Comparison Of Modern Object Detection Algorithms

Amrou CHOUCHENE
Telecom-Paris
amrou.chouchene@telecom-paris.fr

Wassim BOUATAY
Telecom-Paris
wassim.bouatay@telecom-paris.fr

Abstract

As beginners in the field of Machine Learning and Deep Learning, most problems that we had encountered so far were Classification or Regression problems. So, we take advantage of this personal project to try something new and something that we wanted to do but that we didn't get the opportunity to do which is the Detection problems.

In this project, we will look into different modern Object Detection algorithms precisely R-CNN, Faster R-CNN and YOLO. In the first step, we will understand and explain in depth each of these methods. In the second step of the project, we will evaluate these methods on COCO validation set of 2014. We will compare them through the different experiments that we will make.

1. Introduction

1.1. Motivation

The Object Detection problem represents one of the biggest parts of computer vision along with image segmentation and image classification. It helps in understanding and analyzing scenes in images and in videos.

In the classification problem, we are looking for a label to the image. In the segmentation problem, we are looking for similar groups of pixels that belong to the same object. But, in object Detection the goal is to locate the position of one or more objects and to determine the class of each one.

1.2. Applications

Object detection is applied in different applications. Here is a list of some possible applications:
- Video-Surveillance
- Crowd Counting
- Self-driving cars
- Anomaly Detection

1.3. Problem definition

The goal of the project is to study different modern Object Detection algorithms precisely R-CNN, Faster R-CNN and YOLO. We will analyze and understand each method then we will compare them.

First, we will verify the fact that two-stage detectors (YOLO) are more appropriate for real-time detection as they are faster than single-stage detectors (R-CNN and Faster R-CNN).

Second, we will evaluate these methods using different metrics: precision, recall, IoU and mAP.

2. COCO Dataset

In this project, we will work on the famous 'COCO' dataset. 'COCO' stands for 'Common Objects In Context'. It is one of the most used datasets when it comes to object detection, key-point detection and segmentation.

In this project, we worked on the validation set of 2014. It contains images, bounding boxes and labels for 40504 images in 80 different classes.

To better understand the content of the dataset, we first counted the occurrence of each object in the whole image. The results are shown in figure 1.

As we can see, the dataset is heavily biased towards the most frequent class which is $'person'$. The class $'person'$ appears more than 6 times the second class $'car'$ which also appear almost 3 times the tenth class $'handbag'$. So, our dataset is unbalanced. Hence, we will test the performance of our algorithms on the whole dataset, then, on each of the top 10 classes independently and finally on the remaining 70 classes which present the under represented (rare) classes.

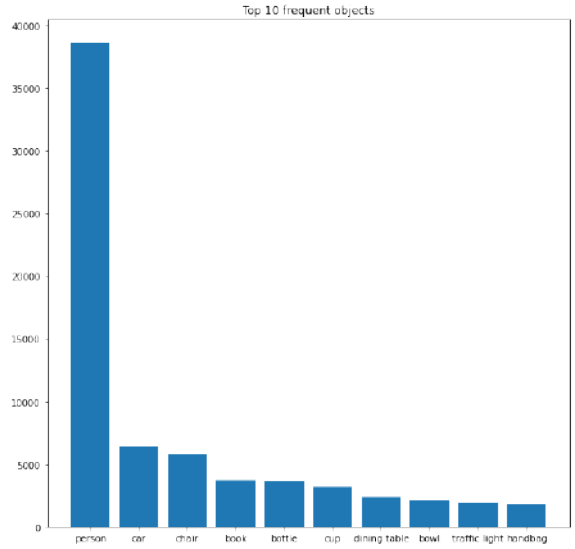

Figure 1. Top 10 most frequent objects

3. R-CNN

3.1. Introduction

Region-based Convolutional Neural Networks also known as R-CNN was proposed by Ross Girshick et al. in 2014 [1]. The R-CNN is one of the two-stages detectors. In the first stage, it extracts regions of proposal. Then, it computes the features using CNN and uses SVM to classify each region and to determine a correct bounding box.

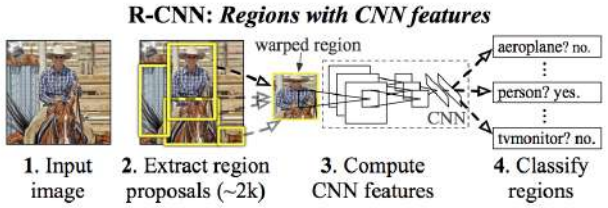

Figure 2. R-CNN model

After the stagnation of performance in the period of 2010-2014, R-CNN came as a simple scalable algorithm with an improvement in the mean average precision (mAP) by around 30% compared to the best result on the VOC 2012 dataset achieving an mAP of 58.5%.

R-CNN can also be extended for segmentation problems. When it was tested on the VOC 2011 dataset, it achieved a segmentation accuracy of 47.9% slightly better than the current best algorithm 47.6%.

3.2. Related works before R-CNN

Before R-CNN, the first detectors were based on methods like SIFT and HOG. Then, sliding-window detectors arose. The main problem with sliding-window detectors is that the strides are very large in the units high up in the network, resulting in challenges to localize precisely the objects.

The best algorithm before R-CNN was OverFeat [2]. This algorithm used sliding-window CNN too. It won the ILSVRC 2013 detection with an mAP of 24.3%, whereas R-CNN performed 31.4%.

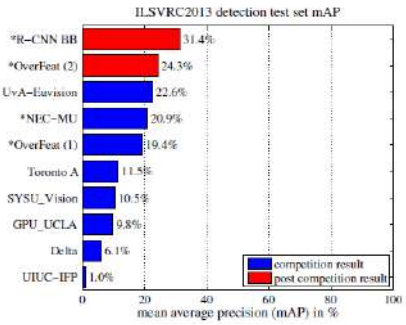

Figure 3. R-CNN vs the other methods in ILSVRC 2013

Figure 3 shows a comparison between RCNN, OverFeat and other methods. It is clear that R-CNN outperforms all the methods at that time.

3.3. How it works

3.3.1 Region Proposals

In the Region Proposals step, we generate a set of candidate detections. From the input image, we chose 2000 regions of interest. These regions are category-independent. In the case of R-CNN, we use Selective Search [3] to get the 2000 proposals.

There are different methods to do the Region Proposals such as Objectness, Constrained Parametric Min-Cuts, Selective Search etc.

Exhaustive Search is very expensive to compute and needs to use a coarse search grid and specific ratios when using the sliding window. Unlike Exhaustive Search, Selective Search doesn't need to use a fixed ratio, enabling us to capture objects at all scales and ratios. More importantly, it is much faster, which frees up the computational power to use stronger techniques in the second stage of feature extraction and classification.

Selective Search is decomposed in three steps. First, we initialize small starting regions using Efficient Graph-Based Image Segmentation [4]. Second, we recursively combine similar regions, following a bottom-up hierarchical grouping. Finally, we use the segmented region proposals to generate boxes encapsulating each region as shown in figure 4.

Figure 4. Selective Search: The first row shows the segmented regions, the second row shows the corresponding proposed bounding boxes. The number of iterations of selective search increases by each column

The similarity function between two regions r_i and r_j should be fast to compute and should be easily propagated when we merge two regions without returning to the pixels of the image. The method described in [3] uses as a similarity function:

$$s(r_i, r_j) = a_1 s_{colour}(r_i, r_j) + a_2 s_{texture}(r_i, r_j)$$

$$+ a_3 s_{size}(r_i, r_j) + a_4 s_{fill}(r_i, r_j)$$

where s_{colour} describes the colour similarity using normalized histograms based on the intensity, $s_{texture}$ describes the texture similarity also using normalized histograms using fast SIFT-like measurement, s_{size} encourages small regions to merge first preventing a single region to domain and glob all its neighbours and finally s_{fill} measures how well two regions fit together, the more touching the two regions are the bigger this value is.

3.3.2 Feature Extraction

Every proposed region is reshaped to 227x227 regardless of the proposed aspect ratio. We feed each region to CNN composed of 5 convolutional layers and two fully connected layers. Finally, we get a feature vector composed of 4096 values representing a region.

The described architecture was proposed by Krizhevsky et al. and it achieved 58.5% on VOC 2007, is also known as the T-NET. Another architecture was proposed by Simonyan and Zisserman in 2014 known as the O-NET. The O-NET is composed of 13 convolutional layers, 5 max-pooling layers and 3 fully connected layers. The new architecture outperformed the old one achieving 66% on VOC 2007. But as O-NET is much deeper, it is 7 times slower.

3.3.3 Classification

We train an SVM for each class. These SVM take as input the extracted features as mentioned above.

3.3.4 Non-Max Suppression

Objects can be identified more than once and many bounding boxes could be set for the same object. This is one of the most common problems with object detection algorithms.

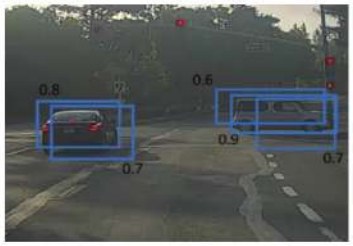

Figure 5. Multiple bounding boxes for the same object. We use Non-Max Suppression to deal with this problem

The Non-Max Suppression technique (NMS) solves this problem. This technique takes the boxes with maximum probability (box with probability 0.9 in the example of Figure 5) and suppressing the close-by boxes with non-max probabilities (0.6 and 0.7 boxes of the same example). The criterion of suppression is Intersection Over Union (IoU):

$$\text{IoU(box1, box2)} = \frac{\text{area of intersection of box 1 and 2}}{\text{area of union of box 1 and 2}}$$

If this criterion is bigger than a certain threshold, we delete the box with the smallest probability.

Next, for every class, we apply greedy non-maximum suppression which eliminates the bounding boxes that have an overlapping coefficient IoU (Intersection-over-Union) with the ground-truth box having the highest score, higher than a certain threshold.

3.3.5 Bounding-Box Regression

To further improve the result of R-CNN, we predict new bounding boxes as an adjustment of the proposed ones. For this task, we train a regressor taking as input the proposed bounding boxes and as output the ground-truth bounding boxes (each box is described by 4 values: the coordinates of the centre, width and height).

4. Faster R-CNN

4.1. Introduction

In 2016 Ren et al. Faster R-CNN [5] improved Fast R-CNN [6] (the second version of the R-CNN algorithm) by integrating the region of proposal step into the CNN model to further reduce the computational cost, and achieves nearly real-time results.

Ren et al. introduced the concept of Region Proposal Network (RPN) to replace the expensive Selective Search.

By using RPN the proposal phase takes only 10ms on test-time instead of 1500ms. RPN acts as an "attention mechanism" telling the network where to look, in other words where the regions of interest are. The role of the RPN is to propose bounding boxes and so it replaces the Selective Search.

Faster R-CNN is cost-efficient and it manages also to improve the accuracy of the previous versions. It have been also adopted for other uses like 3D object detection, instance segmentation and others.

4.2. Related Work

4.2.1 Fast R-CNN

Girshick improved his own R-CNN algorithm in 2015 publishing the Fast R-CNN [6]. He managed to speed-up the computational time of the original R-CNN. Instead of passing each ROI to a CNN, we pass the whole image to a CNN which creates a feature map. Then, Fast R-CNN uses ROI projection on that feature map creating a feature vector for each region. Each feature vector is then passed to fully connected layers to determine the class of each ROI and to refine the bounding box position.

4.2.2 OverFeat and MultiBox

The problem with R-CNN is the use of Selective Search to determine the bounding boxes and it doesn't predict them on its own. So, its performance is very dependant on the results of the Selective Search. Thus the importance of RPN which replaces it. RPN performs better than existing methods that predict the bounding boxes directly.

For instance, OverFeat only works under the assumption of having a single object and MultiBox doesn't share the features between proposal and detection networks and it is not translation invariant, unlike RPN.

4.3. Architecture

In Faster R-CNN, the architecture becomes as follows (Figure 6 below shows the architecture of Faster-RCNN):
- First, we have a fully convolutional neural network shared with the detection network (similarly to Fast R-CNN). The shared network can be composed of 5 layers (Zeiler and Fergus ZF-model) or 13 layers (VGG-16 model).
- Second, the created feature map is passed to the RPN that will regress the region bounds and the objectness probability for each location of the feature map.
- Third, the same feature map along the proposed regions are passed to a detection network named Fast R-CNN (because it is the same network as the one used in Fast R-CNN) that will classify each region.

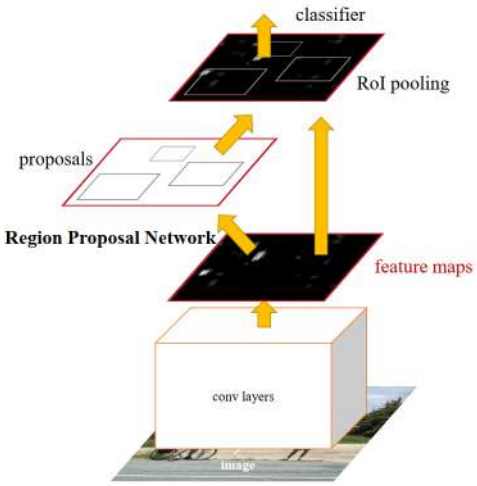

Figure 6. Faster R-CNN architecture

On PASCAL VOC 2007 test set with Fast R-CNN as a detector and using Selective Search (2000 proposal) performs an mAP of 58.7% against 59.9% when we use RPN (with only 300 proposal) and the ZF-model.

4.4. RPN - Region Proposal Network

4.4.1 Architecture

The RPN takes as input the feature map. A mini-network is applied to a sliding window of size nxn (typically $n = 3$) of the feature map. At each window (location), we propose k regions or also known as anchors (typically 9 corresponding to 3 ratios $\{1{:}1, 1{:}2, 2{:}1\}$ and 3 scales $\{128^2, 256^2, 512^2\}$). Using a pyramid of anchors proved to be more efficient to capture objects at different scales than using a pyramid of images or using windows of multiple scales.

We map each anchor to a 256-feature-vector (or 512-feature-vector) . This vector is then fed to two fully connected layers reg and cls.

- reg outputs 4 values for each anchor that present the localization of the box and total of $4k$ values for each location on the feature map. reg is a form of box-regressor.

- cls outputs 2 values for each anchor that estimate the probability of having an object and the background and total of $2k$ values for each location on the feature map. cls is a binary classifier.

So, the output of the RPN is $6k$ values for each location on the feature map. Non-Maximum Suppression (NMS) is applied with a threshold of IoU equal to 0.7 to reduce the redundancy and the proposals' overlapping.

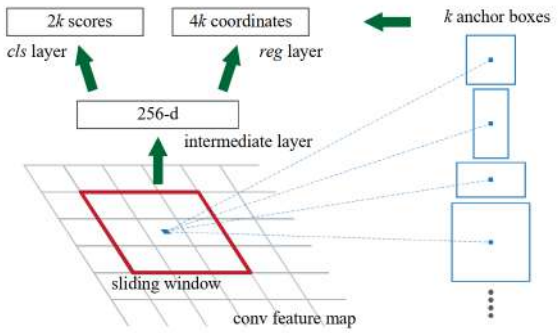

Figure 7. RPN architecture

4.4.2 Loss

RPN is trained using SGD (stochastic gradient descent) and back-propagation. The objective function to minimize is a weighted sum of the log loss (L_{cls}) and smooth L_1 (L_{reg}):

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \frac{\lambda}{N_{reg}}\sum_i L_{reg}(t_i, t_i^*)$$

where p_i is the predicted probability for anchor i, p_i^* equal to 1 if the anchor is positive (either having maximum IoU with a ground bounding-box or an IoU ≥ 0.7) and 0 if the anchor is negative (having IoU ≤ 0.3), t_i and t_i^* are defined in equation (2) of [5] (presenting the difference between the predicted/ground–truth box and the anchors).

4.5. Training RPN and Fast R-CNN

Instead of training the RPN and Fast R-CNN separately. We use *Alternating training*, which is the used solution in the experiments of [6]. First, we train RPN, and use the proposals to train Fast R-CNN. Then the tuned network by Fast R-CNN is used to initialize RPN, but only the RPN layers are tuned and the shared layers are fixed.

5. YOLO

5.1. Introduction

You Only Look Once (YOLO) [7] was proposed by Joseph Redmon et al. in 2016. It re-frames object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using this system, you only look once (YOLO) at an image to predict what objects are present and where they are.

YOLO understands generalized object representation (it can detect correctly objects from artwork data set for instance, even if it's not trained on this type of data).

5.2. Difference between YOLO and R-CNN family

The R-CNN family, that we discussed in previous parts, use regions to localize the objects within the image. The network does not look at the entire image, only at patches of the images.

On the other hand, YOLO deals with object detection in a different way. It takes the entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes. The biggest advantage of using YOLO is that it's incredibly fast and can process 45 frames per second making ot very useful for real-time detection unlike Faster R-CNN for instance which can process only 7 frames per second.

5.3. How does YOLO work

5.3.1 YOLO grids

YOLO first takes an input image, and divides it into grids as in figure 8:

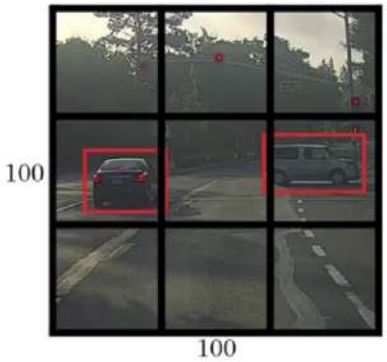

Figure 8. Dividing the input images into grids in YOLO (3×3 grids in this example)

Now that our grid is ready, image classification and localization are applied on element of the grid. YOLO then predicts the bounding box, the corresponding class and the confidence score for each object.

5.3.2 Target vector

The labeled data given to YOLO have the following form:

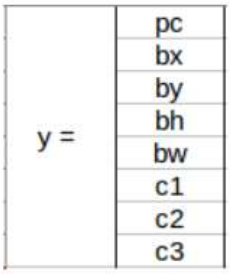

Figure 9. The label vector y (in this example we have 3 classes c1, c2 and c3)

Each label y (Figure 9) is a vector of n_c+5 components, where n_c is the number of classes. The other 5 components are:
- p_c : defines whether an object is present in the grid or not (it is the confidence probability). If there is no object detected, then $p_c = 0$. p_c=1 in the ground truth label.
- b_x and b_y: the coordinates of the object's center. YOLO will take the mid-point of an object and this object will be assigned to the cell grid which contains this mid-point. So up to this point, each cell can detect only one object.
- b_h is the ratio of the height of the bounding box (red box in the above example) to the height of the corresponding grid cell.
- b_w is the ratio of the width of the bounding box to the width of the grid cell.
- $c_i, \forall i \in \{1, .., n_c\} : c_i = 1$ if the object c_i is present in the grid, 0 otherwise. (only one component c_i is non zero since we said that up to now each grid can detect only 1 object)

5.3.3 Anchor Boxes

As we said, each cell can identify only one object. In real life, multiple objects can be present in one cell (i.e the center of multiple objects are localized in the same cell), so here intervene the notion of anchor boxes, as mentioned in [8].

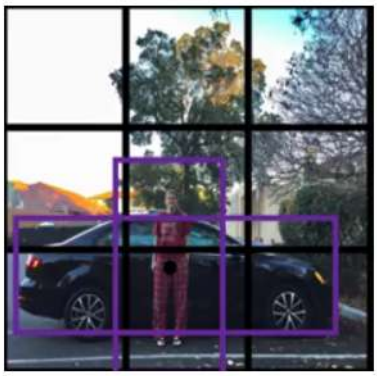

Figure 10. anchor boxes (in purple): The mid-point of the car and the person are in the same grid, so we define 2 anchor boxes in order to detect the 2 objects.

if we use anchor boxes, we might be able to output multiple boxes for multiple classes! To do this, first we pre-define different shapes called anchor boxes or anchor box shapes. Now, for each grid, instead of having one output, we will have multiple outputs. We can always increase the number of anchor boxes as well. So the size of our output vector y will be $n_a \times (5+nc)$, where n_a is the the number of anchor boxes.

5.3.4 Architecture of the network

YOLO's network (Figure 11) has 24 convolutional layers followed by 2 fully connected layers. The first 20 CNN layers are pretrained on the ImageNet 1000-class competition dataset.

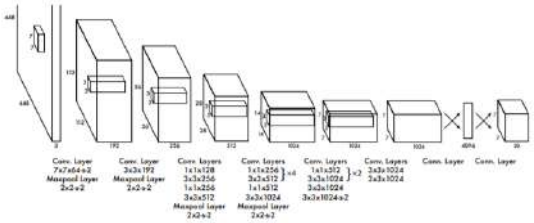

Figure 11. YOLO's Architecture

Figure 12 is a simpler figure of YOLO's network, where we only focus on the first 2 CNN layers and the output layer. In this example we use only one anchor box, 3×3 grids and 3 object classes, so the output size will be of size $3 \times 3 \times 8$. In training phase, both forward and backward propagation are run. During the testing phase, an image is passed to the model, we run forward propagation until we get an output y.

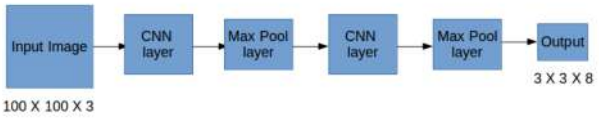

Figure 12. First 2 convolutional layers of YOLO (example of 1 anchor box, 3×3 grids and 3 classes). The layers between the 2^{nd} CNN layer and the output are not visible.

5.3.5 Improvement: Non-Max Suppression

Up to this point, there is a very annoying problem. Objects can be identified more than once. To solve this, as in R-CNN and Faster R-CNN, we use Non-Max suppression as discussed in section 3.3.4.

6. Experiments and Discussion

After analyzing the 3 articles, we decided to compare only Faster-RCNN and YOLO. In fact, RCNN is supposed to be $10 \sim 20$ times slower than Faster RCNN, moreover it is less accurate. So, we will not include RCNN in this section.

For both Faster-RCNN and YOLO, we computed the following metrics: Precision, Recall, IoU and mAP. We calculate the metrics on the whole data, then on each of the top 10 most frequent classes and finally on the remaining classes (excluding the top 10 classes) to see how the algorithms behave for the rare and frequent classes.

	Metrics	All Classes	person	car	chair	book	bottle	cup	dining table	bowl	traffic light	handbag	Rare Classes
0	mAP@.75	0.584638	0.819768	0.702967	0.575226	0.442349	0.686653	0.759367	0.510642	0.690595	0.566506	0.302912	0.530958
1	IoU	0.841847	0.842120	0.836013	0.832594	0.790839	0.830302	0.858254	0.843203	0.869481	0.790855	0.799625	0.846309
2	Recall	0.764832	0.839205	0.703367	0.555340	0.467109	0.679040	0.736410	0.587295	0.658241	0.666731	0.301645	0.757992
3	Precision	0.768789	0.823812	0.731046	0.623891	0.516604	0.713422	0.774354	0.588671	0.683911	0.690901	0.367982	0.774067

Figure 13. Faster-RCNN results on different classes with different metrics

	Metrics	All Classes	person	car	chair	book	bottle	cup	dining table	bowl	traffic light	handbag	other
0	mAP@.75	0.419054	0.411889	0.472554	0.487444	0.086097	0.390911	0.605092	0.411933	0.615297	0.290525	0.348209	0.489645
1	IoU	0.760365	0.736032	0.765973	0.771116	0.566109	0.731324	0.811253	0.792754	0.801106	0.720971	0.736347	0.784501
2	Recall	0.539562	0.699400	0.558495	0.360269	0.230856	0.426979	0.484809	0.330695	0.472404	0.410944	0.134796	0.506720
3	Precision	0.940493	0.966390	0.908243	0.917143	0.942529	0.924005	0.866051	0.852608	0.854484	0.972081	0.786585	0.934905

Figure 14. YOLO results on different classes with different metrics

6.1. Faster-RCNN experiment

We made use of $fasterrcnn\_resnet50\_fpn$ from $torchvision.models.detection$ which is trained on the COCO dataset. We keep the predicted boxes with an accuracy higher than 70% and having and IoU higher than 0.6 with the ground truth (since we have many objects per image, we made a box-matching to know each prediction corresponds to which object of the ground truth).

Quantitative results can be found in figure 13.

6.2. YOLO experiment

We used $YOLO\_v3$, implemented using $Darknet\_19$. YOLO uses Non-Maximal Suppression (NMS) to only keep the best bounding box. The first step in NMS is to remove all the predicted bounding boxes that have a detection probability that is less than a given NMS threshold. In the code we used, we set this NMS threshold to 0.6.

We cloned YOLO's source code, pretrained on COCO dataset, from a github open source repository [9]. We cloned this repository into our personal github account in order to make adequate changes and experiments.

Quantitative results can be found in figure 14.

6.3. Discussion

6.3.1 Execution Time

Device	Faster-RCNN	YOLO
GPU	0.120	**0.020**
CPU	7.410	1.200

Table 1. Execution Time for Faster-RCNN and YOLO in seconds

We notice that YOLO is 6 times faster than Faster-RCNN. And so, this confirms that YOLO (and single-stage algorithms in general) is faster and more appropriate for real-time detection than Faster-RCNN (two-stage algorithms). In fact, Faster-RCNN takes 0.12 seconds on GPU which is equivalent to 8 FPS, we estimate then YOLO to take only ~ 0.02 seconds on GPU which is equivalent to ~ 40 FPS.

6.3.2 IoU

We know that Faster RCNN and YOLO output 2 things: the class of an object and the predicted bounding box.Since precision, recall and mAP are metrics to evaluate how good our detector is in detecting the class of an object, we used IoU metric in order to evaluate the quality of predicted bounding boxes in YOLO and Faster RCNN.

IoU is the area of intersection divided by the area of union between 2 boxes as defined in section 3.3.4.

After matching each prediction to a bounding box from the ground-truth we calculated the IoU between them. The results are shown in figures 13 and 14. We can see that Faster-RCNN has a higher IoU equal to 0.84 compared to YOLO's 0.76 which means that Faster-RCNN's boxes are closer to ground truth boxes than the YOLO's boxes.

6.3.3 Precision & Recall

We recall that:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

Using the experiments' results highlighted in figures 13 and 14, we deduce that YOLO is extremely precise. It's far stronger than Faster RCNN in this criterion as YOLO's precision over all classes is 94.05 % , while Faster-RCNN's precision is only 76,87%, Which means that if YOLO detects an object, this detection is in general correct.

However, Faster-RCNN is stronger than YOLO in the recall criterion as Faster-RCNN's recall over all classes is 76,48%, while YOLO's recall is only 53.96%. This means that Faster RCNN detects more objects than YOLO.

This conclusion can be enhanced if we take a look at the example in figure 15. We can clearly see that Faster-RCNN predicts much more boxes than in the ground-truth, this ensures a high recall but a bad precision. On the opposite, YOLO predicts objects with a high confidence score, thus ensures high precision but at the cost of missing hard objects.

In this example, we have:

	Faster-RCNN	YOLO
Precision	$6/13 = 0.46$	$4/4 = 1.00$
Recall	$6/6 = 1.00$	$4/6 = 0.67$

Table 2. Precision and Recall for figure 15

We saw that YOLO has a better precision and Faster-RCNN has a better recall. We also know that these two metrics depend on the minimal confidence score that we accept. Hence, the need for a trade-off between precision and recall and the need for another metric to combine both of them.

6.3.4 mAP

mAP (Mean Average precision) is a popular metric in measuring the accuracy of object detectors like Faster-RCNN and YOLO. Average precision (AP) computes the average precision value for recall value over 0 to 1. In other words, it's the area under the precision-recall curve. mAP is simply the average of AP, which means that in some contexts we compute the AP for each class and average them. But in some other context, for example under the COCO context, there is no difference between AP and mAP.

Using the experiments' results highlighted in figures 13 and 14 where we used an IoU threshold = 0.75 , we can clearly see that Faster-RCNN is better than YOLO in the mAP criterion. Faster-RCNN has an mAP over all classes equal to 58.46%, while YOLO's mAP is only 41.91%. This can be explained by the weakness of YOLO in the Recall criterion, hence the area under the precision-recall curve in the YOLO case is smaller than Faster-RCNN's case which makes YOLO's mAP smaller than Faster's.

Moreover, figures 16 and 17 explain why YOLO's mAP is small. We can see that YOLO is struggling in detecting small objects like books or traffic lights, while Faster-RCNN is always performing better than YOLO in these cases. Figure 15 also shows that YOLO succeeded in detecting "big" zebras, however, it couldn't detect the "small" ones in the background of the image. This is one of the reasons that make YOLO's recall and mAP small.

Even for Faster-RCNN, the mAP is lower for small objects like handbag (30.29 %) and books (44.23 %). But it is still better than the results of YOLO.

7. Conclusion

After analyzing the three methods and the results of our experiments, we conclude that on one hand, YOLO which is a one-stage detector is much faster and more precise than Faster-RCNN which is a two-stage detector. On the other hand, Faster-RCNN is more powerful and accurate than YOLO in detecting as many objects as possible. We had also presented other strengths and weaknesses of each algorithm.

We also note that even though RCNN is slower and less accurate than Faster-RCNN, studying it was necessary and interesting as the fundamental concept of the RCNN family.

As a conclusion, all these detectors are powerful and their performance is very satisfying in most of the cases. Choosing one algorithm or another depends on the use: If we want a very precise algorithm that almost never predicts wrong classes, or we want a very fast detector, then YOLO is the solution. But if we want a detector that detects most of the objects accurately in a given image, then Faster RCNN is the most suitable algorithm.

References

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, UC Berkeley. *Rich feature hierarchies for accurate object detection and semantic segmentation Tech report*. URL: https://arxiv.org/pdf/1311.2524.pdf.

[2] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, Yann LeCun. *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*. URL: https://arxiv.org/pdf/1312.6229.pdf.

[3] J. Uijlings, K. van de Sande, T. Gevers, A. Smeulders. *Selective Search for Object Recognition*. URL: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

[4] Pedro F. Felzenszwalb, Daniel P. Huttenlocher. *Efficient Graph-Based Image Segmentation*. URL: http://people.cs.uchicago.edu/~pff/papers/seg-ijcv.pdf.

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. URL: https://arxiv.org/pdf/1506.01497.pdf.

[6] Ross Girshick. *Fast R-CNN*. URL: https://arxiv.org/pdf/1504.08083.pdf.

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. URL: https://arxiv.org/pdf/1506.02640.pdf.

[8] PULKIT SHARMA. *A Practical Guide to Object Detection using the Popular YOLO Framework – Part III (with Python codes)*. URL: https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/?utm_source=blog&utm_medium=a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1.

[9] Garima Nishad. *YOLO-Object-Detection*. URL: https://github.com/Garima13a/YOLO-Object-Detection.

Appendix

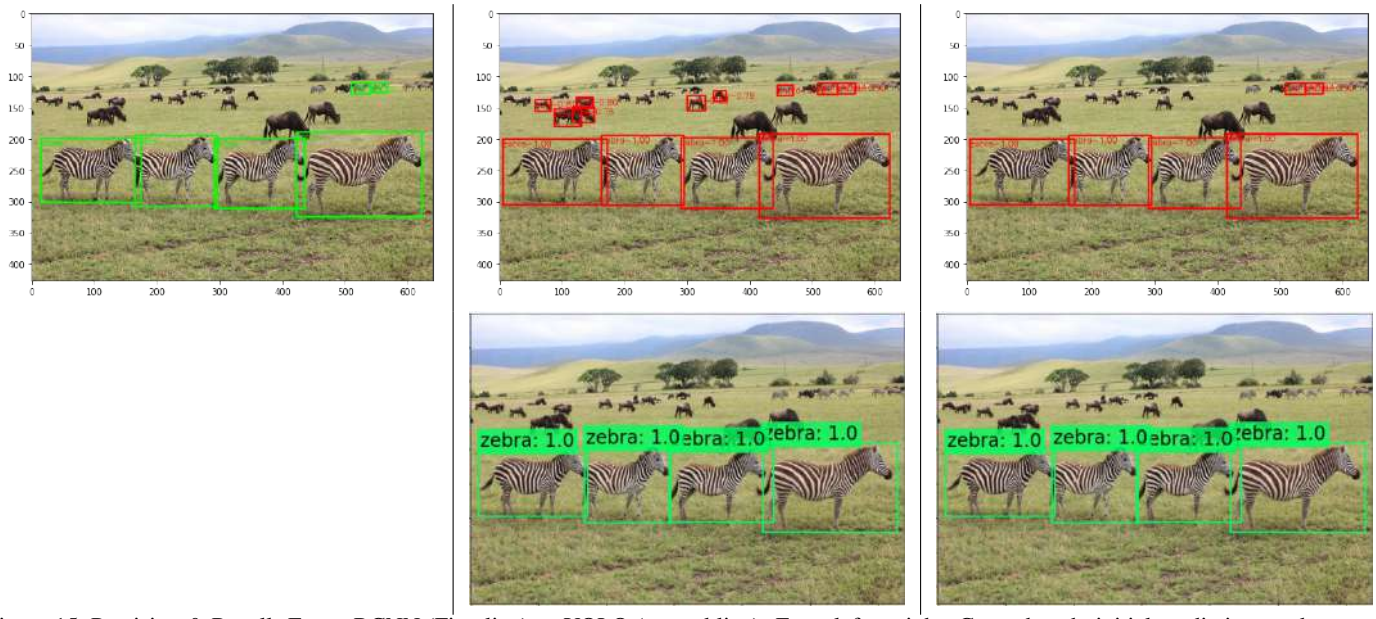

Figure 15. Precision & Recall: Faster-RCNN (First line) vs YOLO (second line). From left to right: Ground truth, initial predictions and correct predictions

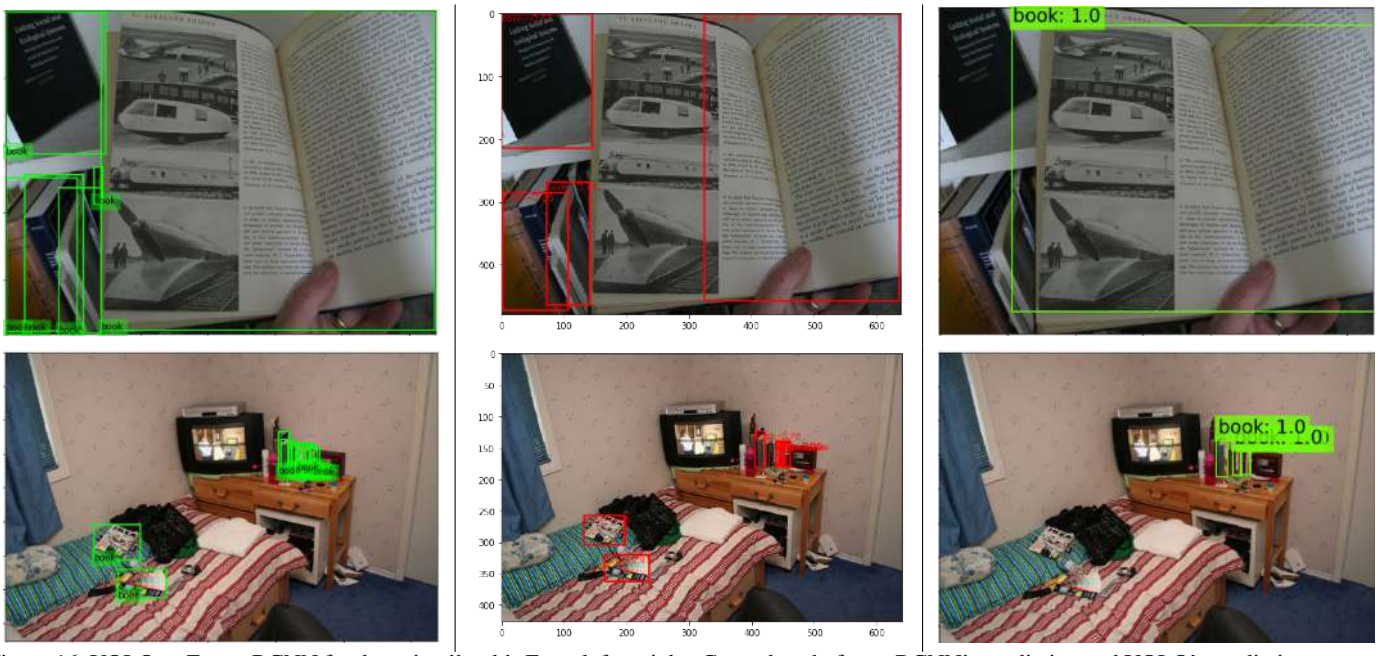

Figure 16. YOLO vs Faster-RCNN for detecting 'book'. From left to right: Ground truth, faster-RCNN's prediction and YOLO's prediction

Figure 17. YOLO vs Faster-RCNN for detecting 'traffic light'. From left to right: Ground truth, faster-RCNN's prediction and YOLO's prediction