Price is All You Need

Marco Boucas
Magali Morin
CentraleSupélec
{firstname.lastname}@student-cs.fr

Abstract

The objective of this paper is the design of a vision system that detects products and their associated prices from a picture of a supermarket shelf. The problem is decomposed into two parts: i) detection of the products and the different price labels on the shelves, and ii) identification of the monetary value of a product from its corresponding label.

Our code is publicly available on github. The data and trained models are available here: Weights and Data folder (you just need to put the 2 folders in the root directory of the project.

To get near a real production use, we developped a *streamlit* interface, that helps see the different steps of the project (from the product detection, using YOLO, to the price tag detection and the value of the price identification). A small video demonstration is available here

1. Introduction

1.1. Context

We have seen recently an increase in terms of food prices, for instance, the cost of fruits and vegetables increased by around 9% compared to 2019. Hence, a familly of four people would need 700€ to be able to eat healthy food according to *Familles Rurales*. And this trend is more general, impacting also the price of pasta, semolina, couscous and fish. In general, the rise is around 1.4%.
The idea of this project is to help low revenue people to eat healthy while complying with their goal of reducing the food cost, by pointing them to low-cost products.

2. Related work

Object detection was originaly made using methods at the opposite range of machine statistical learning. Such methods, like SIFT [9] or HOG [2], were based on the computation of gradients of the image in order to generate features of an image. It is first in the 1990s with the first appearance of CNNs [14] and the regain in interest in 2012 with Krizhevsky et al. [1], that using deep learning (more precisely Convolutional Neural Networks) for computer vision was considered.

In 2014, the first application of deep learning for object detection comes into life with *R-CNN* [11]. The main idea for this model was to use a deep learning model to classify different regions, using a model pre-trained on a large auxiliary dataset (ILSVRC2012 classification). The regions were generated using a different method, such as the Selective Search [13], based on a hierarchical grouping algorithm, to generate around 2000 region proposals. This method leads to higher object detection performance on PASCAL VOC, wtih a mAP of 58.5%, to compare with the 34.3% for a DPM method based on HOG features.

The main pain-point of this approach is the time spent computing the results on each region proposal, which is not suitable for a production approach. Advances such a *Fast R-CNN* [3], which applied the region proposal computation on the feature map of a CNN, reducing the cost of the final classification of each region, exposed the region proposal computation as the final bottleneck.

To achieve near real-time object detection, *Faster R-CNN* [12] was released in a paper in 2015, getting rid of the usual region proposal methods and replacing those with a *Region Proposal Network*, with a shared convolutional layer with the object detection networks, reducing drasticaly the cost for computing proposals.

When it comes to real-time object detection, we cannot avoid speaking about *YOLO* [5]. It proposed a unified model that both generate the boxes, assess the confidence for each box and the probability of each class for it. It has to be noted that YOLO is one of the easiest way to do object detection in production (around 45 frames per second for the first version of the model).

As explained in the context, we are using the dataset from the **CVPR2021** Product Pricing Challenge. This challenge is now ended, and we decided to inspire our project by looking at the top participant's reports.

In the winning team report *USTC-NELSLIP* [6], they decided to use an object detection model to detect the price

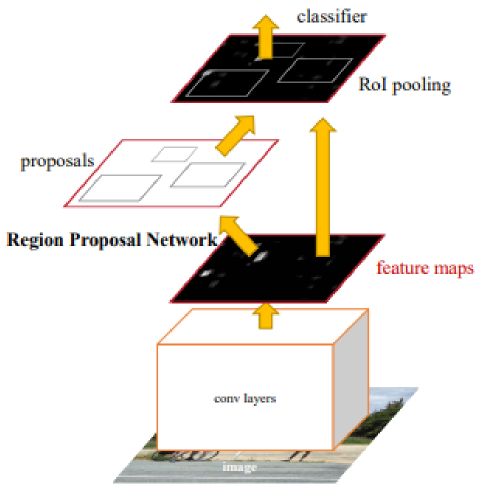


Figure 1. The architecture we will be using (Faster R-CNN). The conv layers that first operate on the image correspond to the backbone we will be changing.

tags. In fact, this process has been done for the top3 teams of the challenge. They decided to develop the model step by step, with different processes taking care of different features. One major point of interest in their report is the definition of the confidence of the global pipeline for one particular price tags. Because a result comes after going through the all pipeline, they carefully designed their confidence for each part of their process, and multiplied the end results. For the model that handles matching products and price tags together on the image, they used the following confidence function:

$$C = \frac{(\sum_{k=1}^{n} D_k^2) - D_i^2}{\sum_{k=1}^{n} D_k^2} \tag{1}$$

with $(D_1, D_2, ..., D_n)$ being the distances between one product and all the price tags, D_i denotes the smallest distance for this product. Even if this task is far much easier than finding the right price written on the tag, it is important to take into account the confidence in the tag itself.

Considering the creation of the price tags dataset, we decided to use something similar to what is proposed in the 2nd solution report [8], which is basicaly an active method (*human in the loop*) method to annotate the dataset.

Optical character recognition (OCR) is a technology used to convert any kind of image containing written text into machine readable text data. The earliest use of optical character recognition can be traced back in the 1970's when Ray Kurzweil created a reading device for the blind [7]. In the 1990's, OCR became very popular to digitize historical documents as it saved a lot of time and reduced inaccuracies and typing errors when manually retyping a text. Today, OCR services are widely available to the public and are mainly used to scan and store documents on smartphones [10].

In our case, we want to extract the price from the pricetag. As some labels are printed and other written by hand, there are different fonts and structures of the text which raised some difficulties to the task. Contour detection allowed us to detect the text appearance on the image in order to recognize the character one by one.

3. Price Detector

We aim to develop during our project a *Price Detector*, which should be a system that takes an image of a supermarket shelf and automatically detects the different products locations and their associated prices. To make this work, we decided to break the process into smaller steps, each one of them taking care of one essential part of the job:

- A product detection (using YOLO)
- A price box detection
- A matching between products and prices
- An OCR (to extract the value of the price)

4. Experiences

4.1. Price Tags Dataset

For the price tags detection, we proceeded using the following steps:

- Annotate a few images (around 50) with some of the most common price tags + some more exotic ones
- Use our model to annotate new images
- Fix the annotations if needed
- Retrain the model on all the newly formed dataset
- Continue until the images are labelled
- Split the dataset into train and set (the model used to annotate is then destroyed)
- Find the best model using hyperparameter tuning, using for a measure of performance using the IoU Metric

4.1.1 First results after 50 images

Using the first dataset (containing around 50 images, for a total of around 1000 tags), we can already analyze the first batch of images, to see the potential pain points of the model (and when it performes the best).

From those samples of model output on unseen data, we can say a few things:

- Classic price tags are for the main part correctly detected, and the box contains the relevant information (price here).

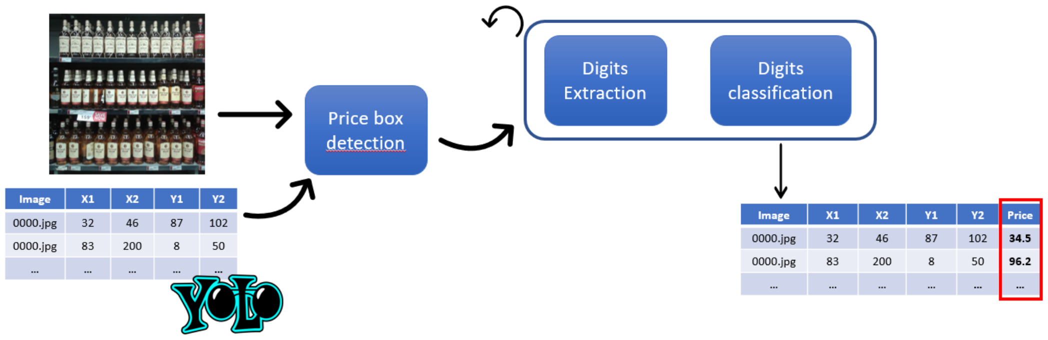


Image	X1	X2	Y1	Y2
0000.jpg	32	46	87	102
0000.jpg	83	200	8	50
...	...	...	...	...

Image	X1	X2	Y1	Y2	Price
0000.jpg	32	46	87	102	34.5
0000.jpg	83	200	8	50	96.2
...	...	...	...	...	...

Figure 2. The representation of our complete pipeline

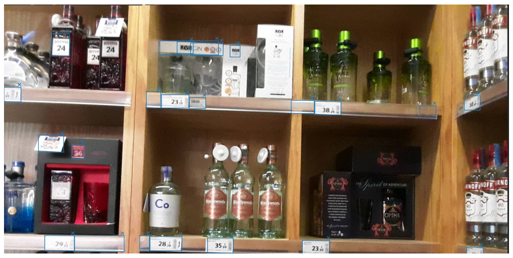


Figure 3. Example of the model predictions, with quite good results in overall, some bottle tags are misclassified as price tags.

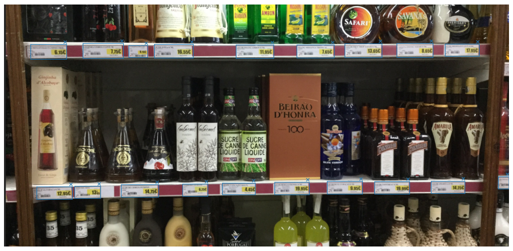


Figure 4. For classic bottle tags, the model seems to correctly identify the price tags.

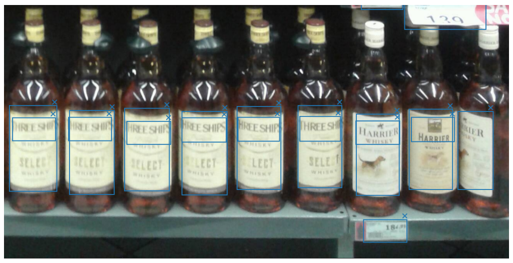


Figure 5. In some cases (mostly wiskey bottles), the model is confused with the bottle tags.

- More original price tags (like in figure 4) are also detected, but contain a lot of boxes inside. We decided to use some hierarchical based algorithm to remove those ”child” boxes

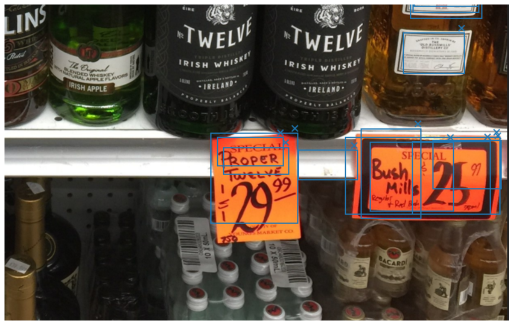


Figure 6. Some more exotic price tags are correctly detected, even if the number of boxes will have to be dealt with.

- Some product tags are also detected as price tags, which is not good. To avoid that, we will remove predictions that overlap too much with a product.

4.2. Price Tag Detection

Using our labelled dataset, we aim to find the model that find all the price boxes. We will use a *Faster R-CNN* model with different backbones and vary some hyperparameters to find the model that maximizes our relevant metric. From a user point of view, the key metric seems to be to correctly matching the tags, and to avoid having to much tags wrongly placed on the image. Even if we are selecting the tags based on the distance, we can’t afford having tags placed incorrectly near each products.

4.2.1 IoU Metric

Our metric will be a generalized **IoU (Intersection over Union)** over all the price tags. The IoU, in the case of only one box on an image, can be computed easily:

$$IoU = \frac{AreaofOverlap}{AreaofUnion}$$

In a more generalized situation, with $y_{i,j} = 1$ if the pixel (i, j) is in one of the boxes of the true price tags (else 0), and $\hat{y_{i,j}}$ for the predicted boxes, we can compute the IoU using the following formula:

$$IoU = \frac{\sum_i \sum_j y_{i,j} \wedge \hat{y}_{i,j}}{\sum_i \sum_j y_{i,j} \vee \hat{y}_{i,j}}$$

4.2.2 Hyperparameter tuning

We will be fine-tuning a Faster R-CNN model, with different backbones and different hyperparameters. In order to find the best set of hyperparameters for our fine-tuned model, we will split our dataset into 3 sets (train, val and test with the usual 70%, 15%, 15% partition), and we will evaluate the ”performance” of several hyperparameters on the validation set).

As for the exploration method, we will apply a Random Search instead of a Grid Search, due to time and computation constraints. The process will be iterated several times, to focus the search each time on the most promising parameters (distributions will be narrowed accordingly at each stage of the process).

We will be optimizing the following hyperparameters during the random search:

- The backbone, either *resnet50* or *mobilnetv3*
- The learning rate, from a log uniform distribution ranging from $1e^{-5}$ to $1e^{-1}$
- The weight decay, either null or from a log uniform distribution to $1e^{-3}$
- The number of epochs for the fine-tuning, from a uniform distribution ranging from 2 to 10

4.2.3 First results analysis

After running the hyperparameter tuning on several sets of parameters (learning rate, number of epochs, ...), we can check the model's results using our metric (see Table 1).

From this first round of train, we can gather some insights about the ”best model configuration”:

- All the top models have learning rates around 10e-3. We will try to dig up around this value later. These values are close to those used in the training of the original model (from 0.1 to 1e-4 according to the original paper [4]) for Resnet50 and mobilnets models.
- The ideal number of epochs seems to be a bit less than 10, even if the fifth model performs quite good with only 3 epochs and a bigger learning rate. *Suggested also by the training curves*
- The best model after this first round is a resnet50 model, with 0.13 points more in terms of mean IOU, which is quite impressive compared to the second model for instance.

4.2.4 Resnet50 models

This first set of training results raises some questions, we will take a closer look at the worst samples for the winning model to verify its performance (and the best ones too). We will also add an early stopping into the pipeline, to try to find the best time to stop the training.

After the second training, which stopped after the 13th epoch, we have the following evolution of the IoU score:

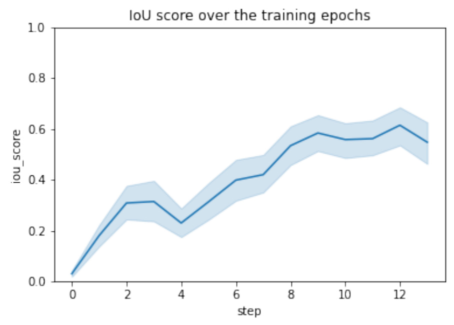


Figure 7. IoU metric during the training of the model

But this is not the final score, using our test dataset and also our method to remove overlapping price tags and products (which uses YOLO to identify the products), we have a final IoU Custom score of **0.66** mean (**0.59** without the overlapping method). The test set is small because it takes time to generate a large dataset. It consists of 20 samples (images with multiple price tags on each, about 20/30 per image). Here are the full results:

Mean IoU	std	min	25%	50%	75%	max
0.661	0.16348	0.284	0.579	0.688	0.756	0.889

After careful examination of the validation test (mainly on the lowest rated images), the model sometimes detects price tags in places where they should not be, mostly due to the perspective (bottle tops that are not at the same distance from the camera than all the products, appear bigger). Also sometimes, due to the light reflection on plastic bars, price tags appear almost white. To make more sense of those results, you can see at the end of this document some examples of the final model predictions: Some price tag detections with the final model (test set) (green: truth, red: predictions)

4.3. Matching prices and products

One small step of our pipeline is the matching between the different price boxes and the product boxes. After some examinations of the training set, we can make a few assumptions about how the prices are located related to their corresponding product:

	IOU Mean	IOU Max	IOU Min	IOU std	Model Type	Learning Rate	Epochs
23	0.583	0.832	0.295	0.157	resnet50	2.4e-3	8
13	0.456	0.705	0.206	0.138	mobilnetv3	3.1e-4	11
20	0.453	0.677	0.253	0.132	mobilnetv3	7.5e-4	7
15	0.445	0.652	0.192	0.135	mobilnetv3	4.5e-4	9
6	0.435	0.672	0.214	0.144	mobilnetv3	1.3e-4	3

Table 1. Hyperparameters tuning

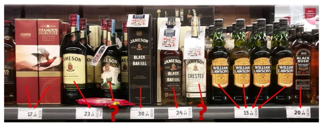


Figure 8. Here is a possible match between the products and their price tags, sometimes the price tag seems not to be written

- The prices are generally below (a few exceptions, with sometimes the price written on the products, but those elements are very rare in our dataset and considered as outliers)
- When a batch of products of the same type is displayed, they are on the same shelf and their price is below the group.
- In most of the cases, the bottom of the product is near the top of the price tag

Considering those observations, we decided to move forward with a pondered L_2 distance, which penalizes the y axis more than the x axis (corresponds to the 2^{nd} hypothesis).

$$D(x_1, y_1, x_2, y_2) = \sqrt{(x_2 - x_1)^2 + \lambda * (y_2 - y_1)^2}$$

With $\lambda = 2.1$ being the parameter that prefers x variations over the y variations. This value has been selected as the best proposition using a cross-validation over our dataset.

4.4. Price value using OCR

We used an OCR to predict the price correponding to each price tag. In our dataset there are four types of price tags.

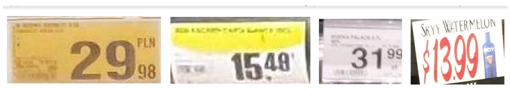


Figure 9. Types of price tags

In order to predict the price, each price tag underwent a series of transformations.

4.4.1 Detecting contours

A price tag undergoes a series of transformation in order to make the contour detection more efficient.

1. Original price tag image
2. Conversion into black and white image
3. Reversion of the colors of the price tag
4. Detection of the contours with the library open CV

The picture below shows the different steps of the transformation.

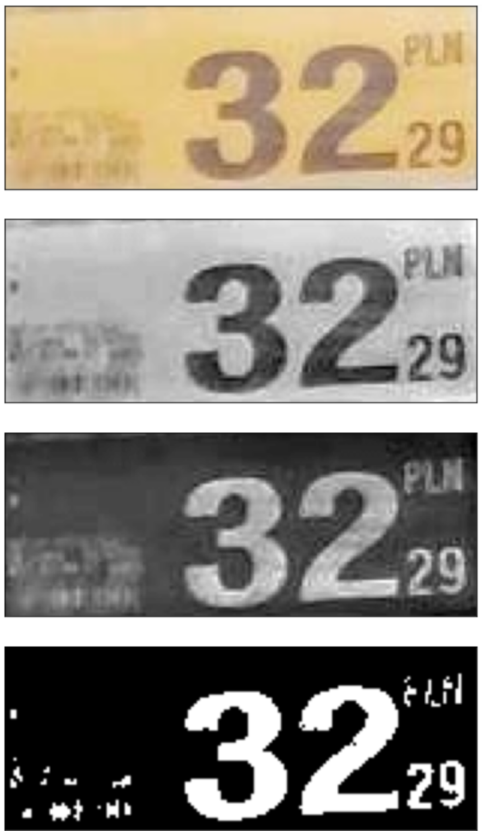


Figure 10. Steps of the transformation of a price tag for contour detection

Once digits contours are detected, boxes are created around them to crop the price tag on a single digit image.

4.4.2 Filtering boxes

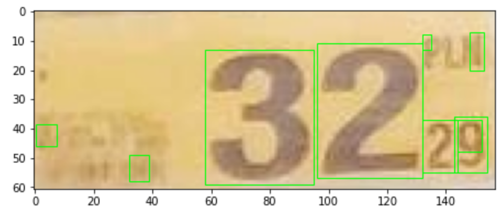


Figure 11. Delimiting digits with boxes

The boxes are then filtered according to the specific properties of a price.

- Digits have greater height than width
- Digits shouldn't be overlapping
- Digits can't be too close to the edges of the price tag
- Digits have to have a minimum size

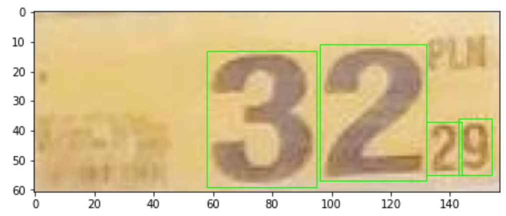


Figure 12. Filtering boxes with digit specificities

4.4.3 Predicting digits

Once digit images are extracted from a price tag, we use a CNN model to predict the digit. The CNN model was originally trained on the MNIST dataset and we finetuned it on the digit images dataset we created.We thought it best to do this for several reasons : the small size of our dataset, to benefit from the training on the MNIST and the patterns already learned by the model). We created a dataset of anotated digits thanks to *Jupyter BBoxWidget*. Each type of price tag of our dataset is represented in our training, validation and test set.

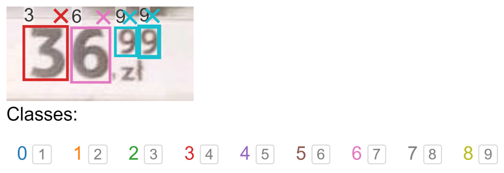


Figure 13. Annotate digits with *Jupyter BBoxWidget*

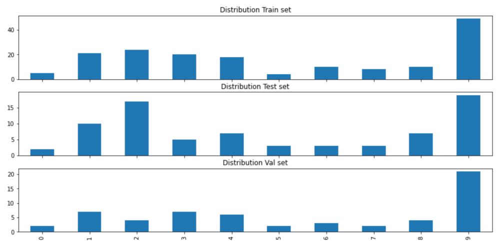


Figure 14. Distribution of digits in the train, test and val set. As expected, the digit "9" is over-represented in our datasets, we carefully removed some of those to make the dataset more balanced. But we need to keep in mind that some digits are not common in prices.

We tested two different finetuning. First, to finetune all the layers of the model and then to finetune only the last layer. We obtained higher performances when finetuning all layers so we saved this model and used it to predict the digits on the price tags.

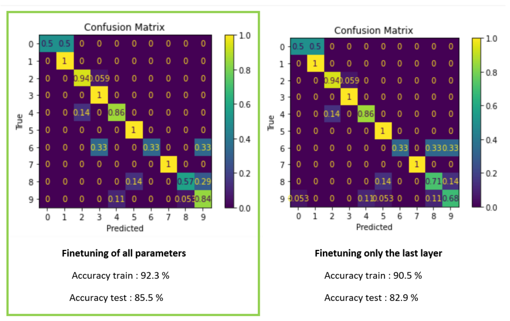


Figure 15. Performances of the finetuned models. Those confusion matrices shows us that the model is sometimes confused when confronted against some digits. For instance, some digits "4" are confused with the "2", same for "3" and "6". And there are some confusions with the "9", probably because of the dataset not being totally balanced even after removing "9".

The MNIST dataset consists of 28 X 28 black and white digit images. We transformed digit images from our dataset to look like images from the MNIST dataset in order to obtain better performances. Black borders were added on the sides and the size was changed to 28 X 28.

- Black borders were added on the sides of the image
- Size was changed to 28 X 28.

Pour l'entraînement, nous avons choisi un batch size de 100 et un learning rate de 0.001.

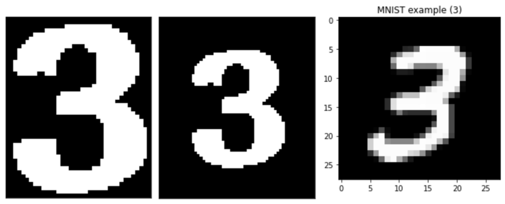


Figure 16. Digit transformation (from left to right) to look like the MNIST dataset (on the right)

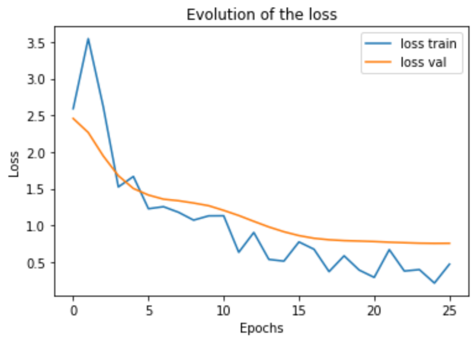


Figure 17. Evolution of the loss when training the model

4.5. Overall Pipeline

The initial challenge was to predict the price of some products, we plugged our pipeline (detection of the price tags + OCR to detect the price) together and used it to predict the price of each product. The final results are not that good because the OCR struggles to identify the prices correctly, but we still have some good matches.

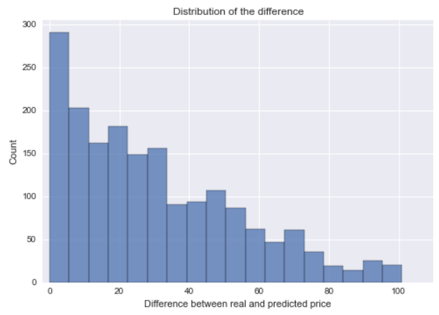


Figure 18. Distribution of the difference between the predicted price and the real one, in euros

To get a better idea of how well the model is predicting the price, here are some numbers:

Proportion	Difference in euros/dollars
0.8%	0.00 $€ (perfect match)
4.9%	0.50 $€
10%	1.00 $€
14.8%	5.00 $€
50.1%	25.00 $€

Table 2. Proportion of the dataset for which the difference is below the threshold. As you can see on the results above, the model is not able to correctly predict the price most of the time. In fact, if we consider a difference of 1€ (which is already high sometimes), the model is able to predict the correct price only 10% of the time

5. Conclusion

As we can see, at the end of the project, our model is able to identify the different price tags for each product, achieving good results on the classic price tags (the european ones, that are formatted properly). One pain point of our model is when it is confronted to unformatted price tags, the model might not be able to generelize correctly (even if it shows quite good results in our dataset).

What could be the next steps would be to combine the object detection model and the price tag detection model, in order to have only one model. Furthermore, using the YOLO architecture, it would really allow a real time object detection (and even on mobiles, with the small versions of Yolo, like for instance *Tiny-YOLO*).

The OCR could be improved by training the model on a larger database of digits from price tags of our dataset. Moreover, changing our method to detect digits, by using another object detection model, might be a solution to avoid detection problems when digits are very close from each others.

One last pain point of our model is to understand what kind of money is used in the tags, because the current formatting will not take into consideration this. For instance, in Europe, we are using euros, and the prices are usually less than 100€ for products of our everyday-life, but when it comes to countries were the value of a product is much higher (more than 100 units of money), our model might not be able to tackle this properly.

6. References

References

[1] I. Sutskever A. Krizhevsky and G. Hinton. Imagenet classification with deep convolutional neural networks, 2012. NIPS. 1

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection, 2005. CVPR. 1

[3] R. Girshick. Fast r-cnn, 2015. ICCV 2015, arXiv:1504.08083. 1

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 4

[5] R. Girshick A. Farhadi J. Redmon, S. Divvala. You only look once: Unified, real-time object detection, 2015. arxiv:1506.02640. 1

[6] Z. Cui H. Xie Z. Zhang Y. Yu W. Su F. Gao J. Yu, L. Zhang and F. Shuang. A solution for product pricing in densely packed scenes, 2021. https://retailvisionworkshop.github.io/pricing_challenge_2021/. 1

[7] Forough Karandish. Un guide facile pour comprendre la reconnaissance optique de caractères (ocr), 2021. 2

[8] A. Kozlov. Read and match: a strong baseline and 2nd place solution to product pricing, 2021. https://retailvisionworkshop.github.io/pricing_challenge_2021/. 2

[9] D. Lowe. Distinctive image features from scale-invariant keypoints, 2004. IJCV. 1

[10] Matt Mills. Ocr, histoire et fonctionnement de cette technologie de reconnaissance, 2020. 2

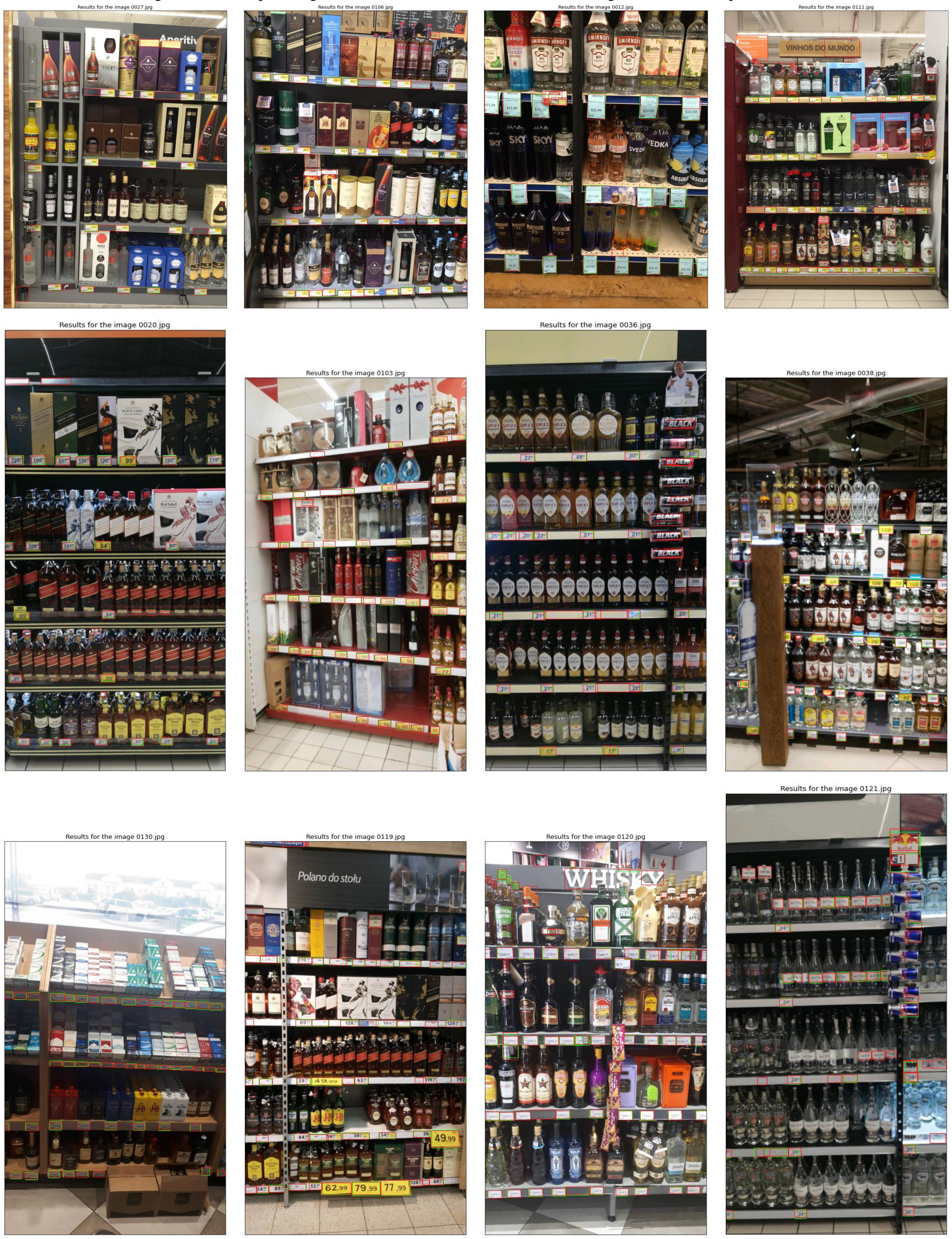
[11] T. Darrell R. Girshick, J. Donahue and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014. 1

[12] R. Girshick S. Ren, K. He and J. Sun. Towards real-time object detection with region proposal networks, 2015. arXiv:1506.01497v3. 1

[13] Jasper Uijlings, K. Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition, 2013. IJCV, 104(2): 154-171, DOI:10.1007/s11263-013-0620-5. 1

[14] Y. Bengio Y. LeCun, L. Bottou and P. Haffner. Gradientbased learning applied to document recognition, 1998. Proc. of the IEEE. 1

Figure 19. Some price tag detections with the final model (test set) (green: truth, red: predictions)


Results for the image 0027.jpg
Results for the image 0106.jpg
Results for the image 0012.jpg
Results for the image 0111.jpg
Results for the image 0020.jpg
Results for the image 0103.jpg
Results for the image 0036.jpg
Results for the image 0038.jpg
Results for the image 0121.jpg
Results for the image 0130.jpg
Results for the image 0119.jpg
Results for the image 0120.jpg