Detection of Canned Beverages

1st Radu Ștefan-Vlăduț
Cohort 1306A
stefan-vladut.radu@student.tuiasi.ro

2nd Petrișor Eduard-Gabriel
Cohort 1311A
eduard-gabriel.petrisor@student.tuiasi.ro

I. Introduction

With the given task of building a program with the function of detecting certain objects by using an individually procured data set we initially pondered what thing we might tailor our project towards, considering the vastness of the things we could've chosen. After much deliberating, we set our sights on the humble and common soda can. With different shapes, sizes and colors, it offers plenty of difference from one brand to another whilst still presenting a unifying form, all that on top of being a readily available object which can be found at any corner store, which greatly aids the task of making our very own data set. But what truly swayed us towards this idea is the difficulty of actually distinguishing the different kinds of cans by use of only your tactile senses. Whilst some cans might present different textures and reliefs, some are essentially virtually indistinguishable from one another, fact which we hope offers our project a kind of real world use.

II. State-of-the-art

Object detection performance, as measured on the canonical PASCAL VOC dataset, has plateaued in the last few years. The best-performing methods are complex en- semble systems that typically combine multiple low-level image features with high-level context. The state-of-the-art methods can be categorized into two main types: one-stage methods and two stage-methods:

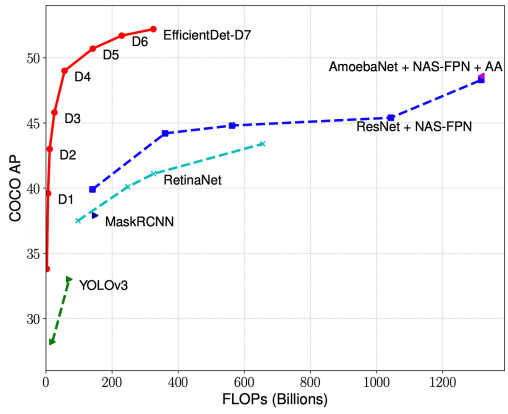

Fig. 1. Comparision in training times for a widespread dataset

Identify applicable funding agency here. If none, delete this.

A. One-stage methods

One-stage methods prioritize inference speed, and example models include YOLO, SSD and RetinaNet.

- ***YOLO***: YOLO is a state-of-the-art, real-time object detection system that can detect over 9000 object categories. YOLO suffers from a variety of shortcomings relative to state-of-the-art detection systems. Error analysis of YOLO compared to Fast R-CNN shows that YOLO makes a significant number of localization errors. Furthermore, YOLO has relatively low recall compared to region proposal-based methods. Thus it focuses mainly on improving recall and localization while maintaining classification accuracy. It's main defining quality however is being extremely easy and out of the box easy to use.
- ***SSD***: 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. Is the first deep network based object detector that does not resample pixels or features for bounding box hypotheses and and is as accurate as approaches that do. This results in a significant improvement in speed for high-accuracy detection. The fundamental improvement in speed comes from eliminating bounding box proposals and the subsequent pixel or feature resampling stage. Improvements include using a small convolutional filter to predict object categories and offsets in bounding box locations, using separate predictors (filters) for different aspect ratio detections, and applying these filters to multiple feature maps from the later stages of a network in order to perform detection at multiple scales.
- ***RetinaNet***: Similar to SSD, RetinaNet uses Feature Pyramid Networks (FPN) to extract multiple-scale information and performs classification and box regression in one stage. In RetinaNet, a novel loss function is adopted to address class foreground and background class extremely imbalance. With such a novel loss function, RetinaNet can achieve high accuracy in object detection with fast inference time. RetinaNet can use different neural networks as the backbone model to learn feature maps, which are the input to the FPN. The third component of RetinaNet is the classification subnet and box regression subnet attached to FPN to compute the results. For example,

the max-pooling output of last three blocks in VGG can be chosen as FPN input in RetinaNet, while the output of last layer in Inception-ResNet block can be selected as the FPN input.

B. Two-stage methods

Two-stage methods prioritize detection accuracy, and example models include Faster R-CNN, Mask R-CNN and Cascade R-CNN.

- ***Faster R-CNN***: Faster R-CNN, is composed of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector [2] that uses the proposed regions. The entire system is a single, unified network for object detection. Using the recently popular terminology of neural networks with 'attention' mechanisms, the RPN module tells the Fast R-CNN module where to look. A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score.3 We model this process with a fully convolutional network , which we describe in this section. Because our ultimate goal is to share computation with a Fast R-CNN object detection network , we assume that both nets share a common set of convolutional layers. In our experiments, we investigate the Zeiler and Fergus model (ZF), which has 5 shareable convolutional layers and the Simonyan and Zisserman model (VGG-16), which has 13 shareable convolutional layers.
- ***Mask R-CNN***: Mask R-CNN, extends Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression (Figure 1). The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. Mask R-CNN is simple to implement and train given the Faster R-CNN framework, which facilitates a wide range of flexible architecture designs. Additionally, the mask branch only adds a small computational overhead, enabling a fast system and rapid experimentation.
- ***Cascade R-CNN***: It is a multi-stage extension of the R-CNN, where detector stages deeper into the cascade are sequentially more selective against close false positives. The cascade of R-CNN stages are trained sequentially, using the output of one stage to train the next. This is motivated by the observation that the output IoU of a regressor is almost invariably better than the input IoU, where nearly all plots are above the gray line. It suggests that the output of a detector trained with a certain IoU threshold is a good distribution to train the detector of the next higher IoU threshold. This is similar to boostrapping methods commonly used to assemble datasets in object detection literature. The main difference is that the resampling procedure of the Cascade R-CNN does not aim to mine hard negatives. Instead, by adjusting bounding boxes, each stage aims to find a good set of close false positives for training the next stage. When operating in this manner, a sequence of detectors adapted to increasingly higher IoUs can beat the overfitting problem, and thus be effectively trained. At inference, the same cascade procedure is applied. The progressively improved hypotheses are better matched to the increasing detector quality at each stage. This enables higher detection accuracies.

  After heavily surveying the above mentioned documents and weighing our options we decided to use YOLO for our project due to it being fairly simple and easy to use out of the box as well as the widely prevalent resources for it.

RELATED WORK

We have combed through different research documents and implementations regarding this subject and came to the initial of question of if we would try using a one-stage or a two-stage implementation. Through the study of reference [7] we learned the characteristics of the R-CNN derived two-stage implementations and whilst they prove to be superior to the one-stage implementations in most metrics, they seemed to be far too complex for the scope of our project. As such, we decided on a one-stage method and after we looked through materials [4] and [5] we ended on using YOLO for our project.

Fig. 2. One of the first photos made for the dataset

METHOD DESCRIPTION

The main defining feature of our project compared to the others is that we had to procure our very own data set to train our YOLO-based network on. Whilst this did seem like an

easy prospect as first, we soon realised the magnitude of the task. After acquiring the images we used a free online tool called CVAT (Computer vision annotation tool) to annotate every image individually. Annotating an image consists of drawing a bounding rectangle around the object you wish the algorithm to detect. With the annotating process done, we used another online tool called Roboflow to split our annotated images into 3 categories: training images, which would be used to train the AI itself and make it recognise the different objects, validation images, which are also used in the training process to ensure that the selected objects are correct and test images which serve the purpose of testing if the AI's training was successful by trying to verify the human placed bounding rectangles to the ones placed by the AI. Aside from just splitting the images into the 3 aforementioned categories, Roboflow also has the ability to artificially inflate the given dataset by splicing, carefully placing artifacts and applying different filters to the images, leading to a greater number of photos to train the AI with aswell as a greater variety of them (See Figure 2).

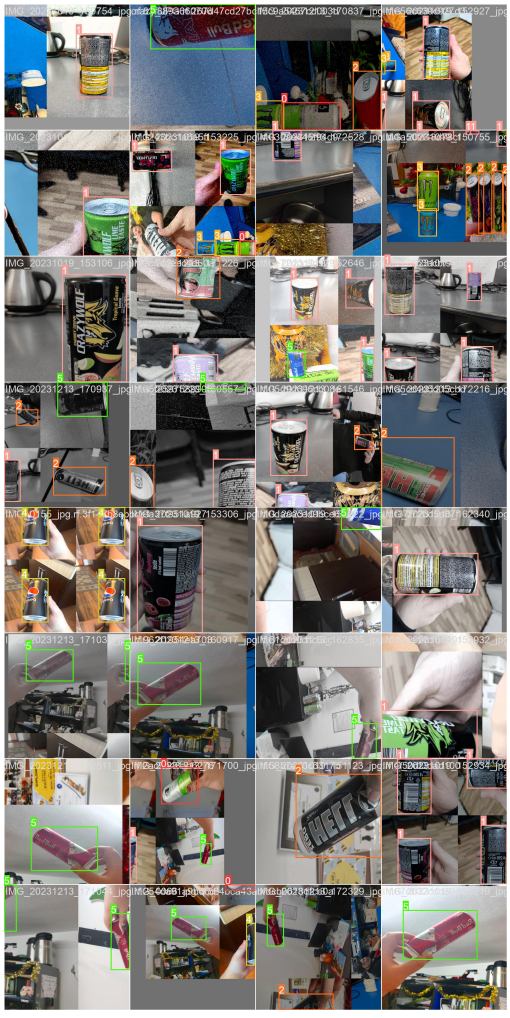

Fig. 3. Example photos of how Roboflow modifies its source material

With all of the data set fully formated we can take to the actual coding phase of the project where the simplicity of the YOLO model comes into play. By importing our data from the Roboflow directly into our Colab Notebook with a few lines of code we can start training the model just by calling the 'train' function and simply specifying the data set it needs to use as well as the epochs, which entail the amount of times it should train itself on the data set. With all of that done, we have a functioning model that can be tested by importing it into another code file and giving it an image link alongside the "predict" command.

Preliminary Results and Conclusions

After researching the available technologies and deciding on what to do we handily made a prototype which consisted of a trained model that can fairly confidently detect cans with confidence of around 80-90 percent. On top of that we had already gathered a bit over 200 images in our data set without taking into account the artificial inflation induced by Robobflow (which we were not using at that moment). One problem that we did run into however is that we initially wanted to make the model detect the individual flavors of the beverage, this however proved a herculean task with the relatively small size of the aforementioned data set, as such we had to relegate ourselves to just detect the brand as a whole. Biggest issue with this however was that we only had data for 4 of 6 types the brand has, making the model have a fairly difficult time recognising the 2 missing brands, which did hurt its prediction confidence quite a lot.

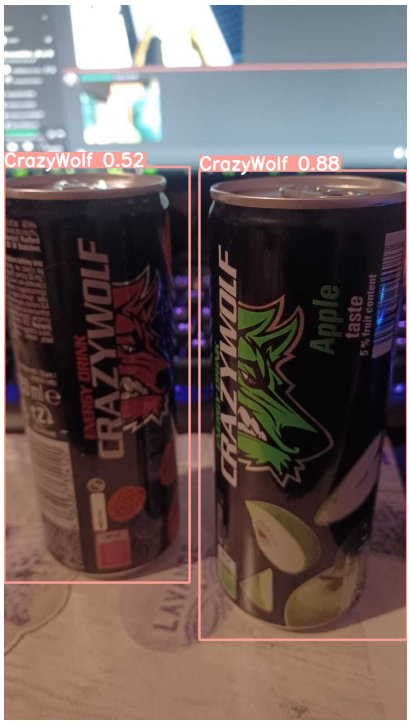

Fig. 4. Result from first prototype

A positive aspect we did notice however was the model's ability to recognise a can with the same prediction no matter

what side it was facing, this was mainly due to our decision to rotate the soda cans as we photographed them. A second issue that came with procuring our own data set was definitely that we did not factor in the background for all the photos which led to the program having severe problems when it was faced to complex backgrounds as well as being confused when faced with cans in awkward positions such as having them be laid on their side, on top of each other, at an angle or having them be very close one to another, to the point where the edges weren't clear. An issue that we were aware of and we tried our best to and hopefully did avoid is over fitting the AI, which happens when it is trained with the same images for far too many times. The conclusions we can draw from the current stage of the project is that the data set needs to be greatly enlarged and present a greater variation of positions for the objects as well as a far more varied backdrop for them.

Advancements in dataset creation and manipulation

Starting off, we did not entirely understand the way the model would dissect and analyze the pictures we were providing to learn. As such most of the data we were creating was made up of cans simply standing up straight, all of them unopened (much like the image in Figure 1) with the only variation from one picture to the other being the camera angle or it being slightly turned to the right or to the left. This method left a lot to be desired when we got to the actual training process. As mentioned above, any model we managed to create had difficulties recognizing anything that was not in the strict straight up shape it was accustomed to. As such, when further expanding the dataset to contain other brands of canned beverages, we undertook a much more minute work flow regarding the actual photographing of the cans. They were flipped, opened, had their keys taken out, dent in places, slightly obscured, turned on their sides, stacked on each other and so on. One aspect which we were surprised never came to our attention was that when someone takes a picture of a can, they usually do so whilst having it in their hand, at that point we did not have a single picture in our repertoire which contained a point-of-view picture of a hand holding a can, as such we made sure to include at least some for each type of can to better our chances of properly pin pointing their location.

Another aspect which we touched on above was the few background changes we had in the dataset we used for the initial prototype. For the last version of the program we tried to have most of the pictures done in different locations or at the very least include slight changes. We also attempted to introduce pictures with human actors in them to better encapsulate a situation which the model might find itself faced with: that of a person holding a can. For such a circumstance we attempted to make pictures featuring most types of people with different, all of them with different levels of obscurity being applied to the main focus of the photo, the soda can. Despite our somewhat lax implementation of the above, the model still has a much easier time recognising human figures holding beverages.

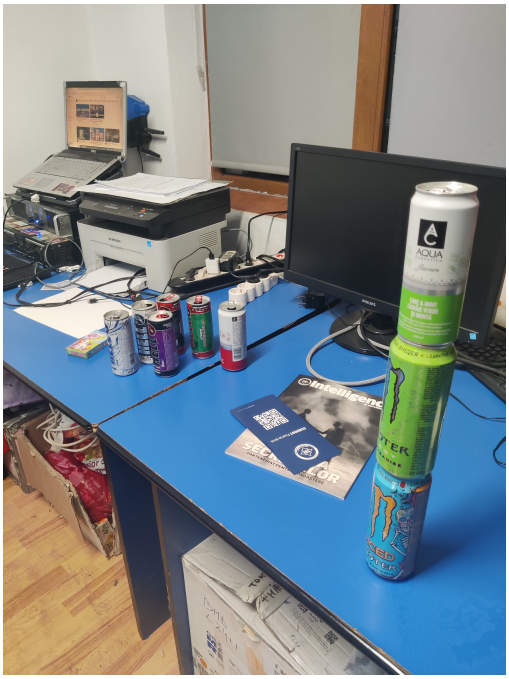

Fig. 5. Picture from the later part of the dataset

Improvements in confidence and precision

At the moment, with the latest version of our dataset, after training a model it can accurately predict 6 different brands of canned beverages each with a varying number of variations ranging from 3-4 to upwards of 6. Currently the model has no issues recognizing cans in any kind of usual position such as on a table or in a persons hand. It also performs very well when faced with cans in uncanny situations, unlike the first prototype, it has no problems recognizing a soda if it is sat on its side, obscured or any other such situations. We have however noticed that the model really suffers with one specific circumstance: a long row of cans. Whilst it can always recognize a number of the sodas in its view, the recall level is fairly low, on every occasion it fails to notice at least 1 or 2 of the cans (See figure). Another shortcoming which we did not manage to filter out is that we the introduction of many more new object classes sometimes, very rarely, the model manages to mix up the different brands, ending up with an incorrect prediction. We believe this to be largely because of the rather unassuming nature of the average can. With every object being of cylindrical shape, the main distinguishing feature the model can go off of is color and the general aesthetic of the can in question. This becomes rather bothersome when you take into account that most of the brands encapsulated have wildly different colors from flavour to flavour. For example, for the category Monster energy drink we have 3 types present: green, blue with orange accents and black with green accents. Due to the diversity of the colors the model struggles to assign

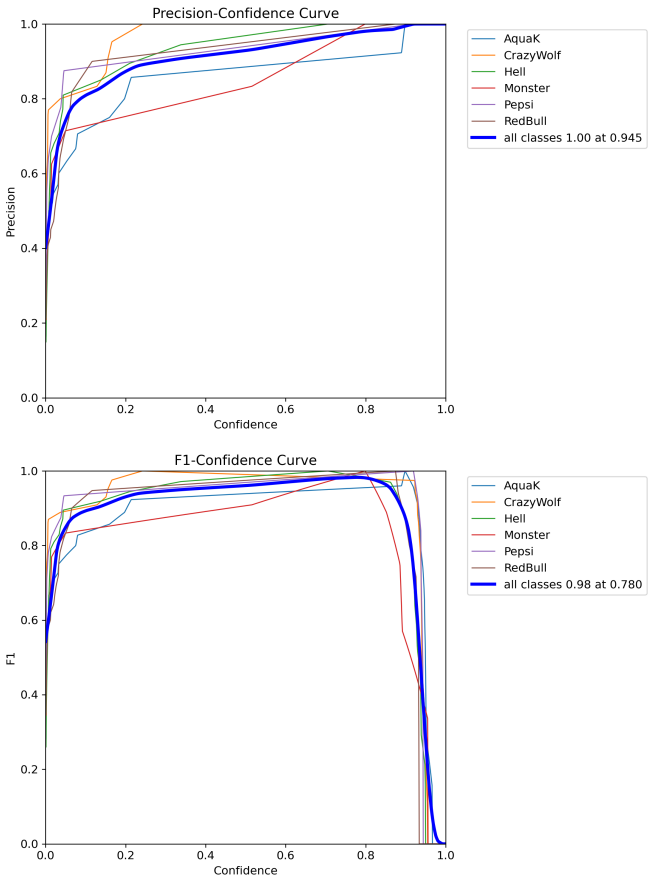

Fig. 6. Precision, recall and confidence levels of the model
(The F1 score calculates the balance between precision and recall. If the F1 score is high, precision and recall are high, and vice versa.)

a definitive characteristic to it. And due to other classes also suffering from the same issue we end up having certain cases in which cans of the same color are sometimes mistaken for one another. Still, an object not being recognized entirely is a much more often occurring issue. Last but not least, something that happened a couple of times for the objects which were annotated last is that due to a new annotating technique the AI very rarely draws a much larger bounding box for an object.

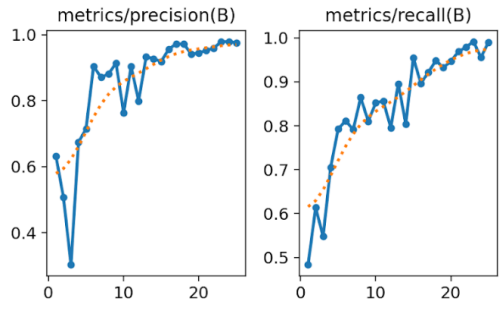

Fig. 7. Recall and precision metrics at the end of training

Whilst the above lines might have been painting a quite grim picture, reality is actually quite far from it. The model in itself has made massive strides in its precision and confidence from the initial prototype, now sporting an above 95 percent precision and recall in its matches.

Fig. 8. Whilst the model manages to catch some of the objects, it still fails to match all of them

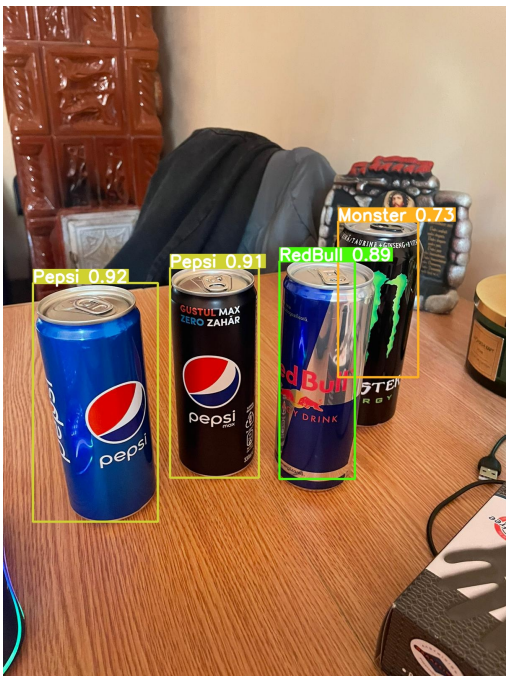

Fig. 9. Result from the last version of the model

III. Conclusions

Whilst we do believe in the end the project is quite accomplished, we still would have greatly profited from certain actions we could've taken or other paths we should have gone on instead. For one, it would have been quite useful if we properly photographed the pictures from the first dataset as it would've saved us the headache of having to retake some of the pictures as well as having better end results with for the brand that we initially started with. Also for starters it would've been much easier to start recognizing brands right away instead of having to make an entire first version which barely worked for different flavours instead. Another thing we could have improved was our understanding of how annotating

the materials themselves was done, as we didn't know that we could mark certain objects as obstructed to better train the AI.

REFERENCES

[1] Mask R-CNN IEEE Paper
[2] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
[3] Cascade R-CNN: Delving Into High Quality Object Detection
[4] SSD: Single Shot MultiBox Detector
[5] YOLO9000 Better, Faster, Stronger
[6] Road Damage Detection Using RetinaNet
[7] Rich feature hierarchies for accurate object detection and semantic segmentation