Cheers!
Computer Vision WS20/21: Detect the Bottle Cap
Proposal / Plan of Attack

Martin Pluisch
Matr. Nr. 9042396
martin.pluisch@smail.inf.h-brs.de

1 INTRODUCTION

Detecting and interpreting objects in images and video sequences is one of the core aspects of computer vision.
As such, the given task "Detect the Bottle Cap", proposed for the Computer Vision lecture of WS20/21 at H-BRS, makes for an interesting and exemplary computer vision project.
For "Detect the Bottle Cap", each student has to create a program which should be able to detect and count all bottle caps within a given image sequence or video. There are four differing trackable classifications:

- Bottle cap lies upside down
- Bottle cap lies face up
- Bottle cap is deformed
- Distractors, which is anything within the region of interest that is not a bottle cap

The mentioned region of interest (or ROI) is, in this case, a "flat rectangular container with homogeneous background color".
Every student also has to shoot ten video sequences of him/her throwing three to fifteen bottle caps plus distractors into said ROI, and label the video footage on a static frame according to the different classifications. Detecting bottle caps, or objects in general, within an image or video sequence is an arbitrary and seemingly miniscule task for a human being, but not so much for a computer. A computer basically has to analyze the actual pixels of an image to make an "educated guess" about the objects within said image. Deconstructing the steps of a human brain and visual system detecting and identifying objects, to then applying this knowledge and train a computer to "see" is a very exciting and prevalent topic.

2 MATERIAL AND METHODS

2.1 Input and Output

Since every student submits ten annotated frames, we will receive a training data package containing around 600 annotated images.
Said training package consists of the roughly 600 images (1920x1080, PNG) and the connected labels as .json files, which individually store the pixel-positions of each dot of a marked labeling-area.
The program should later on receive arbitrary images with (or maybe even without) bottle caps in it. I'd assume a constant image fidelity and input resolution (1920x1080), but I will apply a simple gamma correction with OpenCV to level the input images to some degree.
Using the training package, I'll try to teach my program to identify bottle caps in arbitrary images and video sequences; the output should at least be a simple console output printing the amount and type of detected bottle caps, and an annotated version of the input image/video as seen in figure 2 at best.

2.2 Plan of attack

Besides OpenCV, I'm intending to use a neural network to apply the training package and to train my program.
While looking for viable neural network solutions for Python, I've stumbled across "You Only Look Once" (YOLO) [1] and its latest installment YOLOv5, developed by Ultralytics [2]. As the name suggests, the system "only looks once at an image to predict what objects are present and where they are" , making it really fast at processing images in real-time (full version: 45 FPS, smaller lightweight version: 155 FPS) [1].
If YOLO indeed processes images at around 45 FPS, live-feed bottle cap detection would be possible, which would be the ultimate goal for my implementation - and is thus the main reason why I'm opting for YOLO.
Since this will be my first hands-on experience with object detection, my (possibly naive) plan of attack consists of the following steps:

(1) Pass a video file into my algorithm
(2) Detect and extract a purposeful still frame from the passed video
(3) Find and extract the region of interest
(4) Train YOLO using the aforementioned training data, possibly creating and adding more training data of my own
(5) Pre-process the extracted still frames using gamma correction with OpenCV
(6) Use the generated weights to detect bottle caps within the pre-processed image
(7) Count the detected bottle caps and create an annotated image
(8) Try to adapt and extend the program to detect and classify bottle caps within a live video-feed (e.g. webcam)

The big advantage of YOLO, its speed, also means that it lacks accuracy in direct comparison to alternatives [3]. If the accuracy should become a problem within my implementation, I might switch from YOLO to "Single Shot MultiBox Detector" (SSD), which also uses a single network and presumably delivers better accuracy at less FPS [4].
Besides gamma correction, it might be a good idea to optimize the passed-in images further than that, maybe by compressing the files or shrinking their resolutions for faster read/write speeds; I will test multiple image optimizations and see if there's actually a performance increase.
Capturing live video for my last step, e.g. from a webcam, is easy thanks to OpenCV [5], the (performance) hurdle is grabbing each frame and running it through the trained YOLO neural network.

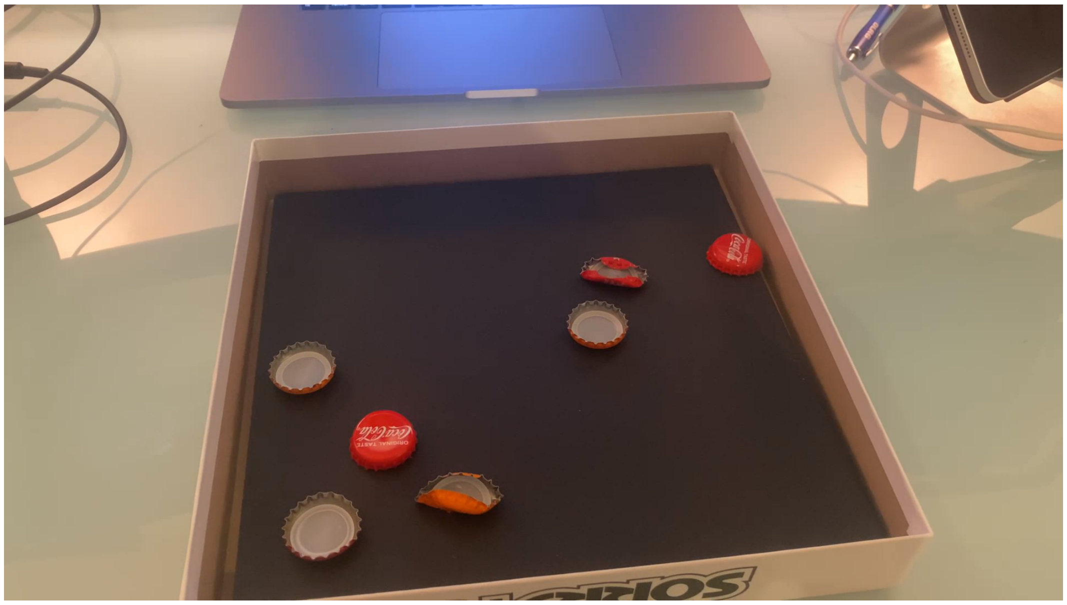

Figure 1: Exemplary still frame from one of my videos

Using interpolation and only grabbing every *n*-th frame might be a good start for first optimizations.

3 ANTICIPATED RESULTS

Going step-by-step, I'll first get a good overview on how to actually implement, train and use YOLO within my python script.
The initial steps of my implementation will revolve around actually reading a .mp4-video file and detecting a still frame for extraction. The still frame I'm looking for should optimally look like the image seen in figure 1.
My approach to solve this first task is to keep track of a frame x and the next frame $x + 1$. Using OpenCV, I'll then calculate the absolute difference between x and $x + 1$ and compare the resulting value to a set threshold, that I'll also have to figure out at this stage.
Besides detecting and extracting the mentioned still frame, I'll also have to make sure that the extracted frame is relevant for me and my algorithm - to do so, I'll search for still frames only in the middle third of the passed in video file. By doing so, I make sure that still frames at the start or end of the video file get ignored. Otherwise, my program would probably detect still frames at the very beginning of the video with no bottle caps present at all.
To implement this approach, I'll first grab the total amount of frames of the passed video, using OpenCV's *CAP_PROP_FRAME_COUNT* property, and then calculate the first and last frame (of my still-frame detection-range) by multiplying the total amount of frames by $\frac{1}{3}$ and $\frac{2}{3}$ respectively.
To find the ROI in the next step, I'll try to find the largest rectangular/box shape present in the frame by finding the contour with the biggest area. Extracting said area can be either done by copying the pixels within, or by simply cropping the input frame to the desired area.
Next, I'll try to use the annotated data package to generate prediction weights for the single neural network of YOLO.
To do so, I'll pass the input images plus the corresponding .json labels to YOLO, so that YOLO learns about the different object classifications I'm trying to track.
Once the prediction weights are constructed, I'm going to try to implement the actual object detection in my python script. I'll read an arbitrary video as input, extract the still frame as mentioned above, use gamma correction using OpenCV, let it run through the single neural network and its weights and see if the predictions are accurate. If the predictions are not accurate enough, I might either create and train more sampling data or switch to the aforementioned SSD approach [4]. Best case scenario, my program will accurately detect bottle caps in varying states within images after this step.
Once the individual bottle cap types are accurately detected, counting them is a self-explanatory task. In this step, I'll also implement my own labeling functions, since my output should be an annotated version of the extracted input image as seen in figure 2.
Last but not least, I'd then try to adapt / extend my python implementation to actually pass in a live-video feed to detect bottle caps "on-the-fly". Although, this step is only possible, if both speed and accuracy of YOLO and the neural network weights are good enough. For this last step, I'd also have to take a look at handling the incoming video-feed and use each frame opposed to extracting one still frame.

3.1 Test design

Due to the step-by-step approach, there are also multiple steps of testing included.
First off, I'll test my still frame extraction algorithm by passing sample bottle cap videos to it. As mentioned before, the result should look a lot like figure 1. As this step produces the basis for the entire

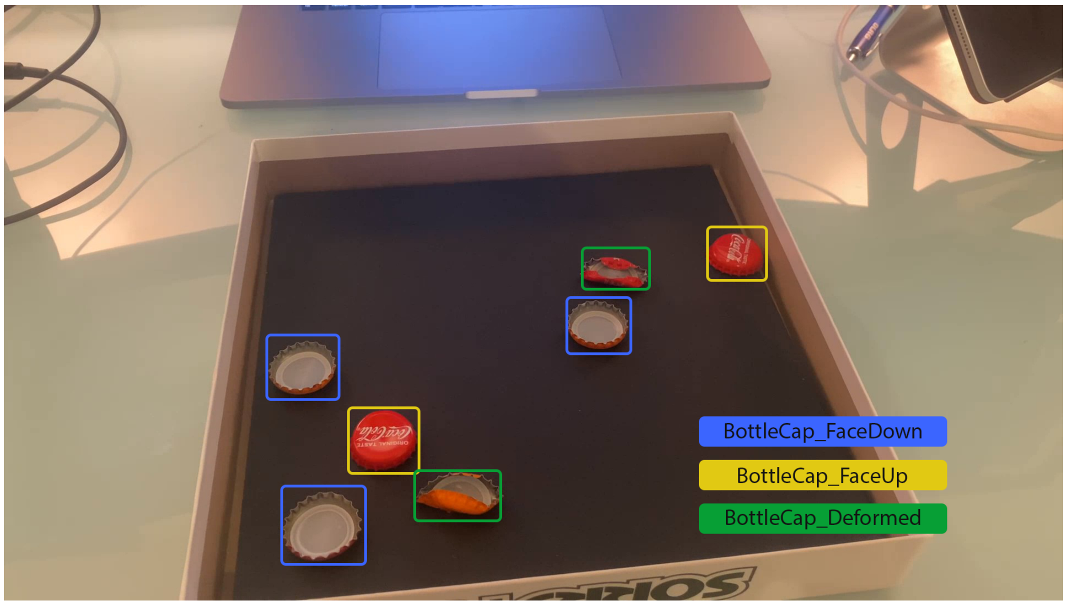

Figure 2: Anticipated output image of my implementation

detection algorithm later on, it has to be very consistent and must not produce false still frames.
The extraction of the ROI is intertwined with this step and will also take place at this stage in testing. The algorithm should receive my still frame from before and use it to extract the ROI as described earlier on; I'll test this step manually simply by looking at the generated output images.
Then, I'll test my gamma correction functions and compare different output images to one another. The gamma correction should make for a level playing field across input images.
Next, once the network is trained and the weights are generated, I'll create very basic test scenes, with only one bottle cap at a time. In total, I will test at least four different images here, one for each classification (bottle cap face up, bottle cap face down, bottle cap deformed, no bottle cap present / only distractors).
If those tests are successful, I'll test images with multiple bottle caps of different classifications present at once. This testing stage is also a test for my own labeling functions, i.e. drawing rectangular markers with matching annotations.
Right afterwards, I will also test the performance impacts of optimizing the incoming image files themselves, i.e. shrinking the resolution / compression. I can't predict yet if there'd be any performance improvement, but I'll thoroughly test and record the effects.
The last testing stage would then revolve around the video input-stream and testing on-the-fly detection. Again, I would start with a simple ROI containing singular bottle caps in varying states, then test multiple bottle caps and distractors visible at once. I'd also adapt and then test my labeling function for a live-feed at this point in time.

3.2 Risks

The main risk is that my initial plan of attack, based on YOLO, is not accurate enough for the sake of detection speed. Thus, I might have to change the plan of attack and switch from YOLO to the already mentioned SSD [4].
Also, due to my lack of experience with OpenCV, YOLO and neural networks, my plan of attack might be a bit too "ambitious" and I might run into a dead-end at some point during my implementation. A simpler approach to detect bottle caps (using edge detection in combination with other standardized CV measures) might be the easier and less risky way of implementing a possible solution; I'm opting for the seemingly harder, but also faster and more accurate approach, since it gives me some good hands-on experience.

REFERENCES

[1] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You Only Look Once: Unified, Real-Time Object Detection. https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf - last visited on 17.11.2020.
[2] Ultralytics. YOLOv5. https://github.com/ultralytics/yolov5 - last visited on 17.11.2020.
[3] G Chandan, Ayush Jain, Harsh Jain, et al. Real time object detection and tracking using deep learning and opencv. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 1305–1308. IEEE, 2018.
[4] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *ECCV*, 2016.
[5] Open Source Computer Vision (OpenCV) Documentation. Getting Started with Videos - Capture Video from Camera. https://docs.opencv.org/master/dd/d43/tutorial_py_video_display.html - last visited on 17.11.2020.