "reduce the number of classes total to 6-7 and retrain the last layer". However, both of this ways caused errors with which we at the end were not able to handle. The solution here can be to train neural network on the selected classes from the scratch using already prepared COCO dataset and annotations.

2)Handling depth information: combine 2D segmentation with depth camera to extract approximate object depth. It looks like there is no correspondence between the coordinates of the object extracted with the help of YOLACT and depth of the image received with RealSense.

4.1 Known bugs and how to deal with them

1) If you get "Too many open files: 'classes.json'" something has probably gone wrong with json encoding. You should open classes.json and replace everything with {"init": "true"}

2) Frame didn't arrive within 5000 - this means camera was disconnected. Try to reconnect it.

3)If the program accidentally crushes and mistake is related to coordinates - run it again.

Chapter 5

How To

In this chapter we are going to discuss some practical examples of using the application.

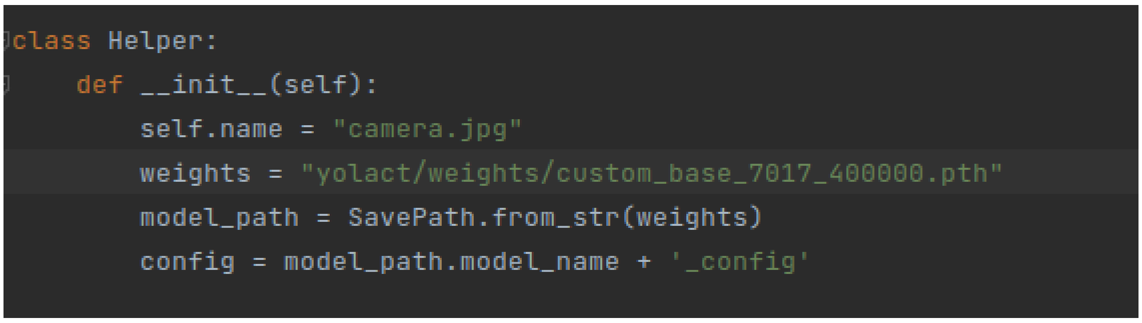
5.0.1 Start

Before you start running anything you should install conda and then init conda environment yolatc-env. Then activate conda environment with conda activate yolact-env, navigate to the folder with the project and run python3 main.py .

5.0.2 Choose weights

In order to run the application with COCO pre-trained dataset change weights in help_module.py to yolact_base_54_800000.pth . If you want to use the weights for a custom dataset use custom_base_7017_400000.pth .

Figure 5.1: Change weights

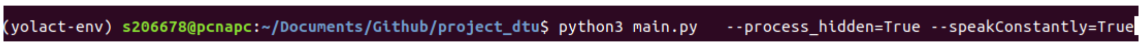


```
class Helper:
    def __init__(self):
        self.name = "camera.jpg"
        weights = "yolact/weights/custom_base_7017_400000.pth"
        model_path = SavePath.from_str(weights)
        config = model_path.model_name + '_config'
```

5.0.3 Hidden objects

In order to test hidden object process you should use custom weights (custom_base_7017_400000.pth) and run python3 main.py –process_hidden=True –speakConstantly=True

Figure 5.2: Process hidden objects



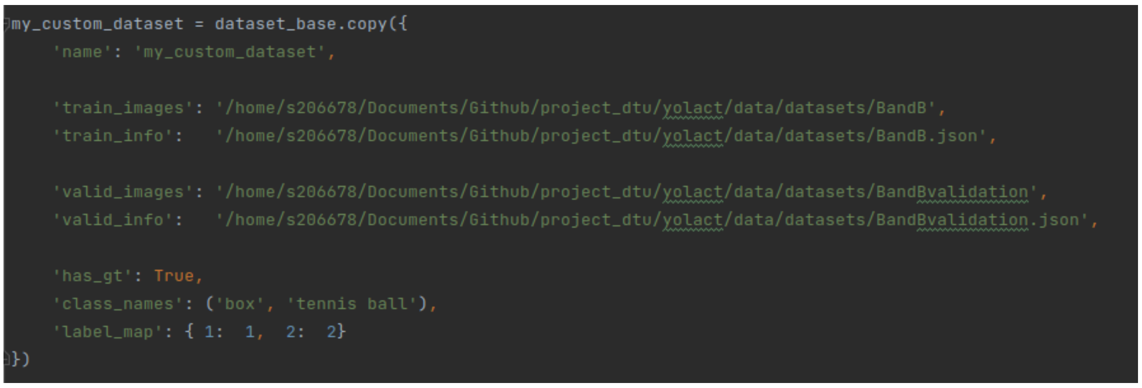
```
(yolact-env) s206678@pcnapc:~/Documents/Github/project_dtu$ python3 main.py   --process_hidden=True --speakConstantly=True
```

If you want to work with the COCO dataset and to choose your own class for container you should navigate to help_module.py and change self.box_class to the name of the object you want to be your container (for example vase or microwave).

5.0.4 Training on a custom dataset

In config.py fill my_custom_dataset with your own data.

Figure 5.3: Config.py



```
my_custom_dataset = dataset_base.copy({
    'name': 'my_custom_dataset',

    'train_images': '/home/s206678/Documents/Github/project_dtu/yolact/data/datasets/BandB',
    'train_info':   '/home/s206678/Documents/Github/project_dtu/yolact/data/datasets/BandB.json',

    'valid_images': '/home/s206678/Documents/Github/project_dtu/yolact/data/datasets/BandBvalidation',
    'valid_info':   '/home/s206678/Documents/Github/project_dtu/yolact/data/datasets/BandBvalidation.json',

    'has_gt': True,
    'class_names': ('box', 'tennis ball'),
    'label_map': { 1:  1,  2:  2}
})
```

Make sure you have resnet101_reducedfc.pth inside your inner weights folder. Then run python train.py –start_iter=0 –batch_size=2 You can interrupt the process with Ctrl+C. The resulting file will be inside weights folder. Once you are done move the resulting file with weights to yolact/weights.

5.0.5 YOLACT through a server

In order to run the YOLACT part of the project on the remote server follow the instructions below:

On local machine:

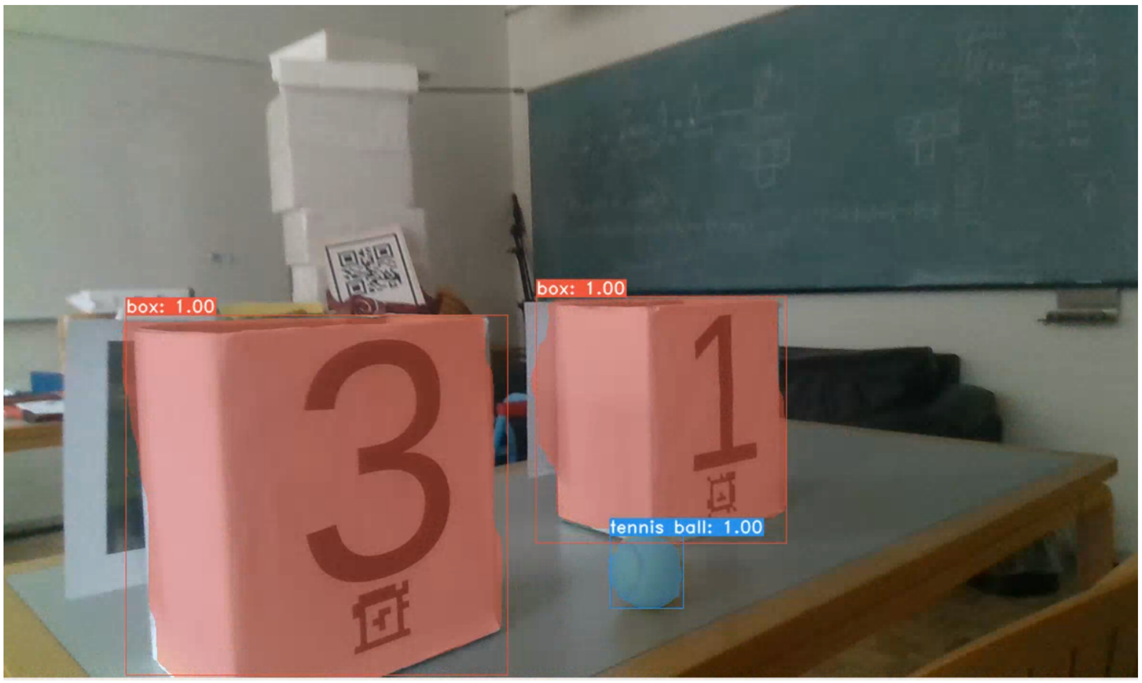
- Connect RealSense camera to your computer
- Run main.py with arguments –isRemote=True, –hostname, –username, –password, where hostname, username and password are the credentials for your account on the remote server

On the remote server:

- Activate conda yolact-env
- You should have YOLACT project and yol_sense.py file
- Run yol_sense.py (at the same time with main.py)

Conclusion

We can declare that we have achieved our goals. We implemented a system which is able to process the visual information from the RealSense camera and detect whether the object was hidden into a box. We have also created our own COCO-style dataset and trained YOLACT on it. Besides this the application now supports work on the remote server and can be used even on a computer without NVIDIA GPU.



Bibliography

1. WU, Xiongwei; SAHOO, Doyen; STEVENC.H.HOI. Recent advances in deep learning for object detection. *Neurocomputing*. 2019. ISSN 0925-2312. Available also from: https://www.sciencedirect.com/science/article/pii/S0925231220301430.
2. BOLYA, Daniel; ZHOU, Chong; XIAO, Fanyi; LEE, Yong Jae. YOLACT. Real-time Instance Segmentation. 2019. Available also from: https://arxiv.org/pdf/1904.02689.pdf.
3. BOLYA, Daniel; ZHOU, Chong; XIAO, Fanyi; LEE, Yong Jae. YOLACT++. Better Real-time Instance Segmentation. 2019. Available also from: https://arxiv.org/pdf/1912.06218.pdf.
4. KELLY, Adam. Train YOLACT with a Custom COCO Dataset. 2020. Available also from: https://www.immersivelimit.com/tutorials/train-yolact-with-a-custom-coco-dataset.
5. KELLY, Adam. Create COCO Annotations From Scratch. 2019. Available also from: https://www.immersivelimit.com/tutorials/create-coco-annotations-from-scratch.
6. BROOKS, Justin. *COCO Annotator* [https://github.com/jsbroks/coco-annotator/]. 2019.