

POLITECNICO DI MILANO
School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering
Master of Science in Computer Science Engineering



A metric evaluation framework for object detection

Supervisor: Prof. Piero Fraternali

Co-supervisor: Nahime Torres

**Master Thesis of:
Jesús María Romero Riveros, 894627**

Academic Year 2018-2019

...to my family and beloved ones.

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisor Prof. Piero Fraternali who has guided me through this work and accepted me as one of the team members in the lab.

I want to express my sincere gratitude to Nahime Torres who has helped me and instructed me a lot during these months. I also want to thank Federico Milani who has supported me with the technical tasks in the server and also for the translation of the *sommario* in this book.

I would like to thank to all the members from the laboratory that has made the office working times an enjoyable and pleasurable time.

I would like to thank all my friends near and far. To the old ones that have always been supporting me remotely from my home country Paraguay, and all my friends all around the world that I met just right after I stepped into Italy who have turned this studying journey into an amusing and enjoyable one.

Furthermore, I would like to thank the Italian government which has provided with economical possibilities to study abroad in Europe through the MAECI Scholarship. Italy received me with the open and warm arms that I will always be grateful for.

Last but not least, I would like to thank my family that has always been by my side supporting and encouraging me to become a better person professionally and personally. Likewise, I would like to thanks Laura, you have given to me the strength and support for finishing this master during these two years. I am grateful for your love.

Contents

Sommario	i
Abstract	ii
1 Introduction	1
2 State of the art	8
2.1 Detectors in Object Detection and their metrics	8
2.2 Object detection datasets and evaluation metrics	11
2.2.1 Evaluation metrics	13
2.3 Libraries, training & evaluation frameworks	14
2.4 Evaluation frameworks and similar works	17
3 Evaluation metrics	19
3.1 Base Metrics	19
3.1.1 Intersection Over Union	20
3.1.2 Precision and recall	21
3.1.3 Precision-Recall curve	22
3.1.4 Average precision & mAP	23
3.1.4.1 Interpolating 11-points method	24
3.1.4.2 Interpolating all points method	24
3.1.5 ROC curve	26
3.2 Other performance metrics	27
3.2.1 Analysis of False Positives	27
3.2.2 Analysis of True Positives	29

3.2.3	Normalized precision measure	30
4	The tool	32
4.1	Framework analysis architecture	32
4.2	DME design	34
4.3	DME implementation	35
4.3.1	Programming language	35
4.3.2	Metrics implemented	36
4.3.2.1	Base metrics analysis	36
4.3.2.2	True Positive metrics implementation	38
4.3.2.3	False Positive analysis	46
4.4	Brief Tutorial	50
5	Testing & Results	51
5.1	Testing settings	51
5.1.1	Hardware settings	52
5.2	Training & validation	52
5.2.1	Setting 1: Detectron2	53
5.2.2	Setting 2: MMDetection	53
5.2.3	Setting 3: iSAID dataset	53
5.3	Test results	54
5.3.1	Setting 1 & Setting 2: True Positives Analysis	54
5.3.1.1	Aspect ratio	54
5.3.1.2	Truncation evaluation	54
5.3.1.3	Occlusion Evaluation	56
5.3.1.4	Bounding-box area evaluation	56
5.3.1.5	Sensitive and Impact	56
5.3.2	Setting 2: False Positives testing results	61
5.3.3	True Positive and False Positive analysis discussion	61
5.3.4	Setting 3: Testing	62
5.3.4.1	iSAID testing dataset preparation	63
5.3.4.2	Setting 3: Testing results	63
6	Conclusion and future works	66

List of Figures

1.1	Computer Vision analogy to human vision.	2
1.2	Definitions of visual recognition tasks in Computer Vision. . .	5
2.1	Timeline of the most recognized algorithm in Object Detection.	12
2.2	Top open-source libraries for deep learning implementation . .	15
3.1	Precision-Recall Curve example.	23
3.2	Average-Precision total interpolation example.	25
3.3	ROC Curve example.	27
3.4	Confusion with similar object examples.	28
3.5	Examples of occlusion levels in airplanes	29
3.6	Examples of truncation	30
4.1	General architecture of the DME framework analysis	33
4.2	Class diagram from Detector Metrics Evaluator	34
4.3	Precision-recall example	37
4.4	Occlusion graph example	40
4.5	Truncation graph example	41
4.6	Bounding box area graph example	42
4.7	Aspect ratio analysis graph example	43
4.8	Parts-visible analysis graph example	44
4.9	Viewpoint analysis graph example	45
4.10	Sensitive and Impact analysis graph example	46
4.11	True Positive analysis per class graph example	47
4.12	False positive impacts analysis per class graph example	48
4.13	False positive trending analysis per class graph example	49

5.1	Setting 1 & 2: Aspect ratio evaluation	55
5.2	Setting 1 & 2: Truncation evaluation	57
5.3	Setting 1 & 2: Occlusion evaluation	58
5.4	Setting 1 & 2: Bounding-box area evaluation	59
5.5	Scenario 3: Sensitive and Impact evaluation	60
5.6	False Positive impact evaluation of two PASCAL VOC classes with Setting 2 detector.	62
5.7	Precision-recall evaluation from <i>Setting 3</i> detector over iSAID classes	64
5.8	Setting 3: Per class True Positive analysis over the best and worst classes performed in the testing results.	65

List of Tables

2.1	Summary of common evaluation metrics for object detection in general image datasets. Source: modified from [41]	14
3.1	Table of confusion (or confusion matrix) [59].	20
5.1	Training settings summary.	51

List of Code Snippets

4.1	DME snippet sample.	50
-----	-----------------------------	----

Sommario

Il rilevamento di oggetti è uno dei maggiori compiti di visione artificiale su cui è stata fatta una approfondita ricerca negli ultimi decenni. La svolta, in questo campo, è arrivata dopo l'introduzione di reti neurali convolutive profonde (DCNN) nel flusso di lavoro dei riconoscitori, e le loro prestazioni sono in continuo miglioramento da allora. Al giorno d'oggi, gli algoritmi per il riconoscimento di oggetti sono applicati a svariate applicazioni della vita reali, dalle auto con guida autonoma a rischiose procedure mediche. È evidente che questi algoritmi di apprendimento profondo sono molto sensibili al fallimento visti i compiti rischiosi che svolgono. In questo lavoro proponiamo una libreria per la valutazione del riconoscimento di oggetti chiamata *Detector Metrics Evaluator* (DME). DME permette di valutare i risultati di questi algoritmi attraverso alcune metriche tipiche e altre più specifiche. Tale analisi potrà aiutare i ricercatori ad ottenere una valutazione profonda del flusso di lavoro del loro riconoscitore, così da incrementare al massimo accuratezza e precisione. L'implementazione di DME prende in ingresso un set di dati di Ground Truth (nel formato Pascal VOC o COCO) e il gruppo di proposte predette dal riconoscitore. Inoltre, durante l'implementazione in Python di DME abbiamo applicato un design generico per lasciare aperta la possibilità agli sviluppatori di adattare lo strumento; aggiungendo nuove metriche o includendo un altro formato per i set di dati. Abbiamo provato il nostro strumento con i dati provenienti da Pascal VOC 2007 e iSAID insieme ai framework di apprendimento Detectron2 e MMDetection, mostrando, con buoni risultati, la sua natura disaccoppiata da uno specifico set di dati o strumento di sviluppo del riconoscitore.

Abstract

Object detection is one of the major tasks that has been researched in depth in the last decades in Computer Vision. The breakthrough into the field arrived after the introduction of Deep Convolutional Neural Network (DCNN) into detectors pipelines, and its performance improvement has gone on the upswing since then. Nowadays, various real-life applications are applying object recognition algorithms from self-driving cars to even healthcare hazardous procedures. It is noticeable that these Deep Learning algorithms are highly sensitive to failure because of the natural hazardous tasks that they perform. In this work we propose a framework library for the evaluation of Object Detection denominated *Detector Metrics Evaluator* (DME). DME allows to evaluate object detection results through typical and other more specific metrics. Such analysis would allow researchers an in-depth assessment of their object detector pipelines so as to increase their accuracy and precision at maximum. The implementation of DME takes as inputs a Ground Truth dataset (in PASCAL VOC or COCO format) and the set of the detector predicted proposals. Moreover, the DME python implementation applied a generic design to let open the option to developers to make the tool to scale; by adding new metrics or implementing another dataset format. We have tested our tool with Pascal VOC 2007 and iSAID datasets along with Detectron2 and MMDetection training frameworks, showing its decouple nature to the specific dataset or framework for the detector implementation with good results.

Chapter 1

Introduction

Computer Vision (CV) is a branch of artificial intelligence that enables computers to see and identify images, processing them as humans would. Using images or videos from a sensing devices (e.g. a camera), deep learning models enable machines to accurately identify and classify the objects.

In CV, the main objective is to take the input (image or video) and to understand and infer something about the image and its contents [1]. In Figure 1.1 we can see an scheme of a simple comparison between *Human Vision* (HV) and *Computer Vision* (CV). While in HV a person interprets through his or her brain what their eyes captures, in CV, a sensing device plays the role of the eyes capturing the image from the outside world and then a computer interprets the information through algorithms. In this case the inferred sample is just bowl with a bunch of delicious fruits [1].

Nowadays CV applications in real-life along with technology are vast. Thanks to CV along with machine learning, the improvement in various fields are truly impressive.

- *Automobile industry for self-driving cars*: now the well-know electric cars company producer *Tesla* ² is recognized from their autopilot feature set in their units. Thanks to cameras and sensors installed in their units and by applying CV algorithms they can provide to drivers the comfortable feature of autopilot. Another company named *Waymo*

²<https://www.tesla.com/>

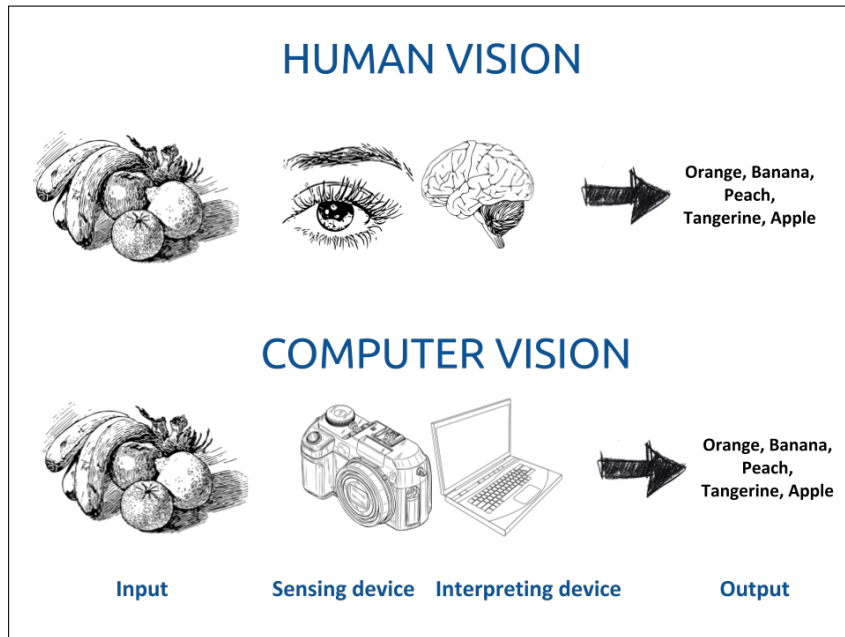


Figure 1.1: Computer Vision analogy to human vision.

¹, a subsidiary from Google, utilizes cameras and sensors into cars in order to make driving experience for drivers safer. It is thought that 20% of road's accident are caused by fatigue drivers [2]. Thus, through hardware and CV they are reducing this statistic figures.

- *Healthcare*: CV is being used to help diagnose health conditions. *Gauss Surgical* has developed a solution that can monitor blood loss in real-time through the use of cloud-based computer vision algorithms. They are using this cellphone application to maximise blood transfusions in real-life scenarios such as Cesarean deliveries [3]. Another study made with Google's AI tools published recently [4] has shown skills of detection of breast cancer which are similar if not better than those of a trained doctor. They have considerably reduced false negatives and false positives doctors diagnostics.

¹<https://waymo.com/journey/>

- *Retail Industry: Amazon Go* ¹ has just released the first smart-store in 2018 in the US. By applying CV, deep learning along with hardware technology (cameras and sensor) in the store they were able to create a just-walk-out store. By tracking a customer as they walk around the store, CV systems keep track of every customer and records the items a customer takes. Then, when they have completed their shop, the customer simply leaves the store without interacting with a cashier for the payment.
- *In-store inventory: Walmart* ² is expanding the use of *Bossa Nova Robotics* shelf-scanning robots to 350 of their stores. These robots are able to identify products with missing labels as well as items that are out of stock or incorrectly priced. CV enables this process to take place while also helping the robots to safely navigate the stores without bumping into customers.
- *Warehouse management: Gather AI* ³ is a Pittsburgh based company that is developing autonomous drones to conduct warehouse inventory management. These operate autonomously and are able to integrate with existing warehouse management systems and devices such as motion sensors. CV systems then scan the recorded images, allowing for an accurate record of available inventory to be created. According to the company's CEO, around 60% cheaper than traditional management methods [5].
- *Industries & factories:* are also using CV for predictive maintenance of their equipment and gears. Many oil and gas producers such as *Shell* are taking advantage of ML and sensors to keep their their equipment and gears in prime condition [6, 7].
- Security & video surveillance is another well-know application area in which CV and ML combination have been applied to reduce thieves

¹<https://www.amazon.com/>

²<https://www.walmart.com/>

³<https://gather.ai/>

in-house and public places. It has been strongly researched through pedestrians detector works like [8, 9, 10, 11] and showed impressive results.

There are even more fields of applications that cannot be completely mentioned in this work, but one important fact that can be inferred is that there exist no room for errors and mistakes in these tasks. Thus, researchers must keep the robustness of their algorithms in a high level to be able to perform such risky and delicate tasks; reducing false positives and false negatives as much as possible and improving accuracy in its maximum.

Diving into the technical CV terminologies, object detection is a main visual recognition problem [12] and there has been a lot of research on it in the last decades in order to arrive in such previous described applications. Object detection's main goal is to find objects of a certain target class (or category) with precise localization in a given image and specify to each object instance a corresponding class label, and generally along with a confidence value.

It is important to recall some definitions in *Computer Vision* to avoid confusion. *Image classification* aims to assign an image one or multiple labels corresponding to the presence of a category in the image; it does not localize the object position within images. *Semantic segmentation* aims to predict pixel-wise classifiers to assign a specific category label to each pixel, thus providing a finer understanding of an object in a image. Last but not least, *instance segmentation* identifies different objects and set each of them a separate categorical pixel-level mask being this the most detailed object identification in images. This last can be seen as an special case of *object detection* where instead of wrapping the object with a bounding box, a pixel-level classification is proposed. In Figure 1.2 the definitions are depicted in samples images for each case. In this work we will mostly focus on *object detection*.

A lot of effort has been developed during last two decades researching detectors for object recognition in CV applying deep learning [13]. Authors in the field usually test their algorithms in open datasets. Mostly, researchers

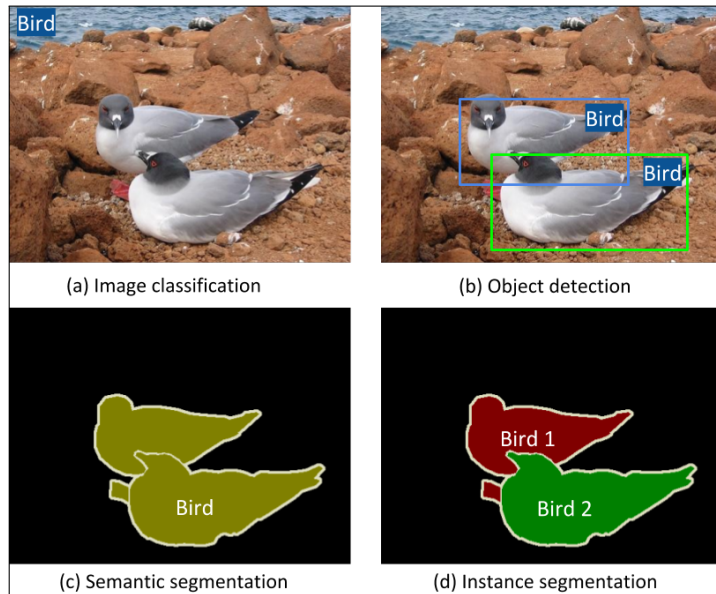


Figure 1.2: Definitions of visual recognition tasks in Computer Vision.

(a) *Image classification* only need to identify if a category of object exist in an image. (b) *Object detection* identify + localize each instance with bounding boxes. (c) *Semantic segmentation* makes a pixel-wise classification in the image for the categories. (d) *Semantic segmentation* is similar to (c) but also discriminates each object instance. Source: Pascal VOC 2007 [13]

have conducted their detectors evaluation applying base well-known metrics in the field such as *recognition accuracy*[14] and *average precision* [13]. These metrics provides an overall evaluation and gives a broad score of the detector’s accuracy over a dataset. Nonetheless, these metrics are not enough to asses the performance and identify the strength and weaknesses for specific behaviours of their detectors. Moreover, such performance summaries do not specify why one method outperforms another or help understand how it could be improved [15]. Sometimes researchers are interested in comprehending in depth how their detectors works addressing specific characteristics of objects. They may be interested in answering the following questions:

- What is the performance of my detector over small-size objects?
- Is my detector able to localize partially-occluded and truncated ob-

jects?

- Can I tune-up my detector to reduce false positive cases like confusion with the background or confusion with other objects, and how will this impact in the overall mean-average-precision (mAP) value?

Considering the scenario in the last question, a strong improvement in one characteristic of recognition may generate only a small improvement in the overall performance, thus a clarification of such case before hand is needed. For this there have been various metrics evaluation and other metrics proposals for object recognition [15]. Authors in [15] considers the evaluation of object size, aspect ratio, point-of-view and occlusion.

Therefore, we find the need to centralize diverse metrics for object detection to better evaluate results when assessing ground-truth and predictions. The contribution of this thesis can be summarized as follows:

- We study and summarize the state of the art of metrics and tools available for object detection evaluation in Computer Vision.
- Given a dataset ground-truth (GT) and its predicted proposals, we developed an framework library tool, named *Detector Metrics Evaluator* (DME), that centralize the evaluation of typical and other specific metrics in object detection. The user can run a group (or all) metrics from a list of available ones inferred according to the input dataset.
- To prove independence of training frameworks and specific datasets, the tool has been tested over the results of *Detectron2* [16] and *MMDetection* [17] training frameworks over Pascal VOC [13] and iSAID [18] datasets.

The structure of this document is detailed as follows. In Chapter 2 the state-of-the-art is presented. We detail a summary of metrics in most-recognized works in object detection. Moreover, we give a briefly description of most used training and evaluation frameworks in the field. In Chapter 3 we provide the theoretical background of the most used metrics, including the

implemented ones. Then, in Chapter 4 the tool architecture and implementations are detailed. We explain all implemented metrics in the framework. Then, in Chapter 5 an evaluation of the tool was made, and the results are presented. Last but not least, in Chapter 6 we wrote the conclusions and proposed future works.

Chapter 2

State of the art

In Section 2.1 we review the most notorious research in Object Detection, from the first attempts without Deep Learning (DL) methods till nowadays ones highlighting their evaluation metrics. In Section 2.2 we give a brief description of the most common datasets and metrics evaluation in Object Detection. Then, in Section 2.3 we describe the most used libraries and training frameworks. Finally, in Section 2.4 we describe current evaluation frameworks available in the literature, and how our tool is differentiated from them.

2.1 Detectors in Object Detection and their metrics

Before the Deep Learning (DL) era, first development of algorithms for object detection was based in a three stage pipeline:

1. proposal generation.
2. feature vector extraction.
3. region classification.

In the first stage the core goal was to search locations by applying intuitive sliding windows methods [19, 20, 21, 22, 23]. Moreover, input images were

resized into different scales and multi-scale windows were also applied in order to capture information about aspect ratios and the scale nature of objects. The idea was to scan the whole image and localize regions that might contain objects (called *regions-of-interest*, ROI). In [20] they evaluated their face-detectors applying *Receiver Operator Characteristic (ROC) curve*. The *ROC curve* shows the trade-off between sensitivity (or True-Positive-Rate, TPR) and specificity ($1 - \text{False Positive Rate}$, FPR). Detectors that give curves closer to the top-left corner in the graph indicate a better performance. A more detailed explanation on this would be boarded in the Chapter 3.

In the second step, on each part of the image, a fixed-length feature vector was obtained from the sliding window. The aim of this was to capture semantic information from the region covered. This feature vector was usually encoded by low-level visual descriptors such as: SIFT (Scale Invariant Feature Transform) [24], HOG (Histogram of Gradients) [22], Haar [25], or SURF (Speeded Up Robust Features) [26], which showed a considerable robustness to scaling, illumination and rotation variances. Again papers [24, 25, 26] centered their evaluation performance through the *ROC curve*.

Last but not least, in the third step the region classifiers were learned to assign labels to the regions covered. *Support Vector Machine* (SVM) [27] were commonly utilized because of their good performance in small scale training data. Others classification techniques like adaboost [28], bagging [29] and cascade learning [23] were employed in the region classification stage providing an improvement in accuracy. In [28, 29, 23] works metrics such as error vs pseudo loss curve, typical error-rate and ROC-curve were utilized to compare their results with other works respectively. In [28] authors defined a sophisticated error measure called *pseudo loss* that focus the loss analysis over specific miss-rate that are harder to discriminate.

Various datasets were born in the cradle of image challenges competition starting from 2007. This type of competitions provided a huge number of annotated images which filled the previous vacuum of lacking datasets for evaluation in the community. Moreover, thanks to these challenges, strong detectors algorithms with considerable performances in would be born in the following years. Besides the images that they provided, also evaluation

metrics were created in order to assess the challenges winners. Among them, one dataset that gained big popularity in the field is Pascal VOC [13] (See section 2.2 for more datasets in Object Detection). [13] is an open dataset utilized for benchmarking and training object detectors. After its creation [13], most of the papers that applied or benchmarked their detector on the dataset, began to assess their work with mean-average-precision (mAP). This metric was introduced in the competition in 2007 ¹ as the core evaluation metric.

Most of the traditional methods for object detection that achieved a good performance applied feature descriptors to obtain embedding for a region of interest. When good feature representations were applied with a robust region classifier, remarkable results [30] were achieved over Pascal VOC dataset [13]. For instance, deformable part based machines (DPM) method algorithms obtained the first place in the Pascal VOC competition from 2007-2009. Getting more in details, DPM algorithms learn and integrate multiple part models with a deformable loss, and mine hard negatives examples with a latent SVM for training. Even though in 2008-2012 period in the Pascal VOC challenge, competitors applied DPM approaches but gained minor improvement in every year that passed, thus leaving bare DPMs limitations. Some of the limitations seen in the DPM methods can be summarized in the following:

- a high numbers of proposals were generated in the first phase, even redundant ones which led to a high number of False Positive during classification stage. Furthermore, authors set the window scales heuristically, and has constrains detecting various specific sizes of objects.
- capturing semantic relevant information in complex context was difficult. Researchers hand-crafted the feature detectors just using low level visual cues [31, 32].
- the optimization along the pipeline was made separately. Hence not achieving a whole optimal solution.

¹Pascal VOC challenge: <http://host.robots.ox.ac.uk/pascal/VOC/>

So far we have detailed more traditional algorithms that did not apply handcrafted feature representation. However, in 2012 there was a transition from these method to deep learning [33] approaches, which generated a huge margin improvement in detectors.

Deep Convolutional Neural Network (DCNN) is a biologically inspired structure for computing hierarchical features [34]. By applying DCNN in image classification, works such as [35, 36, 37] achieved great performances. AlexNet architecture [35] have become one of the most influential papers in the area after achieving a nearly 50% reduction in the error-rate in the *ImageNet Challenge* [38] dataset, which was unprecedented progress at the time. Then, for object detection task, the pipelines that became the breakthrough were *R-CNN* [39] and *Faster R-CNN* [40].

In Figure 2.1 a timeline of the discussed methods is provided in which most recognized works, datasets and most-used metrics are ordered from works that applied traditional handcrafted features till the most representative deep learning works. It is important to mention that the metrics in the timeline are set in the year in which the first appearance in a Object Detection work happened and not the year in which they were invented.

2.2 Object detection datasets and evaluation metrics

In this section we provide a description of the most common benchmarks for general object detection, and the evaluation metrics in these. Most of these datasets were gather with goals of providing an annual challenge for CV tasks (object detection, object sementation, instance segmentation). Thus, metrics evaluations are also provided along with the datasets. We did not add datasets for specific detection cases such as face detection or pedestrian detection.

- *Pascal VOC 2007* [42] is a mid scale dataset for object detection with 20 object categories. As in most benchmarking datasets, it has been

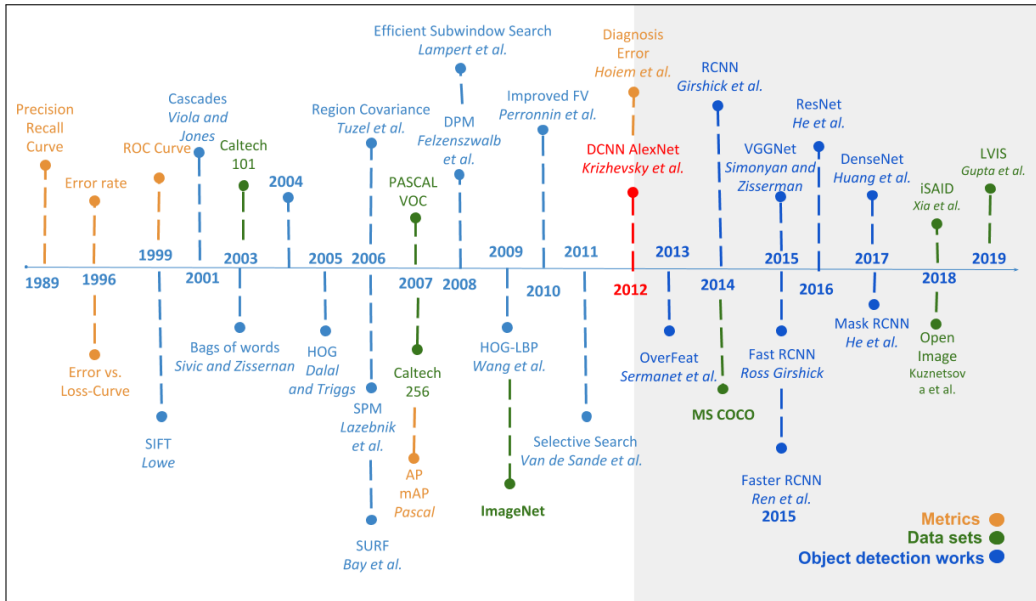


Figure 2.1: Timeline of the most recognized works (blue), datasets (green) & evaluation metrics (orange). In 2012 there was a change from handcrafted feature representation to Deep Convolutional Neural Network by the introduction DCNNs for image classification by Krizhevsky et al. [35] with the AlexNet pipeline. Source: adapted from [41]

separated in three groups: training, validation and test with 2,501, 2,510 and 5,011 images, respectively. In total there are 9,963 images containing 24,640 annotated objects.

- *Pascal VOC 2012* [42] is a mid scale dataset for object detection and segmentation that has the same 20 categories of the previous Pascal VOC2007, but with more images added. It has been splitted in training, validation and test with 5,717, 5,823 and 10,991 images, respectively. However, the annotation information of the test dataset is not available. The train/val data has together 11,530 images containing 27,450 ROI annotated objects and 6,929 segmentations.
- *MSCOCO* [43] is a large scale dataset for objects detection, panoptic segmentation and image captioning. There are various versions of

datasets depending of the year. The last and bigger dataset version dates from 2018, splitted in training, validation and test, it counts with 118,287, 5000 and 40,670 images respectively. MSCOCO18 has a total of 860,001 + 6,781 annotations for training + validation sets. The test set is not annotated though.

- *Open Images* [44] contains 1.9M images with 15M objects in 600 categories. In object detection purposes, only 500 most frequent categories are used to evaluate detection benchmarks, and more than 70% of these categories have over 1000 training samples [45].
- *LVIS* [46] is a new collected benchmark with 164,000 images and 1000+ categories based on MSCOCO 2017 [43] images with the original train-validation-test split.
- *ImageNet* [47] is an important dataset with 200 categories in 516,840 high scale images.
- *iSAID* [18] is a large-scale dataset for object detection and instance segmentation. The images are originally from aerial images from the DOTA dataset [48], and the annotation is based in the MSCOCO format [43]. iSAID counts with 665,451 object instances for 15 categories across 2,806 high-resolution images.

2.2.1 Evaluation metrics

In Table 2.1 we show a summary of the metrics used for evaluation in these datasets. One of the most common metrics after the PascalVOC dataset was *mean Average Precision* (mAP). For VOC 2007 [42], VOC 2012 [42] and ImageNet [47], the metric *Intersection-over-Union* (IoU) threshold of mAP is set to 0.5, and for MSCOCO [43] dataset, a variation of the mAP metric is employed. For the latter [43] there is a variation of IoU threshold 0.5, 0.75 in mAP, and also MSCOCO distinguishes among the AP for object sizes depending on the size (small, medium and large based on their pixel’s area).

Sym.	Meaning	Description	
IoU	Intersection-over-union	The IoU threshold to evaluate localization	
D	All predictions	Top predictions returned by the detectors with highest confidence score.	
TP	True Positive	Correct predictions from proposals.	
FP	False Positive	False prediction from proposals.	
P	Precision	The ratio of TP out of total proposals.	
R	Recall	The ratio of TP out of all positive samples.	
AP	average precision	Obtain by computing e different levels of recall.	
mAP	mean average precision	Average value of AP across all clases.	
TPR	True positive rate	The ratio of positive rate over positive samples.	
Object detection challenges metrics			
mAP	mean average precision	VOC 2007 & VOC 2012	mAP at 0.5 IoU threshold over all 20 classes
		OpenImages	mAP at 0.5 IoU threshold over 500 most frequent classes
		MSCOCO & iSAID	AP _{COCO} : mAP avg over ten IoU values AP ₅₀ : mAP at 0.50 IoU threshold AP ₇₅ : mAP at 0.75 IoU threshold AP _S : AP _{COCO} for small object of area smaller than 32 ² AP _M : AP _{COCO} for object of area between 32 ³ and 96 ² AP _L : AP _{COCO} for large objects of area bigger than 96 ²

Table 2.1: Summary of common evaluation metrics for object detection in general image datasets. Source: modified from [41]

Just to remark, *inference speed* is usually used to evaluate detectors as well. Nonetheless, we just considered detection accuracy metrics in this work because of the nature of our framework analyzer.

2.3 Libraries, training & evaluation frameworks

Nowadays, in order to train a detector, researchers have two main options, implement themselves the detector pipeline for a CV task, or just make use of already developed and optimized training framework.

In the first hand, the development of a detector’s pipeline is independent of the programming language or library. There is not a single criterion for determining the best development for deep learning. Each tool was designed and built to address the needs perceived by the developers and also reflects their skills and approaches to problems [50]. However, there have been certain

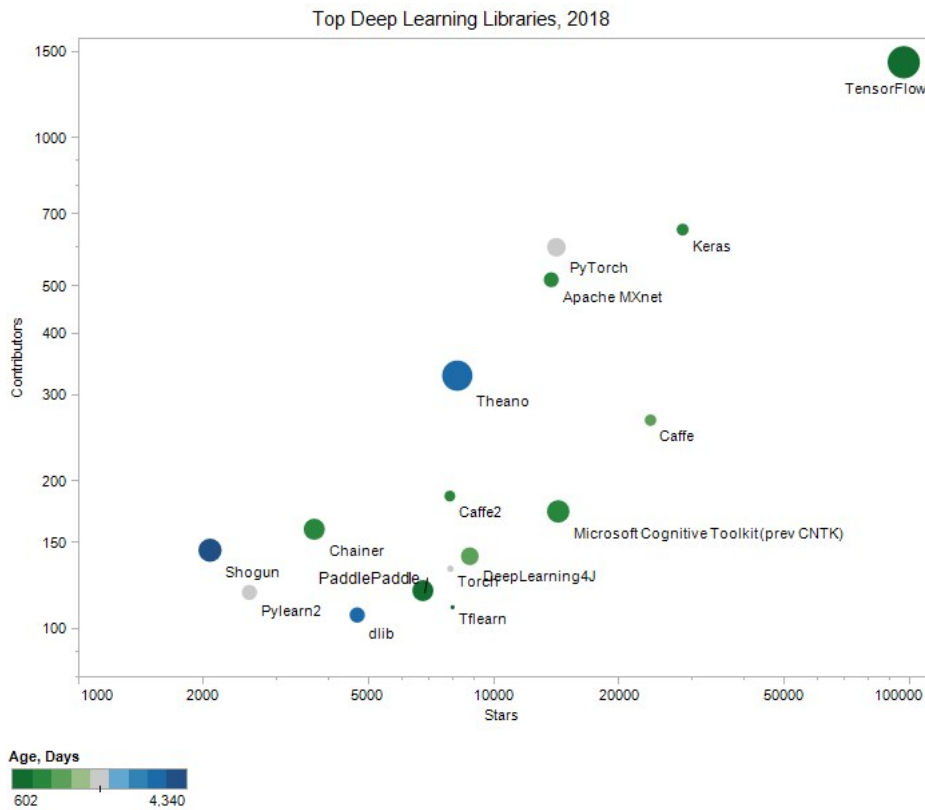


Figure 2.2: Top open-source libraries for deep learning implementation. Top open source deep learning libraries by Github stars and contributors, using log scale for both axes. The color of the circle shows the age in days (greener - younger, bluer - older), computed from Start date given on github under Insights / Contributors. Source: [49]

preference of the community for the interpreted Python¹ language for the implementation of pipelines of general use [49]. In Figure 2.2 we can see a top analysis of the most used open-source libraries used in the community in GitHub² open-code repository. Along the top three most-used libraries that

¹Python <https://www.python.org/>

²Github: <https://github.com/>

has been written in Python, we can cite: TensorFlow¹, PyTorch² and Keras³.

On the other hand, a lot of effort has been incurred in the last years for implementing generic training frameworks by the community. These frameworks have been implemented by researcher in open and collaborative repositories, and they made extensive use of the previous libraries. Most of the detectors mentioned in the previous Section 2.1 are implemented and it is possible to train pipelines for, besides our main focus object detection, different CV tasks (image classification, and both image & instance segmentation). The main advantages of these framework are that they have been tried for a lot of contributors and are free of biases in their implementation and some undesired bugs. Although some of these training framework are still green in development and not all feature options or parameters have been developed, skilled researchers can put hand an small changes to the tools can bring impressive results, instead of reinventing the wheel. Among the most used training frameworks we can cite: MMDetection [17], Detectron2 [16], maskrcnn-benchmark [51], simpledet [52] and matterport/Mask_RCNN [53].

An usual feature in these training frameworks is that, besides generating the training statistics and bounding boxes proposals, they provide some basic statistics and broad evaluations metrics for users to understand their training results. However, these values sometimes are not enough for a deep evaluation and more sophisticated evaluation need to be done; it is here were our proposed framework enter in the game (See Chapter 4 for the tool details). Some metrics provided by these training frameworks, such as mAP, IoU, precision, recall are already summarized in Table 2.1 since most of the evaluation techniques sourced were implemented from datasets toolkits.

In this work we have made extensive use of Detectron2 [16] and MMDetection [16]. The author of this thesis have made various changes to MMDetection in order to add features that later was applied to train models. The changes made are out of the scope of this book.

As this work's title claims, we propose an evaluation framework for Object

¹TensorFlow: <https://www.tensorflow.org/>

²Pytorch: <https://pytorch.org/>

³ Keras: <https://keras.io/>

Detection. The main difference between *training framework* and *evaluation framework* is that the former mainly focus on training detectors and saving their statistics as output, and the later focus in taking the training statistics and generating insights from these; the output from *training frameworks* can be used as input for *evaluation frameworks*.

Moreover, since skilled-coding researchers can rely on their own implementation of the pipelines, the statistic training inputs should be generated through the libraries mentioned above.

2.4 Evaluation frameworks and similar works

There have been various works that have developed evaluation frameworks in the field [54, 55, 15, 56], in this subsection we describe them and provide the differences with our proposed tool with its features.

First of all, Mariano et al. [54] describe a tool named *ViPER*(Video Performance Evaluation Resource), it is developed in JAVA¹ and it has and open-source code available online². Authors implemented seven metrics variation based on base metrics, such as precision, recall, but adapted to the video context, such that the metrics take into account the evolution of the video frames for the computation of these metrics. This work falls out of the scope of object detection in images.

Secondly, Wolf et al. [55] proposed a 3D performance graphs based on precision, recall and harmonic mean metrics. The *harmonic mean* is lineal combination of the precision and recall which allows to emphasize the minimum of of both values (check [55] for the formula). In order to generate a fair comparison among detectors, a scalar single value is generated from the 3D graph, thus the score is used for comparison purposes. They have tested their metrics over text recognition dataset ICDAR 2003 [57]. Even though authors in that specific work declares that the metrics can be used for any kind of object in object recognition with rectangles bounding-boxes, their

¹JAVA <https://java.com/>

²ViPER source code: <https://sourceforge.net/projects/viper-toolkit/files/viper-toolkit/>

metrics have been applied only to text recognition works and not general object detection.

Thirdly, Hoiem et al. [15] implemented a framework and proposed powerful new metrics for object detector analysis. The code is open-source written in MATLAB¹. They follow a fixed structure modular and the code specs all the inputs for the datasets that needs extra annotations. Besides their proposed metrics, basic metrics are not provided. Our framework implement all the metrics in [15] plus the base metrics with Object-Oriented-Programming (OOP) paradigm in Python².

Last but not least, the most similar work to ours has recently been published at the beginning of 2020. Authors in [56] developed a generic framework for evaluation in Python². They found the need to centralize the evaluation metrics for object detection tasks in order to standardize implementation for researchers. However, they centered their efforts in gathering only metrics from four main challenges, specifically: Pascal VOC Challenge[58], GoogleImageV4 [44], COCO Detection Challenge [43], ImageNet Object Localization Challenge [47]. The metrics implemented falls in the category of base metrics in our framework, and have been cited in Table 2.1. In our proposed framework, besides implementing the typical base metrics 2.1, we provided all the metrics implemented for True Positive and False Positive categories analysis [15]. Also, because some of the metrics implemented need extra annotations, a tool was added in order to annotate these to image datasets.

¹MATLAB <https://www.mathworks.com/>

² Python <http://www.python.org>

Chapter 3

Evaluation metrics

In this chapter we present the theoretical background and definitions for the most utilized metrics in CV for Object Detection. In Section 3.1 we review the typical object detection's metrics definition and formulas. Later, in Section 3.2, more in-depth performance metrics are detailed.

3.1 Base Metrics

We summarize under the set of *Base Metrics* the most well-known and widely user metrics for Object Detection. Before going deeper into the detail, it is important to refresh some concepts (confusion metrics, true positives, false positives, among others). In Machine Learning and in problems of statistical classification is quite common to find a *confusion matrix* (also called *error matrix*) [59] describing predicted solution. This matrix shows the basic performance of an algorithm: each row of the matrix represents the instances in a predicted class, while the columns corresponds to the instances in an actual class [60]. The *confusion matrix* name stems from the fact that it depicts if the algorithm is mislabeling a class. In predictive analytics, a table of confusion (sometimes also called a *confusion matrix*), is a table with two rows and two columns that reports the number of false positives, false negatives, true positives, and true negatives.

In Table 3.1 we can find the typical confusion matrix which contains the

		Actual class	
		P	N
Pred. class	P	TP	FP
	N	FN	TN

Table 3.1: Table of confusion (or confusion matrix) [59].

following categories:

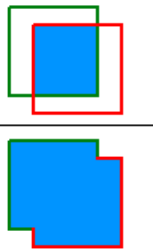
- *True Positive (TP)*: equivalent to a hit.
- *False Positive (FP)*: equivalent to a false alarm. A negative case that was classified as positive.
- *False Negative (FN)*: equivalent to a miss. A positive case that was miss classified.
- *True Negative (TN)*: equivalent to a correct rejection.

3.1.1 Intersection Over Union

Intersection Over Union (IoU) is a measure based on Jaccard Index [61] that evaluates the overlap between two bounding boxes. It requires a ground truth bounding-box B_{gt} and a predicted bounding box B_b . By applying the *IoU* we can tell if a detection (or proposal) is valid (True Positive) or invalid (False Positive). *IoU* is given by the overlapping area between the predicted bounding box and the ground truth bounding box divided by the area of union between them:

$$IoU = \frac{B_b \cap B_{gt}}{B_b \cup B_{gt}} \quad (3.1)$$

Analogous to the formula in Equation 3.1, we illustrate the *IoU* between a ground truth bounding box (in green) and a detected bounding box (in red). Notice the area of union of both bounding boxes. Hence, an IoU of 1 implies that the predicted and the ground truth bounding boxes perfectly overlap.

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{img1}}{\text{img2}}$$


When describing the *IoU*, it is usually defined along with what is called threshold. The threshold is just a restriction value that defines what is the minimum value of *IoU* that we consider to be a correct detection or True Positive (TP). Depending on the metric, it is usually set to 50%, 60%, 75% or 95%. We define below how the values in a confusion matrix are defined in taking into account the *IoU*.

- *True Positive (TP)*: A correct detection. Detection with $IOU \geq \text{threshold}$.
- *False Positive (FP)*: A wrong detection. Detection with $IOU < \text{threshold}$
- *False Negative (FN)*: A ground truth not detected. For instance, when a ground truth is present in the image and model failed to detect the object.
- *True Negative (TN)*: Does not apply. It would represent a correct misdetection. In the object detection task there are many possible bounding boxes that should not be detected within an image such as the background. Thus, TN would be all possible bounding boxes that were correctly not detected (so many possible boxes within an image).

3.1.2 Precision and recall

Precision and *recall* are metrics to evaluate the detection performance in detectors. *Precision* is defined as the ability of a model to identify only

the relevant objects. It is the percentage of correct positive predictions in a certain class and is given by:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{All_detections} \quad (3.2)$$

Recall (also known as *Sensitivity* or *True-Positive-Rate*) is defined as the ability of a model to find all the relevant cases (all ground truth bounding boxes). It is the percentage of true positives detected among all relevant ground truths in a certain class and is formulated as:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{All_ground_truths} \quad (3.3)$$

3.1.3 Precision-Recall curve

The *precision-recall curve* is a way to evaluate performance of an object detector as the confidence value is changed in intervals by plotting a curve for each object class [62].

An object detector of a particular class is considered good if its precision stays high as the recall increases, which means that if you vary the confidence threshold, the precision and recall will still be high. Another way to identify a good object detector is to look for a detector that can identify only relevant objects (0 FP = high precision), finding all ground truth objects (0 FN = high recall).

A low performance detector needs to increase the number of detected objects (increasing FP = lower precision) in order to retrieve all ground-truth objects (high recall). For this reason, the *precision-recall curve* usually starts with high precision values, decreasing as recall increases. A perfect performance in a *Precision-recall curve* would be a curve that reach the right-top-corner; the closer the curve to the top-right corner, the better.

In Figure 3.1 an example of this metrics is depicted along three different classes. This figure was generated by our DME framework after evaluating a detector result. In this case, the *airplane* class (blue lines) performs better

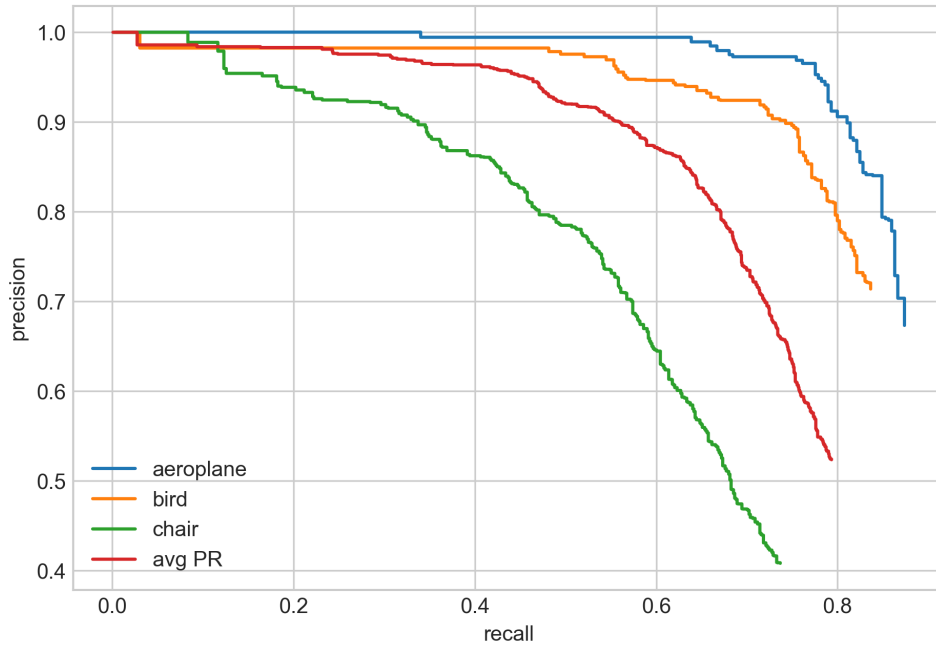


Figure 3.1: Precision Recall Curve example. The *airplane* class (blue lines) performs better than the other classes as the confidence value decreases. On the other hand, the *chair* (green lines) class perform worse. Source: DME framework.

than the other classes in the graph as the confidence value decreases. On the other hand, the *chair* class performs worst among them. Notice that the lines in read is an average among the three classes. This kind of curve is utilized within the PASCAL VOC 2012 challenge [58] as metric evaluator.

3.1.4 Average precision & mAP

Another form for comparing the performance of object detectors is to calculate the area under the curve (AUC) of the *Precision- Recall* curve. As AP curves are often zigzag curves going up and down, comparing different curves (different detectors) in the same plot usually is not a trivial task because the curves tend to cross each other much more frequently. Thus, a numerical

metric such as *Average Precision* (AP) is a better option to compare the overall performance different detectors. In practice AP, is the precision averaged across all recall values in the range [0-1].

From the AP the mean-Average-Precision (mAP) is obtained. The well-known metric mAP corresponds to the mean of the AVs along all classes analyzed by a single detector in a single running.

There are two ways of calculating the AP. The first method is done by calculating 11-points of interpolation, and the second method is computed by calculating the interpolation of all points in the precision-recall curve. The last method is the one that has been used in the Pascal VOC Challenge since 2010 [58].

3.1.4.1 Interpolating 11-points method

The 11-point interpolation tries to summarize the shape of the precision-recall curve by averaging the precision at a set of eleven equally spaced recall levels [0, 0.1, 0.2, 0.3, ... , 1] [13].

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} p_{interpol}(r) \quad (3.4)$$

with

$$p_{interpol}(r) = \max_{\tilde{r}: \tilde{r} \geq r} p(\tilde{r}) \quad (3.5)$$

where $p(\tilde{r})$ is the measured precision at recall \tilde{r} . The AP is obtained by interpolating the precision only at the 11 levels taking the maximum precision whose recall value is greater than .

3.1.4.2 Interpolating all points method

The second method consisting in all points defined in the following equation:

$$\sum_{r=0}^1 (r_{n+1} - r_n) p_{interpol}(r) \quad (3.6)$$

with

$$p_{interpol}(r) = \max_{\tilde{r}: \tilde{r} \geq r_{n+1}} p(\tilde{r}) \quad (3.7)$$

where $p(\tilde{r})$ is the measured precision at recall \tilde{r} [13].

In this case, instead of using the precision observed at only few points, the AP is now obtained by interpolating the precision at each level, taking the maximum precision whose recall value is greater or equal than its current recall value. Hence, the area under the curve is obtained. In the Figure 3.2 a visual example on how the total interpolation looks like over a Precision-Recall graph. This last option is the one implemented in the DME framework.

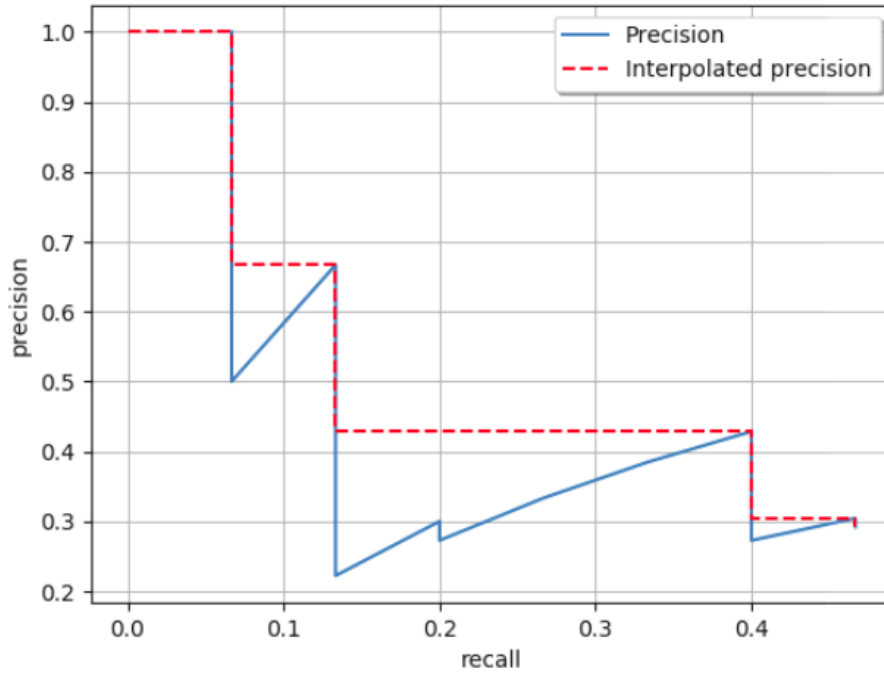


Figure 3.2: Average-Precision total interpolation example. From this graph, the Area-under-the-curve is obtain to finally computer the AP.

3.1.5 ROC curve

As it was mentioned in Chapter 2, a useful tool when predicting the probability of a binary outcome is the *Receiver Operating Characteristic* curve or *ROC* curve.

Basically, the ROC curve is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different threshold values in range [0.0 - 1.0]. In other words, it plots the false alarm rate versus the hit rate.

The *True Positive Rate* (TPR) is an alternative name for the *recall* or *sensitivity*. As defined in Equation 3.3, it is calculated as the number of *true positives* divided by the sum of the number of *true positives* and the number of *false negatives*. It describes how good the model is at predicting the positive class when the actual outcome is positive.

The *False Positive Rate* (FPR) can be defined as:

$$FPR = \frac{FP}{FP + TN} \quad (3.8)$$

It is also called the *false alarm rate* as it summarizes how often a positive class is predicted when the actual outcome is negative.

The *ROC curve* importance relies on the fact curves of different models can be compared directly in general, or even for different thresholds. Moreover, the *area under the curve* (AUC) can be used as a summary of the model performance.

When we predict a binary outcome, it is either a correct prediction (TP) or not (FP). There is a tension between these options, the same occurs with TN and FN. In general, good models are represented by curves that bow up to the top left corner of the plot. The closer the lines to the top-corner, the better. See Figure 3.3 for a simple example.

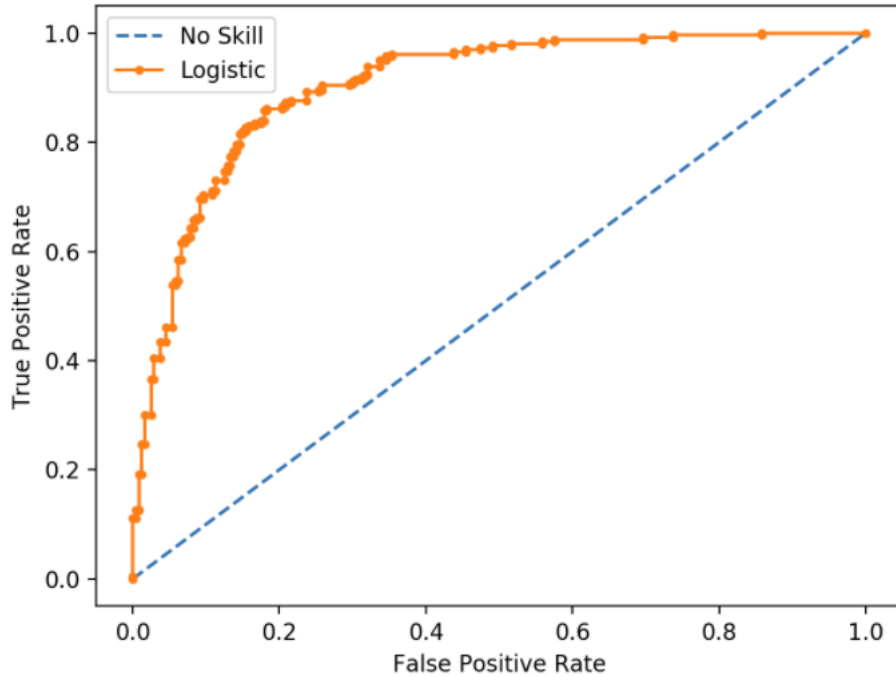


Figure 3.3: ROC curve graphs the TPR vs FPR. Lines for a poor Classifier (blue dotted lines) and a Logistic Regression Model classifier (orange lines). Source: modified from [63]

3.2 Other performance metrics

In this section we introduce more specific metrics that allows in-depth assessment in object detection. Based on the metrics in [15], we have implemented the evaluation platform that take as a base project the metrics in [15]. The work in [15] analyses metrics for *False Positive (FP)* and *True Positives (TP)* cases.

3.2.1 Analysis of False Positives

False positives (FP) are one of the major types of errors found when analysing computer vision detectors. The work in [15] addressed the category of FP

depending on the value of the IoU (overlap) with the ground truth object. They types of FPs are classified as follows:

1. *Localization error* an object from the target category is detected with a misaligned bounding box ($0.1 \leq IoU < 0.5$). Duplicate detections (two or more detections for the same object) are also included in this category.
2. *Confusion with similar objects*, when the IoU is at least 0.1 and it is confused with an object in the same category. In the Figure 3.4 examples of this scenario is seen. All the detections were confused with objects from similar category to *cow* from the PASCAL VOC 2007 categories. Images source: Pascal VOC 2007 [13].
3. *Confusion with dissimilar categorized objects* when the overlapping ratio is at least 0.1 and the object corresponds to a different category.
4. *Confusion with background or other not labeled object*.

For items 2 and 3, a previous grouping of similar object categories is necessary. For example, in Pascal VOC dataset [13] the following semantic categories were created: all vehicles, animals including persons, furniture chairs, dining tables, sofas, flying objects aeroplanes, birds. Examples of these metrics analysis can be found later in in Chapter 4, in which the metrics implementation are explained.



Figure 3.4: Confusion with similar object examples. In these pictures all the FP detections were confused with objects from similar category to cow from the PASCAL VOC Category.

3.2.2 Analysis of True Positives

Just to recall, *false negatives* (FN) are ground truth objects that are detected with a low confidence or completely missed by the detector. On the other hand, *true positives* (TP) are objects proposals that were correctly detected. According to [15], we can analyze certain intuitive characteristics in images that may cause a detector not to detect objects correctly or miss them completely. Thanks to these metrics, it is possible to analyze both TP and FN at the same time. These characteristics in the images could be:

1. *Occlusion*: when a part of an object is obscured by another surface. Possible classifications are as follows: N (none), L (low), M (medium), H (high). In the Figure 3.5 the difference among these levels can be seen.
2. *Truncation*: when a part of an object is outside the image. Options: N (not truncated), T (truncated). Check Figure 3.6 for picture examples.
3. *Bounding-box area*: Object size is measured as the pixel area of the bounding box. It is formulated as the product of width-height. According to their size, they can be classified into: XS (extra small), S (small), M (medium), L (large), XL (extra-large).
4. *Aspect ratio*: opposite to the previous metrics, the *Aspect ratio* corresponds to the width-height ratio. Possible classifications are as follows: XT (extra-tall/narrow), T (tall), M (medium), W (wide), XW (extra wide).

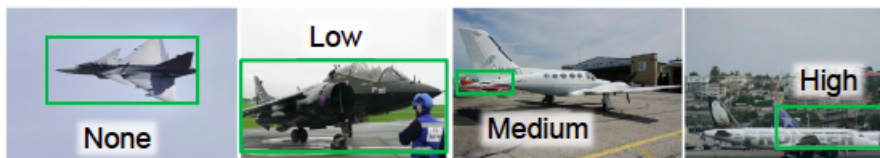


Figure 3.5: Examples of occlusion levels in airplanes. Levels: N (none), L (low), M (medium), H (high). Source: Pascal VOC2007 [13] dataset

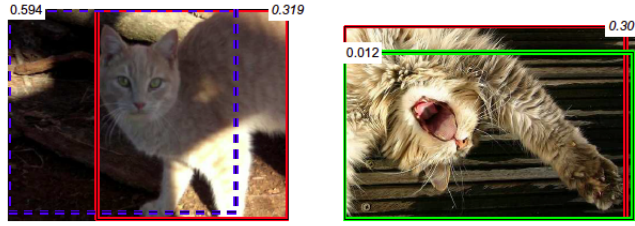


Figure 3.6: Examples of two truncated cats in two images. Truncation occurs because the object is stays out of the boundaries in an image. Source: Pascal VOC2007 [13] dataset

5. *Parts visible*: it defines whether a certain part an object is visible. This depend specifically on the physical characteristics of the object. For instance, possible options for a cat class can be: body, ear, face, leg, tail. In Figure 3.6 both cat faces and ears are visible, but not their tails and either their four legs.
6. *Viewpoint*: tells the point-of-view of object position in the image. Possible options are: bottom, front, rear, side. Still we can add more point-of-view according to our needs.

For metrics such as *occlusion*, *truncation* and *part-visibility* some extra annotations in the metadata dataset are needed. Usually such information is not provided in the regular images dataset. Thus, researchers in [15] had to extend the dataset adding the options cited above, except for the truncation metric which is already provided in the Pascal VOC [13]. This task was done by the same person in order to avoid subjectivity and potential biases.

3.2.3 Normalized precision measure

In subsection 3.1.4 and subsection 3.1.5 we defined the typical definition of AP and ROC curve, respectively. When we want to analyze sensitivity of objects characteristics this metrics are not enough. AP is sensitive to the number of positives samples, and ROC curves are tough to summarize. In

[15] they proposed a way to normalize precision so that a comparison along with objects characteristics can be done fairly.

Based on the Equation 3.3 of recall and the Equation 3.2 of precision, it is true that:

$$P(c) = \frac{R(c).N_j}{R(c).N_j + FP} \quad (3.9)$$

where:

$P(c)$ = precision, fraction of correct detections

$R(c)$ = recall, fraction of objects detected

N_j = number of objects in the class or all Ground Truths

$F(c)$ = number of incorrect detections or False-Positives

Thus, according to [15], in order to apply a comparison among objects with various number and because of class imbalance [64], N_j should be normalized to a general value N . Hence, the *normalized precision* P_N can be defined as:

$$P_N(c) = \frac{R(c).N}{R(c).N + FP} \quad (3.10)$$

where N_j corresponds to the previous normalized N_j .

For PASCAL VOC 2007 [13], [15] proposed a value of $N = 0.15$ which is close to the average of N_j over all object in the 20 categories in the dataset. Moreover, by just computing a simple average of the *normalized precision* P_N we can obtain the average normalized precision AP_N . This normalization is crucial for the computation and class comparison along TP and FP metrics later explained in Chapter 5.

Chapter 4

The tool

In this chapter we present the framework tool named as *Detector Metrics Evaluator* (DME). In Section 4.1 we describe the architecture of DME. Then, in Section 4.2 the tool design is outlined, and in the followed Section 4.3, implementation details are explained. In Section 4.4 we provide a short example on how to make use of the framework.

4.1 Framework analysis architecture

The proposed DME framework tool can be seen as a box that takes inputs data, make analysis from these and then generates graphs and reports. In Figure 4.1 an outline of the tool can be observed.

Firstly, two types *inputs* must be provided to the tool in order to work: *ground truth* and *detection proposals*. The *ground truth* corresponds to the actual dataset annotated with bounding-boxes. In Section 2.2 we cited a number of open datasets available in the literature. Then, the second input to provide are the *detection proposals* or object proposals that the trained model generates as output when applied for inference on a set of images. This data can come from one of two sources:

- Researches own-implementation of pipelines that like to code from zero. The most common approach is to utilize python libraries, such as Tensorflow or Pytorch.

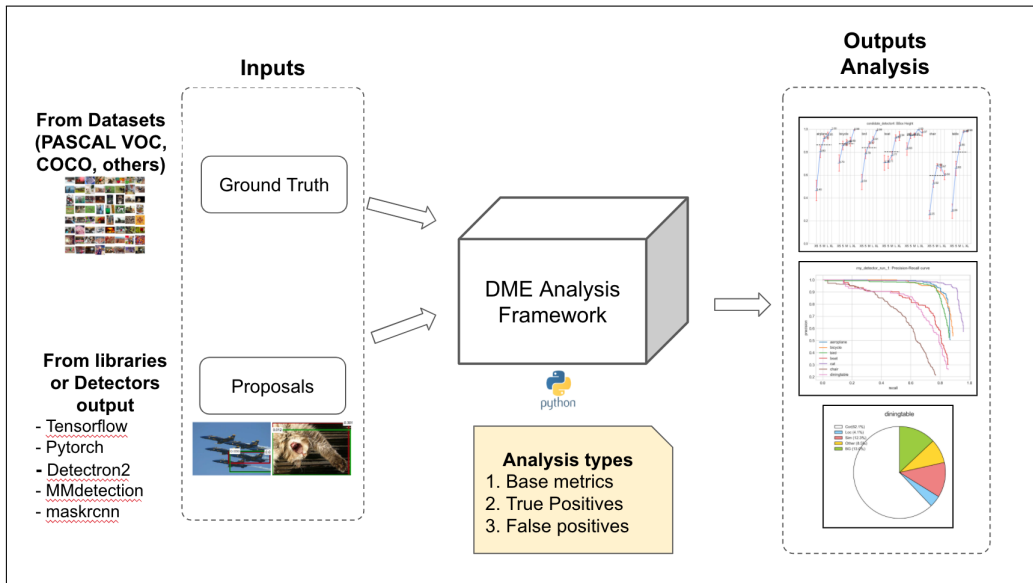


Figure 4.1: General architecture of the DME framework analysis. It can be seen as a box that receives input and generates outputs. The inputs: Ground Truth (from datasets), and Proposals (generated by detectors implementations). As outputs it generates the analysis as graph or reports.

- Already implemented training frameworks, such as *MMDetection* [17] or *Detectron2* [13] (See section 2.3 for libraries and training frameworks details). Probably this is the most preferred method for nowadays researchers.

Secondly, in the center of Figure 4.1 we have the DME Analysis Framework which was designed to be scalable and usable. It is implemented in Python language. The types of Analysis that can be executed are divided into three groups: *base metrics*, *true Positives* and *false Positives*. More details would be given in the following subsections.

Finally, the DME *outputs* correspond to the graphs and reports generated thanks to graphic libraries that are used along with other Python libraries.

In the following section we explain the DME design, and later, the implemented metrics are described in detail.

4.2 DME design

The DME tool was designed applying basic Object Oriented Programming (OOP) paradigm. Because of the relative simplicity of the code, two main classes were generated: Analyzer & Dataset.

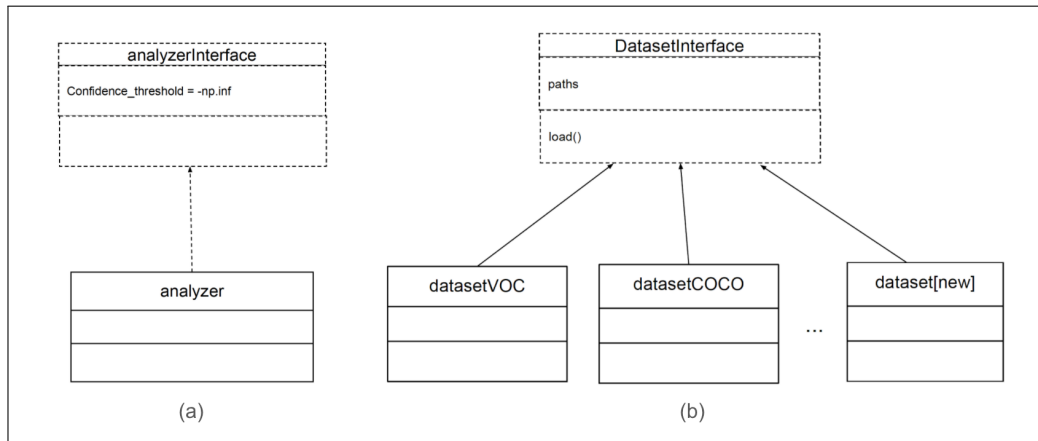


Figure 4.2: Class diagram from DME. There are two basic classes: Analyzer(a) and Dataset(b). The later is implemented for each type of dataset format.

In Figure 4.2 a class diagram from the DME design is depicted. First of all, the class *Analyzer*, which implements the *AnalyzerInterface*, is in charge of the whole analysis and graphics drawing tasks using the data from the *Dataset* class. The Analyzer receives an instance of the *Dataset* class when an instance is created. *Analyzer* implements every metric through their private and public methods, and calls the public methods from *Dataset* instance. The class was implemented taking as the core dataset PASCAL VOC [13]. We decided this because the metrics from the groups *True Positive* and *False Positive* based in the implementation from [15] were already implemented with this format [13]. Moreover, PASCAL VOC 2007 [13] counts with one more attribute than MSCOCO [43]; the *truncation* attribute. This is convenient because one more analysis is possible to execute in this way (See subsection 4.3.2 for more details about the metrics implemented).

Secondly, the class *Dataset* inherits from the interface with the same name. Even though only one class inherits from the interface, we considered appropriate to add it to the design since we wanted to let researchers the possibility to add others implementations of analyzer with others formats and options. *Dataset* goal is to save the path parameters from both, ground-truth from bounding boxes, so it can load the data into memory and save the data to a dictionary in a specific format so that *Analyzer* can make use of it. As can be seen in Figure 4.2.(b), an implementation for every dataset format is needed. Currently there are two formats implementations of dataset available: *DatasetVOC* and *DatasetCOCO* which implements PASCAL VOC [13] and MSCOCO [43] formats, respectively.

4.3 DME implementation

In this section we state the implementation details from the language used to the main analysis methods and parameters.

4.3.1 Programming language

Based in the design explained, DME was implemented in Python¹ language. We have chosen Python as the language for implementation because of the following reasons:

- Python is an interpreted language and it has been one of the most preferred by the Deep Learning research community [49]. Thus, integration with other modules from other Deep Learning tools would be an added advantage.
- The language flexibility allows to apply *Object Oriented Programming (OOP)* principles.
- Python code readability can help to ease researchers tasks. When applying coding standards such as PEP8 ² when coding, it increases its

¹ Python <http://www.python.org>

² Python PEP8 <https://www.python.org/dev/peps/pep-0008/>

readability and contribute to its usability.

4.3.2 Metrics implemented

In this subsection we detail the metrics that have been implemented within DME framework. These have been classified into three groups as follows: *base metrics analysis*, *true positive analysis* and *false positive analysis*.

We implemented part of the DME platform based in metrics evaluation in [15]. The work in [15] analyses metrics for *True Positive (TP)* and *False Positive (FP)* cases which have been included in the groups with the same name.

4.3.2.1 Base metrics analysis

In this group we added all the base metrics that have been widely applied in the field of object detection such as precision and accuracy recognition (see Table 2.1). The base metrics are as follows:

Intersection-over-Union

As it was described in Subsection 3.1.1, *Intersection-over-Union* (IoU) defines intersection between the Ground-truth object and the Bounding-Box proposal, divided by the union of the same two. Even though no graphic was generated for this metric, the computation of it is done when other method are called and use as a base this metric. IoU implementation was based the formulas in Equation 3.1.

Average-Precision

The AP is a metric that helps to evaluate the overall performance of a detector (see Subsection 3.1.4). In our proposed framework the *AP* and *mean-AP* are both calculated. In order to apply this calculation, *precision* and *recall* are calculated first based on Equation 3.2 and Equation 3.3, respectively. Later, the computation of the AP and mAP are both based in the formulas of Equation 3.6 and Equation 3.7 taking into account the last implementa-

tion of the PASCAL VOC 2012 Challenge [58]. The result is then saved for the analysis for the following metrics.

Precision-recall curve

The precision-recall curve allows to evaluate a detector performance by plotting the precision (y-axis) and recall (x-axis) lines while the value of the confidence is decreased; the closer the line to the top-right, the better (see Subsection 3.1.3).

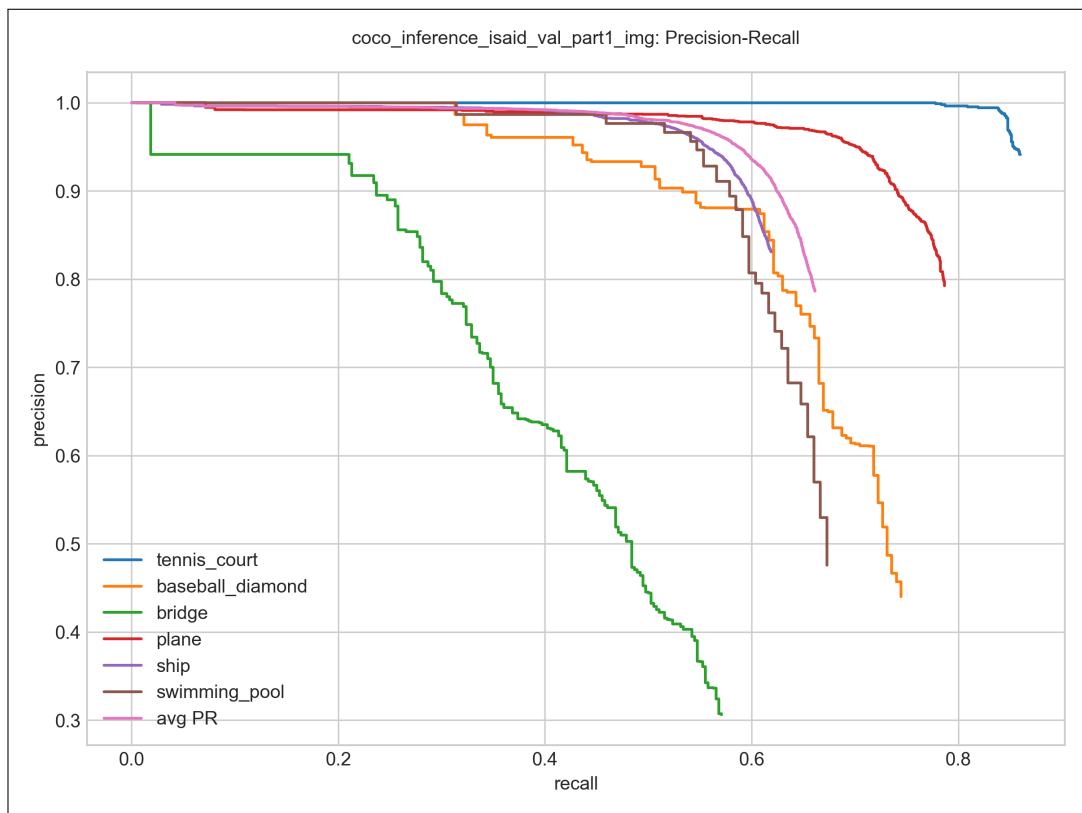


Figure 4.3: Precision-recall curve output after executing method `analyse_precision_recall_per_class()`.

There are two methods that can be invoked to obtain the Precision-Recall analysis:

- `analyse_precision_recall()`: returns the overall precision-recall curve

by analysing all the classes in the dataset, and these classes are taken as one. It allows to evaluate the overall performance of a detector.

- *analyse_precision_recall_per_class(category_selected=, images_grouped=)*: generates the precision recall per class. It might need two not mandatory parameters:

category_selected: an array of classes indicating the classes to draw the curves. If it is not specified, it takes the first six classes in the dataset ordered by name.

images_grouped: flag to generate the P-R curves in just a single image, or separated images. By default, it is set to *True*.

After calling *analyse_precision_recall_per_class(category_selected=..., images_grouped=...)*, a graph such as in Figure 4.3 is generated.

4.3.2.2 True Positive metrics implementation

In subsection 3.2.2 we explained all the types of TP analysis that can be done based on the work [15] along with their parameters and their graph output. A brief description of the graph is discussed to depict how can the metric be used. As it was already mentioned before, the AP_n (AP normalized) version is applied here, and all metrics in the TP analysis group make use of it (see subsection 3.2.3 to recall the AP_N formula).

Before getting into the methods details, we introduce here their common parameters. Since all methods invoke the respective analysis and generate its corresponding graph, each of them receives the same optional parameters. These parameters are:

- *category_selected*: an array of string with the classes names. If it is not set, the first six classes in the list are taken.
- *graph_localization_level*: it defines the localization criteria to consider: 'strong' or 'weak'. '*Strong*', the default value, considers the duplicate detection as localization errors, otherwise with the '*weak*' options, it does not.

The following TPs metrics have been implemented:

Occlusion

The occlusion method checks how has been the performance for a class in occluded and non-occluded objects. To generate the analysis, just call the method `analyze_occlusion()`, and the graph as in Figure 4.4 would be generated. In the graph we can check the AP_n value in the Y-axis, and in the X-axis, the possible options of the occlusion according to the class (see subsection 3.2.2 for the occlusion options). The blue lines link the performance depending on the options AP_n score. The dotted black horizontal lines indicates the average AP_n per class. Last but not least, the red vertical lines specify the standard error. As an example, this detector had problems identifying *High-Occluded* airplanes, instead it had no problem with identifying *None-occluded* airplanes. Furthermore, along the seven classes, the *bicycle* category showed more robustness against occlusions. It achieved only 0.69 AP_n for the lowest performance in the occlusion options.

Truncation

The truncation method checks how has been the performance for a class with truncated and non-truncated objects. To generate the analysis, just call the method `analyze_truncation()`, and a graph as in Figure 4.5 would be generated. Analogous to the line-colors in the previous analysis, in this graph example we can check that most classes were better identified when there was no-truncation in their images. Among them, the cat class was no susceptible to truncation. Moreover, the class table that worked remarkably even better with truncation. The *chair* class was the most susceptible to truncation and also the worst performing one.

Bounding-box area

The Bounding-box area method analyzes the performance level of a detector depending on the object pixels area per class. Call the method `analyze_bbox_area()`, and the graph such as in Figure 4.6 would be generated. The line colors are the same than in the previous analysis. The sizes ranges

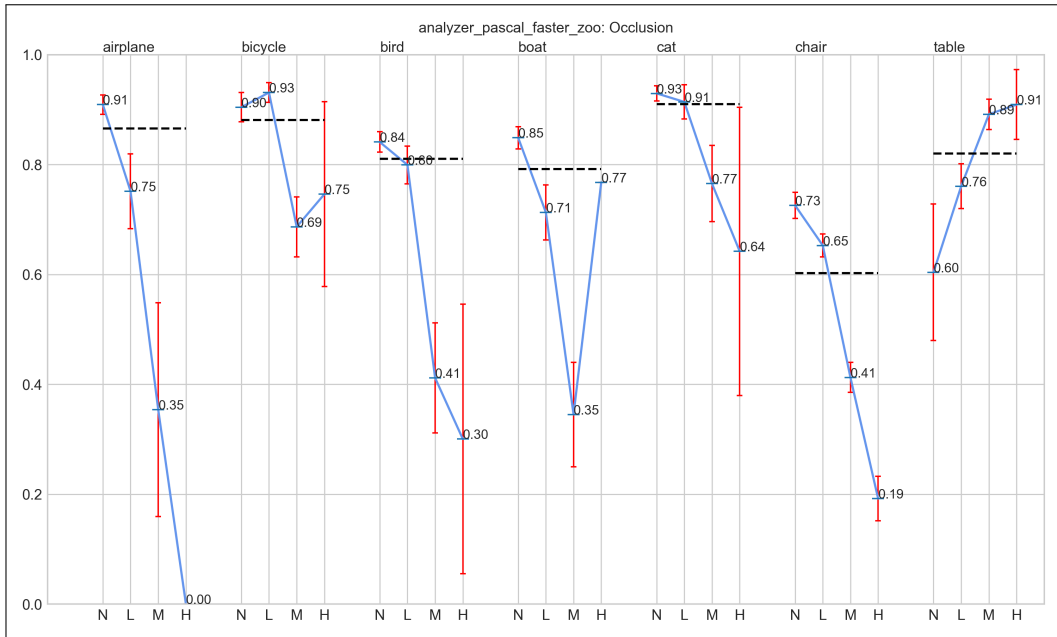


Figure 4.4: Occlusion graph generated after executing method *analyze_occlusion()*. In the X-axis the occlusion options per class are shown: None, Low, Medium, High. The Y-axis shows the AP_N . Black dashed lines indicate overall AP_N . Standard error bars are in red.

are listed below the graph according to the categories. These ranges are the result of the product *width-height*, thus they are measured in *square-pixels*. Figure 4.6 shows an example, the line-colors are the same than in the previous analysis. In the example, we can observe that the classes *airplane*, *cat* and *table* had the same increasing performance when varying from *extra-small* to *extra-large* area sizes. From this type of graph, one could infer for example, that the bigger is the object the easier it is to identify it; or find that at a certain threshold, the size of the object does not provide any improvement. This could help us to tune the model up or add some more images with a certain size to the dataset to improve.

Aspect ratio

The aspect-ratio method analyzes the performance level of a detector de-

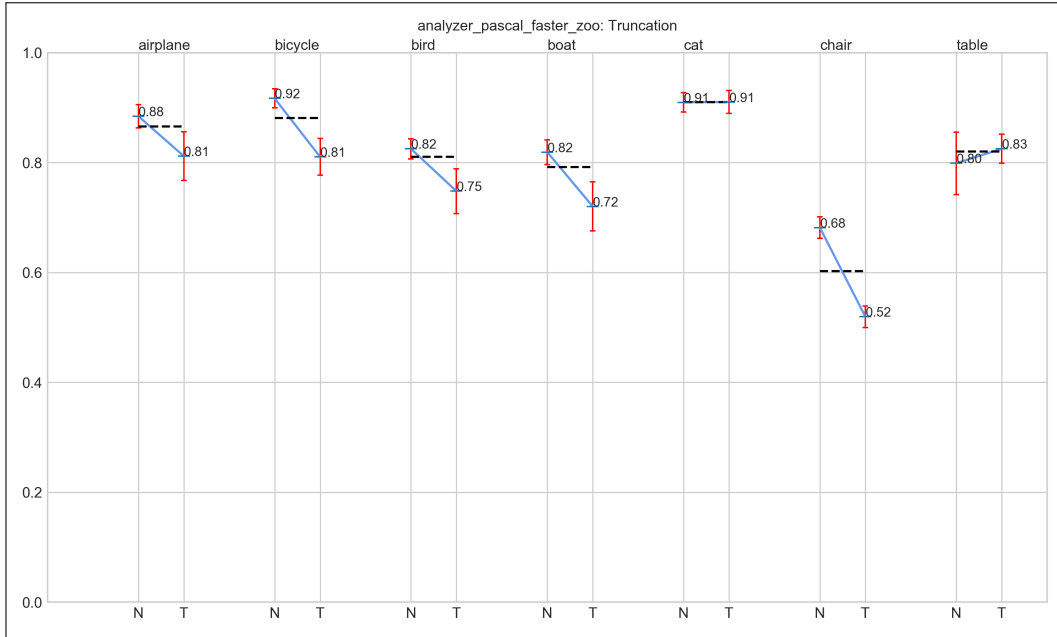


Figure 4.5: Truncation graph generated after executing method *analyze_truncation()*. The X-axis options are: N (Non-truncated), T (Truncated). The Y-axis shows the AP_N . Black dashed lines indicate overall AP_N . Standard error bars are in red.

pending on the object aspect-ratio (width/height). Call the method *analyze_aspect_ratio()*, and the graph as in Figure 4.7 would be drawn. Analogous to the line-colors in the previous analysis, in the example we can observe that *bicycles* with extra-thin characteristic were less identified along the other aspect ratio classification. Its best performance was acquire with the Medium classification (ranges [0.87-1.43]). That means bounding-boxes closer to a perfect square aspect ratio basis.

Parts visible

The Parts-visible method analyzes the performance level of a detector depending on what part of the object is seen in the images. Calling the method *analyze_parts_visible()* to make the analysis and a graph as in Figure 4.8 would be drawn. The line-colors are the same than in the previous analysis.

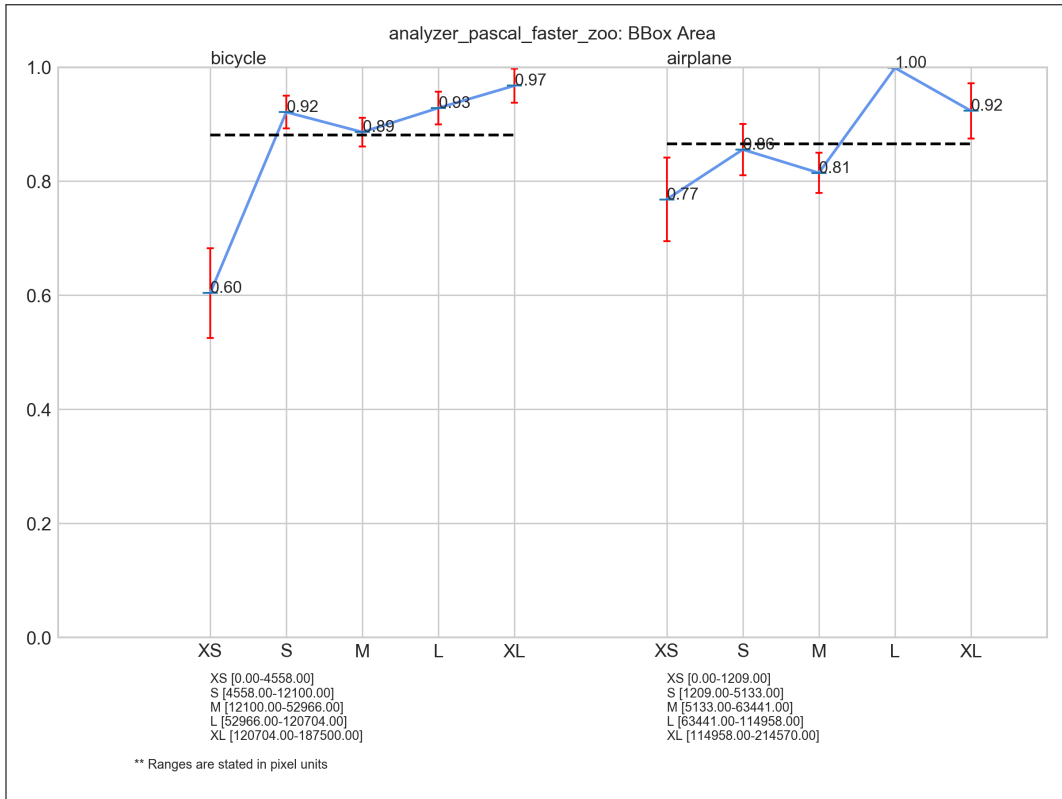


Figure 4.6: Bounding-box area graph generated after executing method *analyze_bbox_area()*. The X-axis options are: XS (extra-small), S (small), M (medium), L (large), XL (extra-large). The Y-axis shows the AP_N . Black dashed lines indicate overall AP_N . Standard error bars are in red. Sizes ranges are listed below the graph according to the categories in square-pixels unit.

The X-axis shows the parts that depend on the class category. Notice that in the example *airplanes* have four parts (body, head, tail, wing), and *tables* only two (table-leg, table-top). We can see in the case of *airplanes* that for all the four parts, when they are visible the accuracy improves, also that the parts that were not seen in the images that compromise the most the performance are head and wings. With this type of graph we could understand and discriminate which are the parts of the object that makes it more recognizable, and have the most relevance for a detector in order to identify

it.

Viewpoint

The viewpoint method analyzes the performance level of a detector depending on which side of the object is seen in the images. That means, what point-of-view is adopted in the line sight to the object. Calling the method `analyze_sides_visible()` to make the analysis and a graph as in Figure 4.9 would be the output. Analogous to the line-colors in the previous analysis, in the example we can observe that *airplanes* more or less had the same performance depending on which angle was seen in the image ($\approx 0.86 AP_n$ in average). In the case of the table class, we can see that the absence of the top point-of-view is the one with the higher negative impact in performance

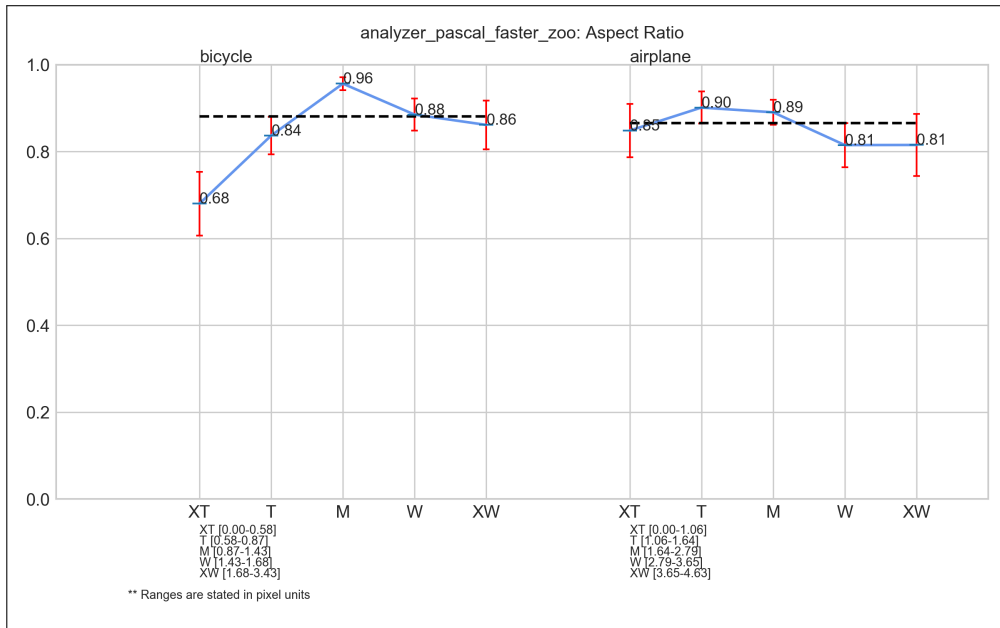


Figure 4.7: Aspect ratio analysis graph generated after calling method `analyze_aspect_ratio()`. The X-axis options are: XT (extra-thin), T (thin), M (medium), W (wide), XW (extra wide). The Y-axis shows the AP_N . Black dashed lines indicate overall AP_N . Standard error bars are in red. Sizes ranges are listed below the graph according to the options.

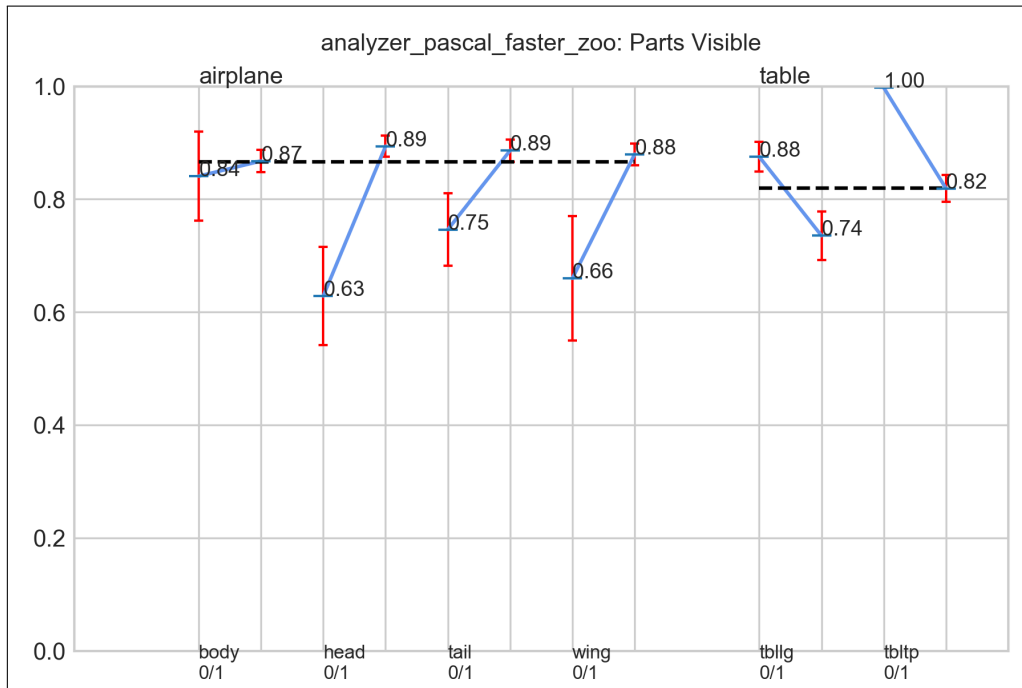


Figure 4.8: Parts-visible analysis graph generated after calling method *analyze_parts_visible()*. The X-axis indicates the visible parts of an object which depends on category. The Y-axis shows the AP_N . Black dashed lines indicate overall AP_N . Standard error bars are in red.

which is remarkably noticeable. Cases like this might urge to the researcher how to improve it, or if that is even relevant for his/her particular use case.

Sensitive and Impact over the overall AP

The sensitive and impact plot is summary graph that joins all the values from the previous analysis into one. The goal of this is to show how would improve the overall AP_n if a certain metrics is improved to the maximum and no misses occurs. In Figure 4.10 an example of the graph is showed.

In the figure, there are two values showed for each analysis. The maximum and minimum represent the performance in each type of metrics. The average values is computed among all the evaluated classes. The difference between the *maximum value* and the *minimum value* represents the sensitiveness, and

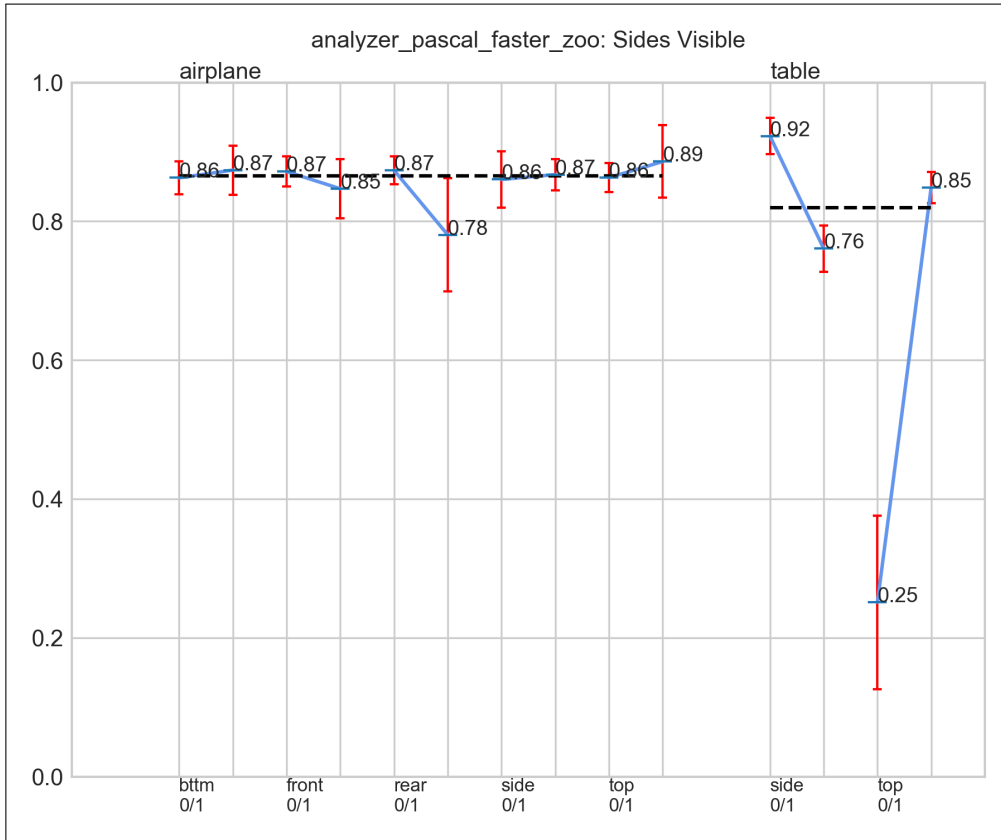


Figure 4.9: Viewpoint analysis (or sides visible) graph generated after calling method *analyze_sides_visible()*. The X-axis indicates view points options which depends on the object category analyzed. The Y-axis shows the AP_N . Black dashed lines indicate overall AP_N . Standard error bars are in red.

the difference between the maximum value and the average corresponds to the impact over the AP_n . For instance, we can see that in the Figure 4.10, the detector is more sensitive to occlusion ($0.87 - 0.446 = 0.424$). Moreover, the metric which can be improved the most is size, which can get to reach an impact of $0.111 AP_n$. This graph is usually more useful when comparing more than two detectors in general.

All True Positives metrics

If you want just to execute all the possible True Positives (TP) methods, you can just call *analyze_all_true_positives()* which would execute all the

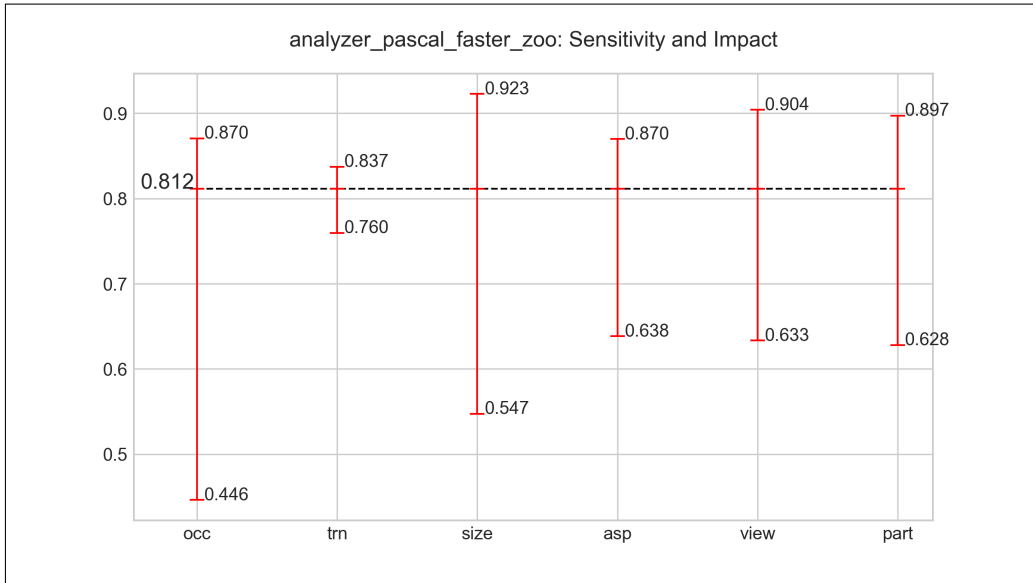


Figure 4.10: Sensitive and Impact analysis graph generated after calling method `analyze_avg_sensitivity_impact_plot()`. The X-axis indicates the type of evaluation. The Y-axis shows the AP_N . Black dashed lines indicates the overall AP_N . Difference between *max value* and *min value* indicate the sensitiveness; difference between max value and the average corresponds to the impact over the AP_n .

methods explained previously withing TP analysis group. Moreover, it is possible also to gather all the previous TPs analysis in a single graph grouped by class. In Figure 4.11 an example of a grouped graph of the class chair in PASCAL VOC 2007 [13] is showed. If you need this last graph, it can be invoked by the method `analyze_tp_per_class()`. This last method would generate a graph such as in the figure for each class.

4.3.2.3 False Positive analysis

In subsection 3.2.1 the FPs analysis categories were explained. There are two types of FPs analysis that can be called: `analyse_fp_impact()` and `analyse_fp_trending()`. These two methods method receives the same parameters as the TPs analysis above.

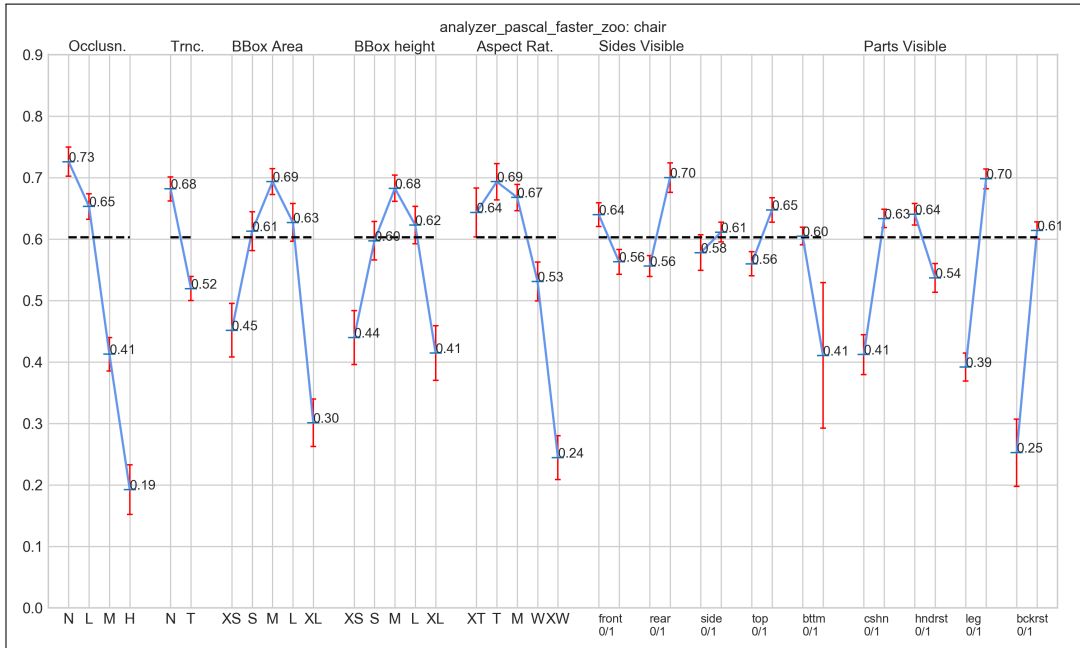


Figure 4.11: True Positive analysis per class generated after calling method `analyze_tp_per_class()`. The X-axis options depending of the metrics evaluated. The Y-axis shows the AP_N . Black dashed lines indicate overall AP_N . Standard error bars are in red.

First, The method `analyze_fp_impact()`, after producing the analysis, draws two types of graphs per each class. An example can be seen in the Figure 4.12 in which two analysis were carried; each analysis is composed of two graphs. The pie shows the fraction of top-ranked false positives that are due to poor localization (*Loc*), confusion with similar objects (*sim*), confusion with other category (*Other*) or confusion with background or unlabeled objects (*BG*). Moreover, the horizontal bar displays the absolute AP improvement by removing all FPs of one type. *B* removes confusion with background and non-similar objects. The first *L* part segment display improvement if duplicate or poor localizations are removed; the second displays improvements if the localization error were corrected, this means turning FPs into correct detections.

To give a brief comparison between the two classes, in the Figure 4.12

we can observe that the *airplane* category would have a better improvement of 0.06 AP if all localization errors are corrected, rather than in the *cat* category which just would improve in 0.034 AP. This also can be reviewed when observing the two category's pies. There are more *Location errors* in the *airplane* class than in the *cat* category in general.

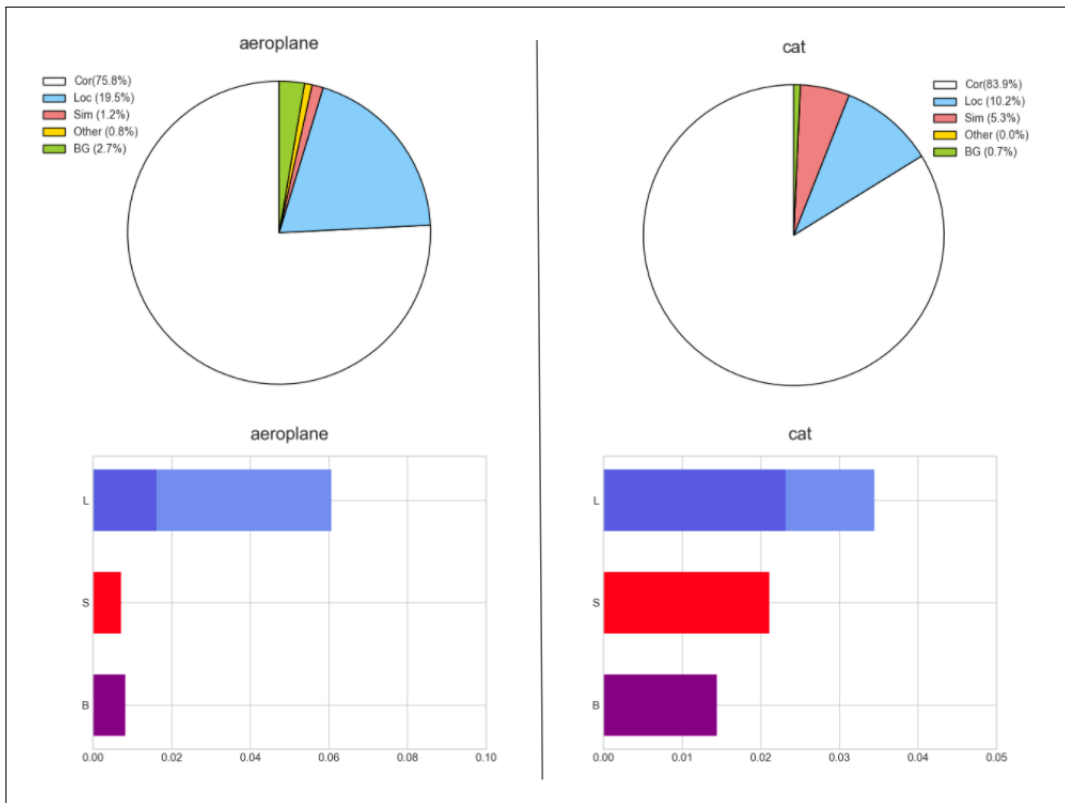


Figure 4.12: False positive impacts analysis generated after calling method *analyse_fp_impact()*. It draws two different types of graphs. The pie shows the percentage of FP depending on the category. Bar graphs display absolute AP improvement by removing all FP of one type. B: removes confusion with background and non-similar objects. L: the first bar segment displays improvement if duplicate or poor localization are removed; the second segment in lighter blue display improvement if the localization error were corrected, turning false detections into positives.

Secondly, the method *analyse_fp_trending()* generates an analysis in

how the trending occurs while the FP cases were identified during the analysis. This analysis draws two different types of graphs. The stacked-area shows the accumulated percentage of FP versus the number of FP cases presented in cuts of: 25, 50, 100, 200, 400, 800, 1000, 3200. These figures were chosen in order to make comparable the different graphs from all the classes. The scatter plot represents the same information as the graph above, but not in a non-stacked way basis. These graphs could be useful in order to understand the behaviour of false positive per class specific in the specific dataset. In the Figure 4.13 we can see an example of these graphs in airplanes and cat classes as in the previous analysis.

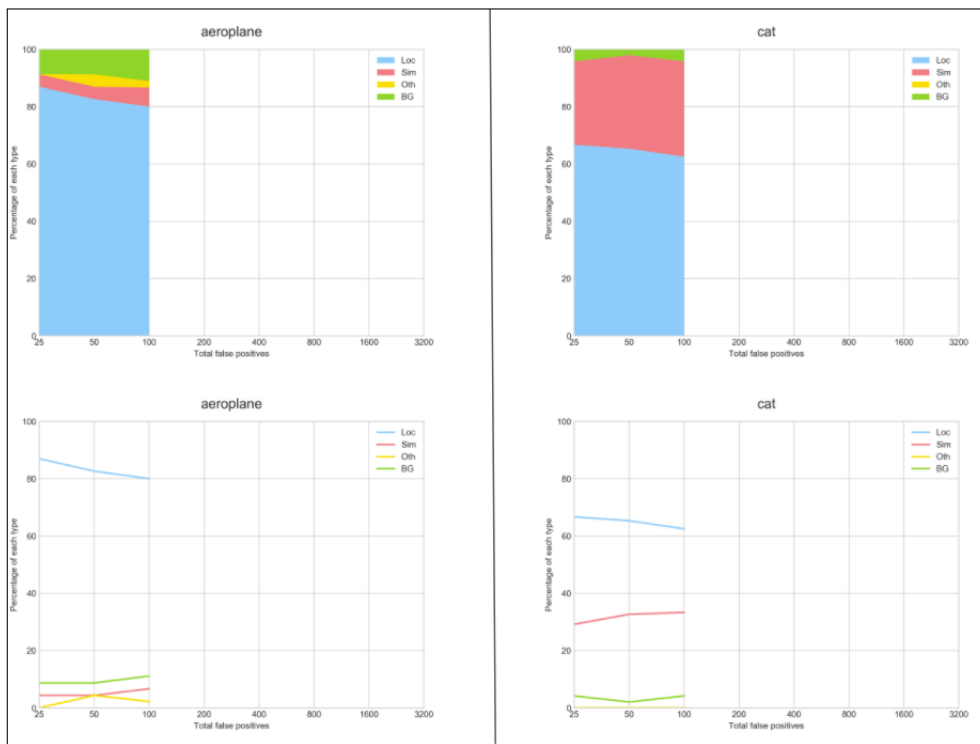


Figure 4.13: False positive trending analysis generated after calling method *analyse_fp_trending()*. It draws two different types of graphs. The stacked-area shows the accumulated percentage of FP vs the number of FP cases. The scatter plot represents the same information as the graph above, but not stacked.

4.4 Brief Tutorial

In this section we show a brief tutorial on how to call the previous methods. DME have been developed thinking in researchers that would like to improve their detector rather than working on the evaluation tool modification. As can be seen in Code Snippet 4.1, we only need to follow the steps below:

1. Import the classes: `Analyzer` and `DatasetVOC/DatasetCOCO`.
2. Provide the path parameters for the `Dataset` class.
3. After generating an instance of the `Analyzer`, you can call `load()` with the analyzer instance. `Load()` will check the paths and, according to the annotations available, it will load the data into memory.
4. Call any analysis you would like try from the previous section. You can also call `analyze_all_true_positive()` or `analyze_all_false_positive()` which would proceed to analyse every metric possible, according to the dataset available. In case that a metric might not be possible to run because of the lack of some extra annotation, the user would be notified in this case.

```
1 from dme import DatasetVOC, Analyzer
2 ...
3 my_dataset = DatasetVOC(dataset_gt_param=[path],
4     proposal_path=[path])
5
6 my_analyzer = Analyzer('my_analyzer', my_dataset)
7 my_dataset.load()
8 my_analyzer.analyze_occlusion() # call the analysis
9 my_analyzer.analyze_all_true_positive()
```

Snippet 4.1: DME snippet sample.

Chapter 5

Testing & Results

In this chapter we present the testing and results obtained with our developed framework *Detector Metrics Evaluator* (DEM). First, we present three testings settings in Section 5.1. Then, the training and validation of the three settings are described in Section 5.2. Finally in Section 5.3, the testing results are presented for the three testing scenarios.

5.1 Testing settings

In order to test the DME framework we propose three testing settings summarized in Table 5.1. The main goal of these three configurations is to prove the independence of DME with respect to the the training framework and datasets.

For these Settings, two different datasets were utilized. Firstly, the well-known Pascal VOC 2007 [13] is a mid-scale dataset for general object classification and detection divided in 20 categories. The three splits (training, val-

	Dataset	Pipeline	Training Framework
Setting 1	Pascal VOC 2007	FasterRCNN: R50-C4	Detectron2
Setting 2	Pascal VOC 2007	FasterRCNN: R50-FPN	MMDetection
Setting 3	iSAID	FasterRCNN: R50-FPN	Detectron2

Table 5.1: Training settings summary.

idation and tests) counts in total with 10,022 images annotated with 24,540 objects. Secondly, *iSAID* dataset is a large-scale dataset for object detection and instance segmentation. The images are originally to aerial images from the DOTA dataset [48], and the annotation is based in the MSCOCO format [43], which DME implemented. *iSAID* counts with 665,451 object instances in 15 categories across 2,806 high-resolution images.

Regarding the two training frameworks employed, on the first hand, *Detectron2* [16] is one of the most used ones because of its speed and relative good documentation available online. It is developed and maintained by Facebook Research Lab ¹ based on PyTorch ². On the other hand, MMDetection [17] is an open source object detection toolbox also based on PyTorch ². It is part of the *OpenMMLab* project developed and maintained by the *Multimedia Laboratory, CUHK* (Chinese University of Hong Kong). Both are well-known projects with their own pros and cons.

5.1.1 Hardware settings

The hardware settings of the running machine was an instance of Ubuntu 18.0.4 Server deployed over Docker. The processor was an Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz with 64GB of memory. The server counted with two NVIDIA GeForce RTX 2080 Ti ³ with 11GB each. The dataset was stored in a Solid-State-Drive (SSD) to improve read/write latency.

5.2 Training & validation

In this section we are going to describe each setting training and validation from Table 5.1.

¹<https://github.com/facebookresearch/detectron2>

²<https://pytorch.org/>

³<https://www.nvidia.com>

5.2.1 Setting 1: Detectron2

For training and validation using the PascalVOC 2007 dataset, we applied the weights from a model that was already trained & validated in both PascalVOC 2007-2012 (train-val sets) available in the repository from the training framework ¹. According to the log in the previous repository, the model reached a mAP of 51.9% over the testing dataset.

5.2.2 Setting 2: MMDetection

The MMDetection [17] pipeline was trained from a pretrained baseline ResNet50 (48 Convolution layers along with 1 MaxPool and 1 Average Pool layer). We run 20 epochs, from which the best epoch turn out to be the iteration number 11. This last achieved a 65.70% mAP performance over the validation dataset, which was the one used for the testing purposes. The initial learning rate was set to 0.01.

5.2.3 Setting 3: iSAID dataset

We let *Detectron2* to train the instances for 100,000 epochs for the Faster RCNN pipeline [40] over iSAID training dataset (1411 high-scale aerial images). Since the 11Gb memory from our available GPU were not enough to fit the high-resolution images for the runnings, we patched the images into 800x800 pixels sizes. During training, we applied transfer learning [65] by using the weights from a pretrained setting of the same pipeline over MSCOCO dataset [43]. Even though we did not necessarily aim to get a high precision accuracy in the training due that this testing were oriented to test DME framework, the last epoch achieved an mAP of 39.8% over the iSAID validation dataset. The initial learning rate was configured to 0.001.

¹https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md

5.3 Test results

In this section we present our testing inference running & the analysis over the three proposed settings. In subsection 5.3.1 the TP analysis for both Setting 1 and Setting 2 are showed. Then in the following subsection 5.3.2 a FP evaluation is explained. A discussion of the previous TP and FP is outlined in subsection 5.3.3. Finally in subsection 5.3.4 a last testing result is showed for Setting 3.

5.3.1 Setting 1 & Setting 2: True Positives Analysis

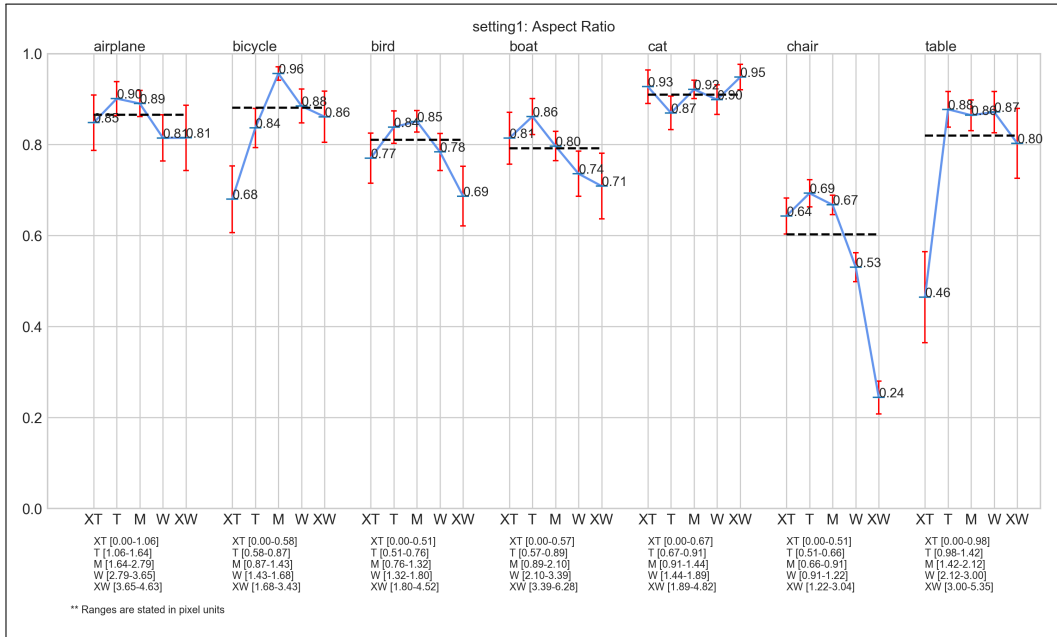
In this section, the True Positives Analysis for both Setting 1 (S1) & Setting 2 (S2) are showed. The goal is to test the DME framework with results coming from different training-frameworks (S1, *Detectron2* [16] and S2, *MMDetection* [17]) to prove the neutrality of our tool with respect to the training framework. Even though the results were grouped, there are not intentions to compare both S1 and S2 settings considering that the parameters and number of training were not even similar.

5.3.1.1 Aspect ratio

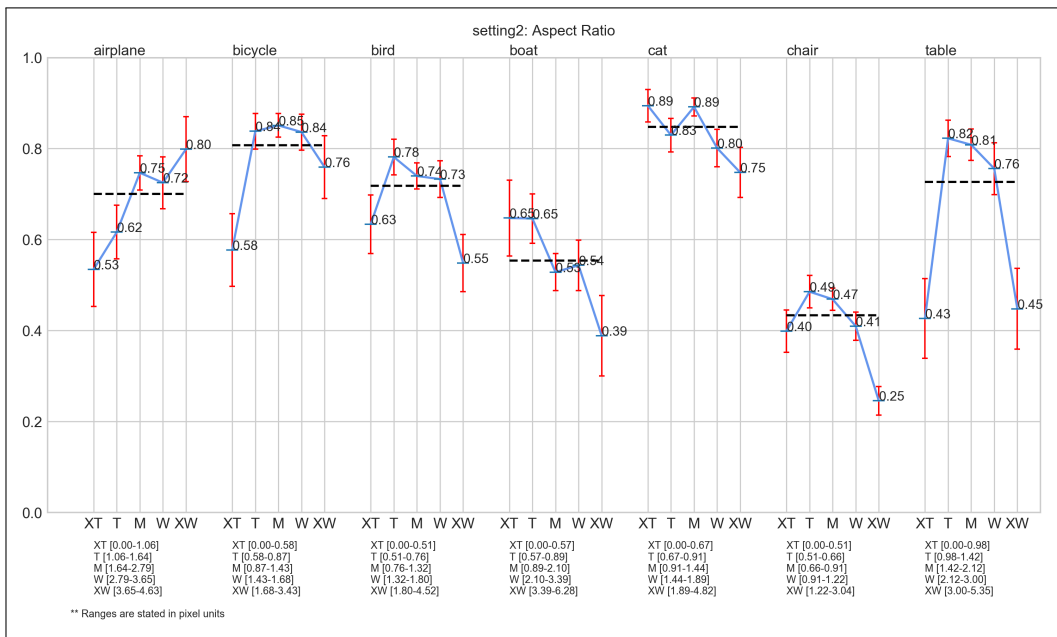
In Figure 5.1 we can see the results of the Aspect Ratio (width/height ratio) analysis in both Setting 1 (S1) and Setting (S2). Both detectors had their best performance identifying the cat class. Nevertheless, both had problems identifying chairs. If we check the class airplane in both graphs, we see that S1 performed better when the airplane had thin aspect, but S2 preferred extra-wide airplanes. Overall we can see that in both settings, the classes behaved similarly.

5.3.1.2 Truncation evaluation

In Figure 5.2 we can see the truncation analysis for both Setting 1 (S1) and Setting (S2). If we compare an overall behaviour of both settings detector we can see that every class except airplanes had the same behaviour. We can



(a) Setting 1: Aspect Ratio



(b) Setting 2: Aspect Ratio

Figure 5.1: Setting 1 & 2: Aspect ratio evaluation from 7 classes: airplane, bicycle, bird, boat, cat, chair, table. Pascal VOC 2007 dataset [13].

see that S1 preferred non-truncated airplanes, while S2 identified truncated ones better.

5.3.1.3 Occlusion Evaluation

The occlusion assessment is showed in Figure 5.3. For both Setting 1 (S1) and Setting 2 (S2) detectors the classes bicycle, bird, chair and table had the same descendant behaviour for the identification of the occlusion category (see that blue form). Both S1 and S2 identify bicycles better when they have low occlusion (probably when there is a person riding the bike). Moreover, the worst performance in the same category occurs when medium-occlusion was present.

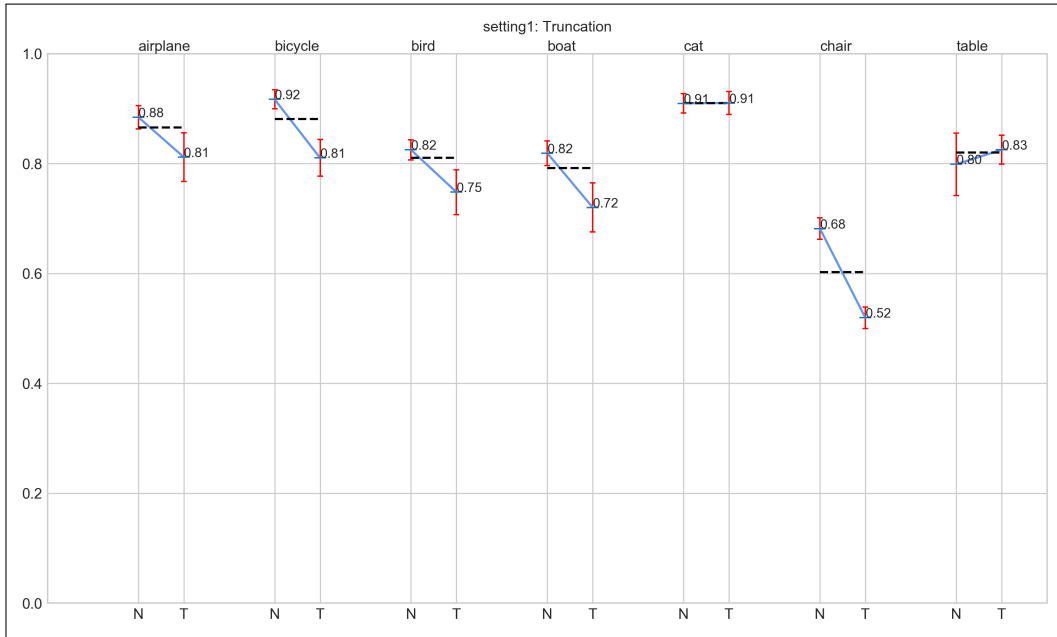
5.3.1.4 Bounding-box area evaluation

In Figure 5.4 we can observe the graphs from the Bounding-Box area evaluation. The area is defined as the product width-height. In setting 1 (S1), the airplane class had a 100% performance when identifying areas in the range Large [63,441-114.958 ps] (notice the options range below the figure). While for the same class Setting 2 (S2) achieved a 95%. Moreover, if we focus on the boat category, S1 achieved a great performance in various of the area sizes with an average of 0.79 AP_N . Nonetheless, S2 obtain a 0.56 AP_N in average.

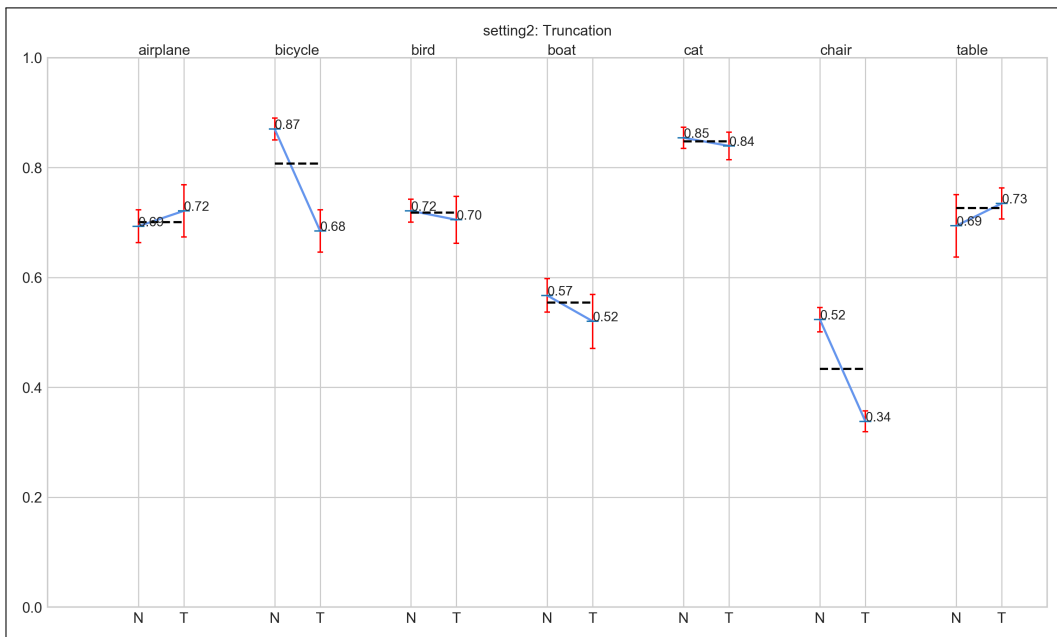
5.3.1.5 Sensitive and Impact

As the last TP evaluation for for both Setting 1 (S1) and Setting 2 (S2), we propose to analyze the Figure 5.5. As explained in section 4.3.2.2, the analysis joins all the possible TP positive analysis and give an overall perspective of the metric assessment. First, if we focus in the S1 graph, we can see that it achieved a good overall performance in all the studied metrics being truncation the less sensitive one.

Regarding Setting 2 (S2) detector. If we check the area size, we can see that S2 is quite sensitive to it (0.586 AP_N) in comparison to the other metrics. Furthermore, S2 area size have an impact of 0.185 AP_N . This means that the

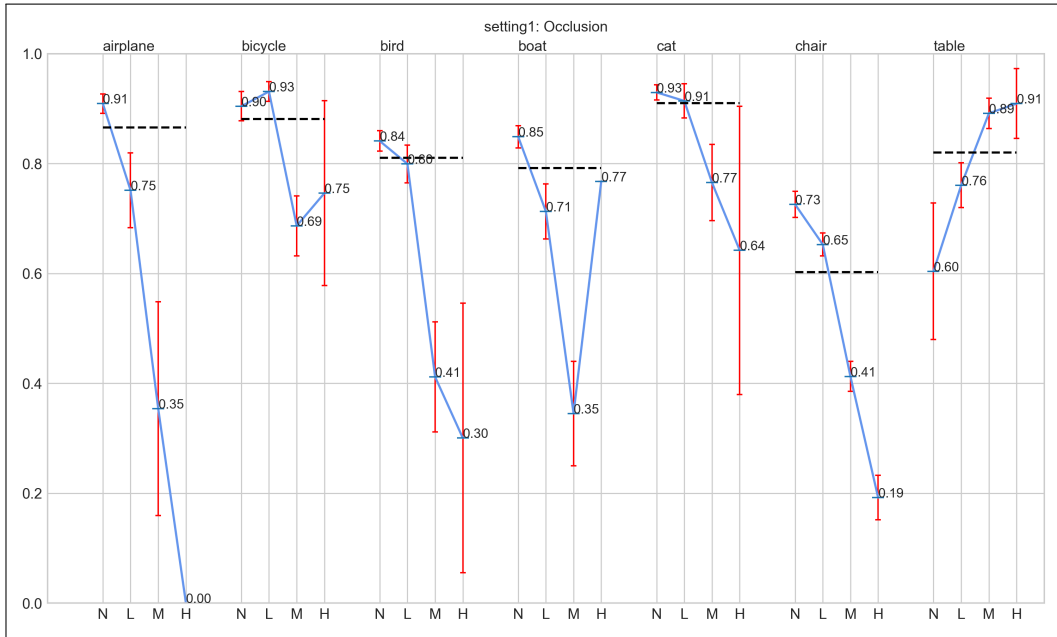


(a) Setting 1: Truncation

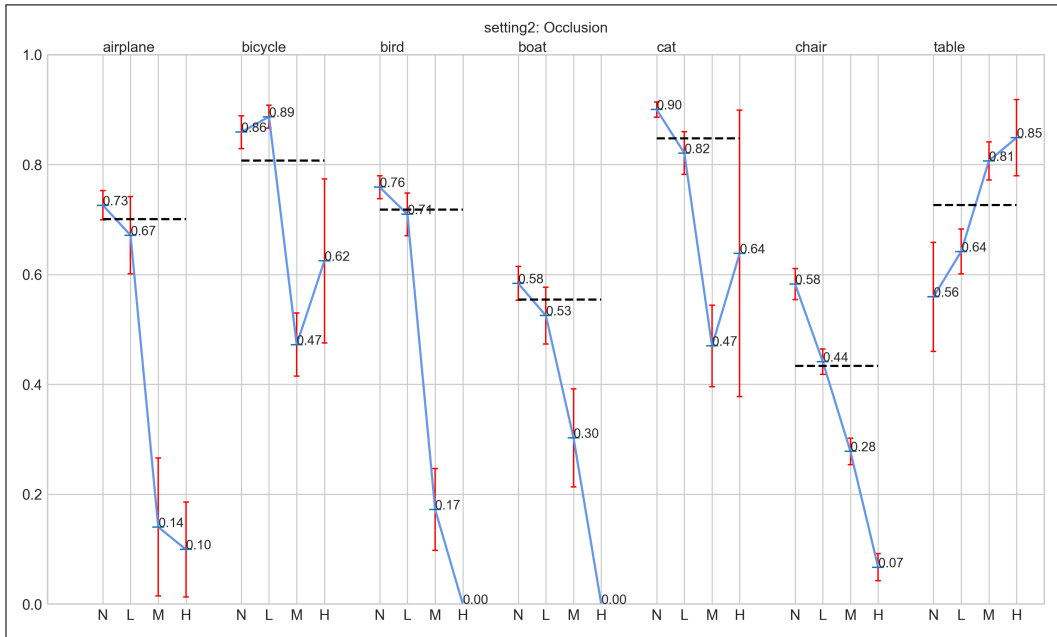


(b) Setting 2: Truncation

Figure 5.2: Setting 1 & 2: Truncation evaluation from 7 classes: airplane, bicycle, bird, boat, cat, chair, table. Pascal VOC 2007 dataset [13].

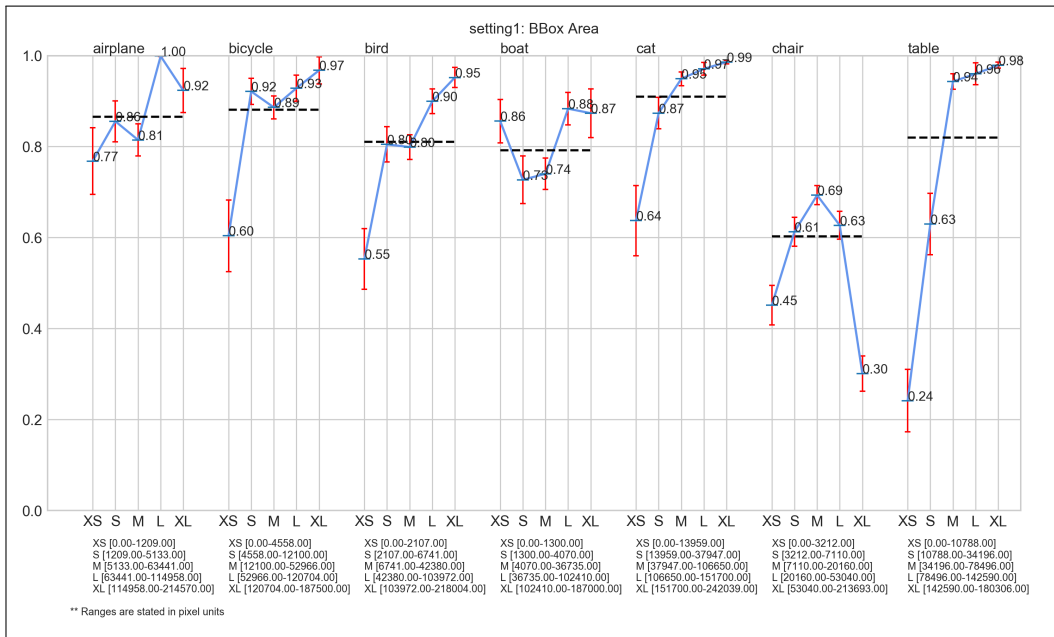


(a) Setting 1: Occlusion

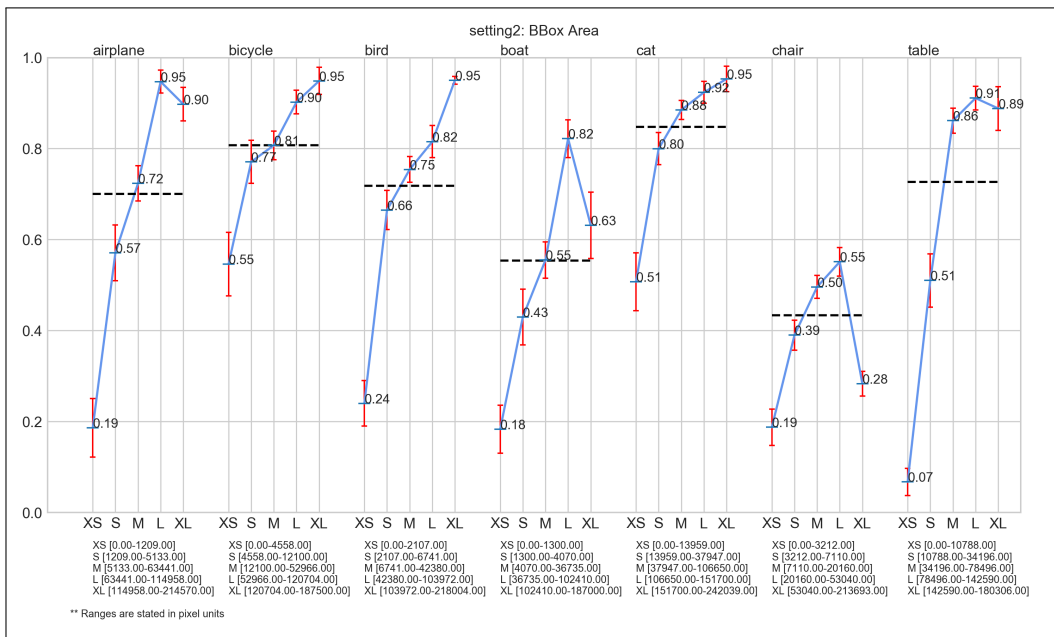


(b) Setting 2: Occlusion

Figure 5.3: Setting 1 & 2: Occlusion evaluation from 7 classes: airplane, bicycle, bird, boat, cat, chair, table. Pascal VOC 2007 dataset [13].



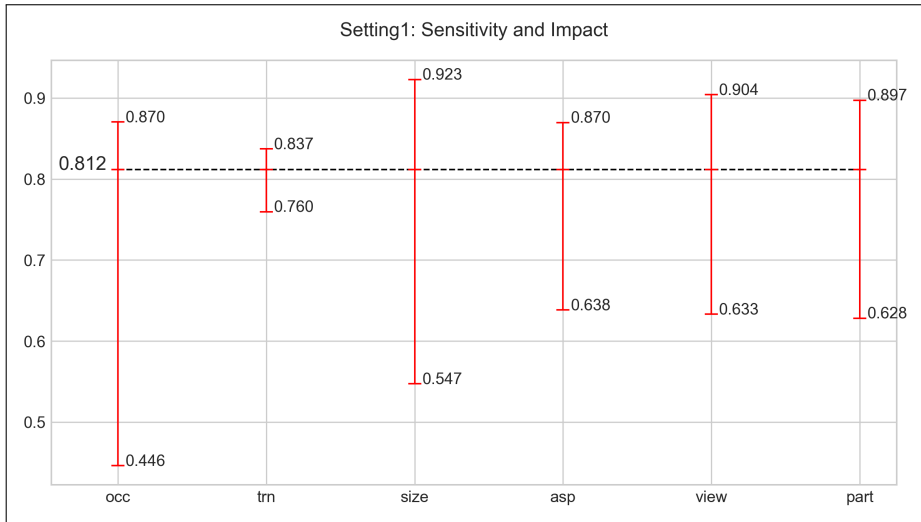
(a) Setting 1: Bounding-box area



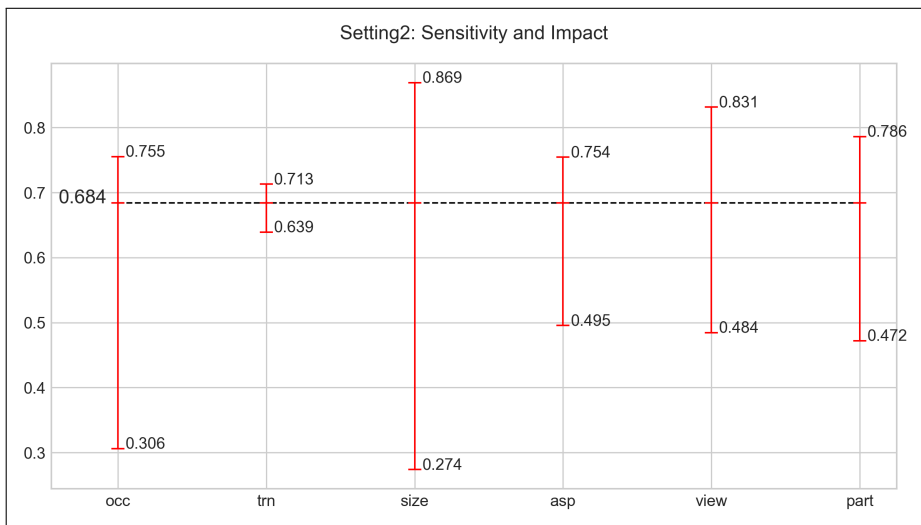
(b) Setting 2: Bounding-box area

Figure 5.4: Setting 1 & 2: Bounding-box area evaluation from 7 classes: airplane, bicycle, bird, boat, cat, chair, table. Pascal VOC 2007 dataset [13].

mAP_N could be improved a lot if we focus on improving the performance in Bounding-boxes area metric rather than the other metrics. As occurs with S1, the truncation turns to be less sensitive among the other metrics.



(a) Setting 1: Sensitive and Impact evaluation



(b) Setting 2: Sensitive and Impact evaluation

Figure 5.5: Setting 1 & 2: Sensitive and Impact evaluation evaluation from 7 classes: airplane, bicycle, bird, boat, cat, chair, table. Pascal VOC 2007 dataset [13].

5.3.2 Setting 2: False Positives testing results

The False Positives analysis provide more insight when they are used to evaluate just one detector performance. In Figure 5.6 we can see the airplane and dinning-table classes FP impact analysis.

In the airplane class we can see that most of the FP was due to Location misses (27%). If we work to remove the location type errors from the category, we would get an improvement of 0.139 AP which is a considerable amount. Even though *dining table* had most of only 53% correct, the performance in the location errors would be a little bit less than the *airplane* category (0.13 AP); this happens because there are more airplanes objects than dinning, 285 and 206 respectively. Regarding the similar categories in Pascal VOC, the detector in Setting 2 got confused a lot with *sofas* and *chairs*. Moreover, S2 found a considerable number of misses with background or other similar objects. Even though this number may seems a lot, the possible impact to the AP would be just 0.06 if we get to correct this flaw from it.

5.3.3 True Positive and False Positive analysis discussion

The main goal for the previous TP and FP analysis applying the Setting 1 and Setting 2 was to prove that our DME implemented framework is detached to the training framework. After generating the input format for the DME from both Detectron2 and MMDetection, we have proved the previous goal.

Overall, there were some insights that showed the robustness of the analysis, independently of the setting. For example, in both settings we can see that the size of the object can have a noticeable impact on the performance, and that could be one direction in which the user could dig into to improve on further experiments. In this case, small tables are the ones that performed the worst, so maybe some data augmentation could be applied to improve in this aspect. It is important to mention that these were random setting that we decided to apply to focus mostly in the tool testing. Even though we have just compared two different training frameworks, we can let the model

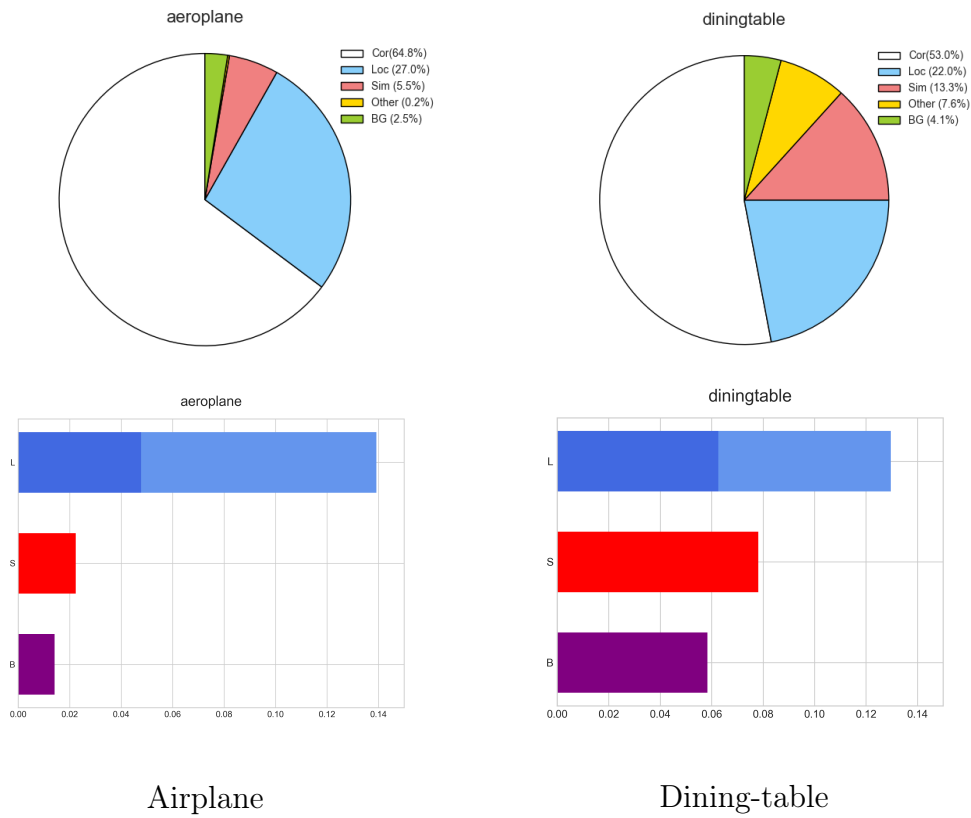


Figure 5.6: False Positive impact evaluation of two PASCAL VOC classes with Setting 2 detector.

fix and vary just one parameter to check if it has some influence in specific size/aspect ratio of objects.

5.3.4 Setting 3: Testing

In this section, the goal is to test the DME framework works with another dataset besides *Pascal VOC 2007* [13]. The chosen dataset for this was iSAID [18] to assess the behaviour of the pipeline (FasterRCNN [40]) with Detectron 2 [16].

5.3.4.1 iSAID testing dataset preparation

Before getting into the testing inference task, we had to apply a modification to the famous train-val-testing sets for Setting 3. Since *iSAID* testing test do not count with annotations, we half-splitted the validation set to use one of the halves as testing purposes, and the other half as the validation set. After applying inference to our testing dataset (half-split of the original validation set, 229 high-scale images), we achieve a 38.351 mAP. Over the predicted bounding boxes as proposals and our testing ground-truth dataset as inputs for the DME framework, we obtained the following analysis evaluation below.

5.3.4.2 Setting 3: Testing results

In this subsection, we show the results after analyzing the precision-recall curves along with the TPs results displayed per class graphs with Setting 3 context.

Precision recall curves and per-class True Positive Analysis

In Figure 5.7 you can find the precision-recall curve from *Setting 3*(S3) detector. In the figure, we see that tennis-court class achieved the best performance identifying objects from its categories. When checking Figure 5.8.(a), we can see that S3 preferred extra-thin tennis court. This could be a good starting point to check if the training dataset counts more with thinner aspect ratios tennis court rather than wider ones.

Regarding the bridges class in S3 result, the class acquired the worst performance Figure 5.7. When checking the Figure 5.8 we can see the Medium Aspect-ratio bridges were the best identified with only 0.54 AP_N .

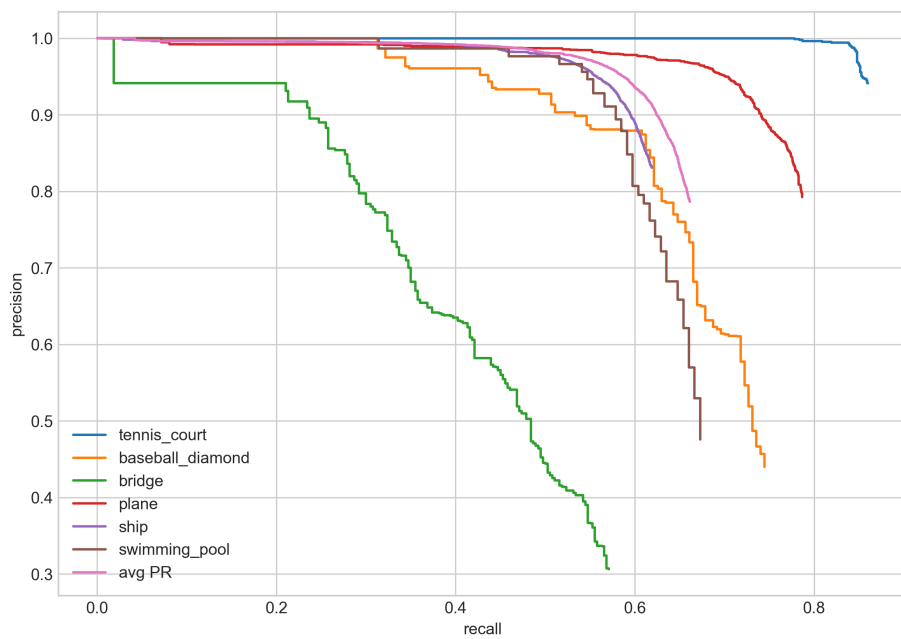
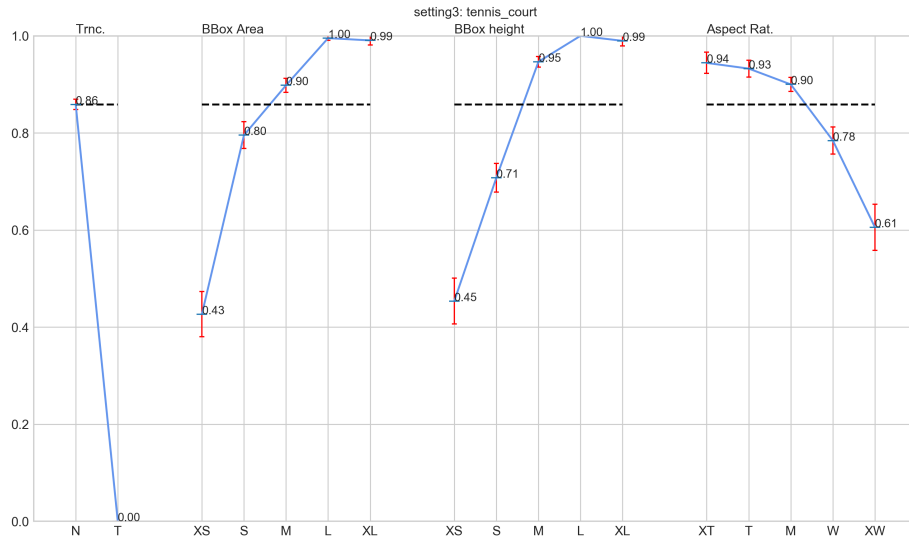
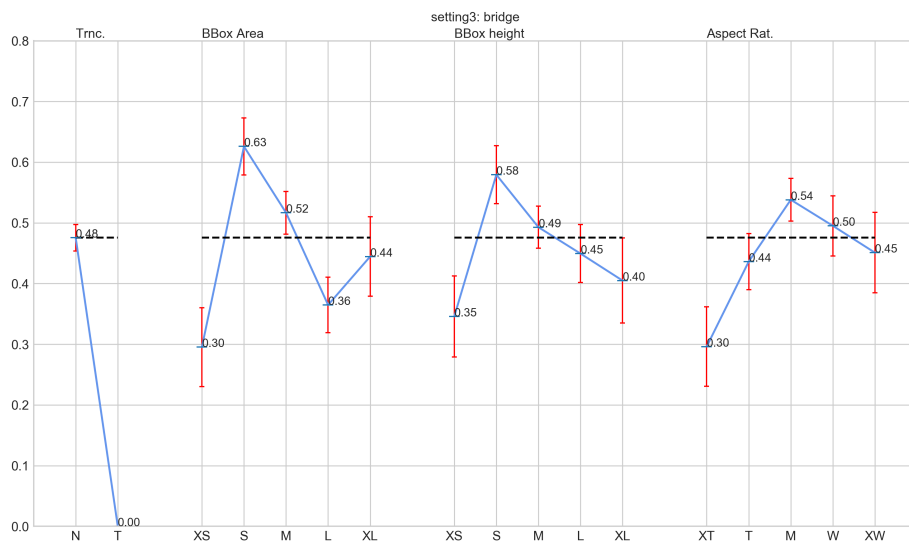


Figure 5.7: Precision-recall evaluation from *Setting 3* detector over iSAID classes



(a) Setting 3: Tennis-court category classes



(b) Setting 3: Bridge category

Figure 5.8: Setting 3: Per class True Positive analysis over the best and worst classes performed in the testing results.

Chapter 6

Conclusion and future works

In this work we have proposed and developed a framework library tool for evaluating object detection pipelines in CV. The *Detector Metrics Evaluator* has proved to be a versatile and practical tool for researchers that would like to analyze their detector proposals with the ground-truth dataset. These inherent advantages comes from the interpreted almost natural language programming as Python.

Moreover, because it has been design utilizing *Object Programming Principles* (OOP), it provides to developers the possibility to scale it by adding new datasets and metrics over the already implemented platform. Currently it supports two types of dataset formats: Pascal VOC 2007 and MSCOCO formats.

We have tested our tool with three tests settings which included a pipeline such as *FasterRCNN* with two training framework (*Detectron2* and *MMDetection*) two different datasets (PASCAL VOC 2007 and MSCOCO). We have proved that the tool can generate interesting analysis and provide powerful feedback to researches in order to improve their performance, and even fine tuning them. Moreover, with these testing settings, we have validated the neutrality of DME with respect to the training framework and the dataset as well.

Nowadays, the wide use of Object Detection algorithms in real-life situation has pushed to researchers in public and even private companies, incre-

ment the robustness of their algorithm, and DEM has proved to be a suitable tool for that task.

To continue with this work we envision the following:

- First, we would like to increase the scope of our framework by adding the possibility to evaluate not only object detection scenarios, but also apply it to image segmentation cases. We have analyzed the developed modules and we believe that with small changes to them we can achieve to include image segmentation evaluations in the series of options already available.
- Second, we would like to let the code into an open repository such that other researches in CV can make use of it, and also extend it by adding more dataset formats, and metrics evaluations.
- Third, add a GUI component over the DME library aimed to non-coders so that they can make use of the tool by just setting the parameters through an user interface.
- Last and the most ambitious one, we would like to add an intelligent component to suggest, where possible, interpretations and model corrective actions based on the value of specific metrics.

Bibliography

- [1] James J DiCarlo, Davide Zoccolan, and Nicole C Rust. «How does the brain solve visual object recognition?» In: *Neuron* 73.3 (2012), pp. 415–434.
- [2] *Wake Up and Drive: Fatigue Causes 20 Percent of Crashes | EHS Today*. <https://www.ehstoday.com/safety/article/21917988/wake-up-and-drive-fatigue-causes-20-percent-of-crashes>. (Accessed on 05/09/2020).
- [3] Sahar V Doctorvaladan et al. «Accuracy of blood loss measurement during cesarean delivery». In: *American Journal of Perinatology Reports* 7.02 (2017), e93–e100.
- [4] Scott Mayer McKinney et al. «International evaluation of an AI system for breast cancer screening». In: *Nature* 577.7788 (2020), pp. 89–94.
- [5] *AI startup Gather uses drones and computer vision for warehouse inventory | VentureBeat*. <https://venturebeat.com/2019/08/15/ai-startup-gather-uses-drones-and-computer-vision-for-warehouse-inventory/>. (Accessed on 05/09/2020).
- [6] Sander Suursalu. «Predictive maintenance using machine learning methods in petrochemical refineries». In: (2017).
- [8] Garrick Brazil, Xi Yin, and Xiaoming Liu. «Illuminating pedestrians via simultaneous detection & segmentation». In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 4950–4959.

- [9] Xianzhi Du et al. «Fused DNN: A deep neural network fusion approach to fast and robust pedestrian detection». In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, pp. 953–961.
- [10] Shiguang Wang et al. «Pcn: Part and context information for pedestrian detection with cnns». In: *arXiv preprint arXiv:1804.04483* (2018).
- [11] Rodrigo Benenson et al. «Ten years of pedestrian detection, what have we learned?» In: *European Conference on Computer Vision*. Springer. 2014, pp. 613–627.
- [12] Thomas Huang. «Computer vision: Evolution and promise». In: (1996).
- [13] Mark Everingham et al. «The Pascal Visual Object Classes (VOC) Challenge». In: *Int. J. Comput. Vision* 88.2 (June 2010), pp. 303–338. ISSN: 0920-5691. DOI: 10.1007/s11263-009-0275-4. URL: <https://doi.org/10.1007/s11263-009-0275-4>.
- [14] Gregory Griffin, Alex Holub, and Pietro Perona. «Caltech-256 object category dataset». In: (2007).
- [15] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. «Diagnosing error in object detectors». In: *European conference on computer vision*. Springer. 2012, pp. 340–353.
- [16] Yuxin Wu et al. *Detectron2*. 2019. URL: <https://github.com/facebookresearch/detectron2>.
- [17] Kai Chen et al. «MMDetection: Open MMLab Detection Toolbox and Benchmark». In: *arXiv preprint arXiv:1906.07155* (2019).
- [18] Syed Waqas Zamir et al. «iSAID: A Large-scale Dataset for Instance Segmentation in Aerial Images». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2019, pp. 28–37.

- [19] Andrea Vedaldi et al. «Multiple kernels for object detection». In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 606–613.
- [20] Paul Viola and Michael Jones. «Rapid object detection using a boosted cascade of simple features». In: *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. Vol. 1. IEEE. 2001, pp. I–I.
- [21] Hedi Harzallah, Frédéric Jurie, and Cordelia Schmid. «Combining efficient object localization and image classification». In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 237–244.
- [22] Navneet Dalal and Bill Triggs. «Histograms of oriented gradients for human detection». In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 886–893.
- [23] Paul Viola and Michael J Jones. «Robust real-time face detection». In: *International journal of computer vision* 57.2 (2004), pp. 137–154.
- [24] David G Lowe. «Object recognition from local scale-invariant features». In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.
- [25] Rainer Lienhart and Jochen Maydt. «An extended set of haar-like features for rapid object detection». In: *Proceedings. international conference on image processing*. Vol. 1. IEEE. 2002, pp. I–I.
- [26] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. «Surf: Speeded up robust features». In: *European conference on computer vision*. Springer. 2006, pp. 404–417.
- [27] Marti A. Hearst et al. «Support vector machines». In: *IEEE Intelligent Systems and their applications* 13.4 (1998), pp. 18–28.
- [28] Yoav Freund, Robert E Schapire, et al. «Experiments with a new boosting algorithm». In: *icml*. Vol. 96. Citeseer. 1996, pp. 148–156.

- [29] David Opitz and Richard Maclin. «Popular ensemble methods: An empirical study». In: *Journal of artificial intelligence research* 11 (1999), pp. 169–198.
- [30] Pedro Felzenszwalb et al. «Discriminatively trained mixtures of deformable part models». In: *PASCAL VOC Challenge* (2008).
- [31] David G Lowe. «Distinctive image features from scale-invariant keypoints». In: *International journal of computer vision* 60.2 (2004), pp. 91–110.
- [32] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. «Multiresolution gray-scale and rotation invariant texture classification with local binary patterns». In: *IEEE Transactions on pattern analysis and machine intelligence* 24.7 (2002), pp. 971–987.
- [33] Yann LeCun, Y. Bengio, and Geoffrey Hinton. «Deep Learning». In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.
- [34] Ivars Namatēvs. «Deep convolutional neural networks: Structure, feature extraction and training». In: *Information Technology and Management Science* 20.1 (2017), pp. 40–47.
- [35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks». In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.
- [36] Karen Simonyan and Andrew Zisserman. «Very deep convolutional networks for large-scale image recognition». In: *arXiv preprint arXiv:1409.1556* (2014).
- [37] Kaiming He et al. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [38] Olga Russakovsky et al. «ImageNet Large Scale Visual Recognition Challenge». In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

- [39] Ross Girshick et al. «Rich feature hierarchies for accurate object detection and semantic segmentation». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.
- [40] Shaoqing Ren et al. «Faster r-cnn: Towards real-time object detection with region proposal networks». In: *Advances in neural information processing systems*. 2015, pp. 91–99.
- [41] Li Liu et al. «Deep learning for generic object detection: A survey». In: *International journal of computer vision* 128.2 (2020), pp. 261–318.
- [42] Mark Everingham et al. «The pascal visual object classes challenge: A retrospective». In: *International journal of computer vision* 111.1 (2015), pp. 98–136.
- [43] Tsung-Yi Lin et al. «Microsoft coco: Common objects in context». In: *European conference on computer vision*. Springer. 2014, pp. 740–755.
- [44] Alina Kuznetsova et al. «The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale». In: *arXiv preprint arXiv:1811.00982* (2018).
- [45] Xiongwei Wu, Doyen Sahoo, and Steven CH Hoi. «Recent advances in deep learning for object detection». In: *Neurocomputing* (2020).
- [47] Jia Deng et al. «Imagenet: A large-scale hierarchical image database». In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
- [48] Gui-Song Xia et al. «DOTA: A large-scale dataset for object detection in aerial images». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3974–3983.
- [49] Dan Clark. *Top 16 Open Source Deep Learning Libraries and Platforms*. <https://www.kdnuggets.com/2018/04/top-16-open-source-deep-learning-libraries.html>. (Accessed on 05/11/2020). Apr. 2018.

- [50] Bradley J Erickson et al. «Toolkits and libraries for deep learning». In: *Journal of digital imaging* 30.4 (2017), pp. 400–405.
- [51] Francisco Massa and Ross Girshick. *maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch*. <https://github.com/facebookresearch/maskrcnn-benchmark>. (Accessed on 05/11/2020). 2018.
- [52] Yuntao Chen et al. «SimpleDet: A Simple and Versatile Distributed Framework for Object Detection and Instance Recognition». In: *Journal of Machine Learning Research* 20.156 (2019), pp. 1–8. URL: <http://jmlr.org/papers/v20/19-205.html>.
- [53] Waleed Abdulla. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. https://github.com/matterport/Mask_RCNN. 2017.
- [54] Vladimir Y Mariano et al. «Performance evaluation of object detection algorithms». In: *Object recognition supported by user interaction for service robots*. Vol. 3. IEEE. 2002, pp. 965–969.
- [55] Christian Wolf and Jean-Michel Jolion. «Object count/area graphs for the evaluation of object detection and segmentation algorithms». In: *International Journal of Document Analysis and Recognition (IJDAR)* 8.4 (2006), pp. 280–296.
- [56] Sergio Lima Netto Rafael Padilla and Eduardo A. B. da Silva. «Survey on Performance Metrics for Object-Detection Algorithms». In: *International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020.
- [57] Simon M Lucas et al. «ICDAR 2003 robust reading competitions: entries, results, and future directions». In: *International Journal of Document Analysis and Recognition (IJDAR)* 7.2-3 (2005), pp. 105–122.
- [58] *The PASCAL Visual Object Classes Homepage*. URL: <http://host.robots.ox.ac.uk/pascal/VOC/> (visited on 01/15/2020).

- [59] Stephen V Stehman. «Selecting and interpreting measures of thematic classification accuracy». In: *Remote sensing of Environment* 62.1 (1997), pp. 77–89.
- [60] David Martin Powers. «Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation». In: (2011).
- [61] Raimundo Real and Juan M Vargas. «The probabilistic basis of Jaccard’s index of similarity». In: *Systematic biology* 45.3 (1996), pp. 380–385.
- [62] Vijay Raghavan, Peter Bollmann, and Gwang S Jung. «A critical investigation of recall and precision as measures of retrieval system performance». In: *ACM Transactions on Information Systems (TOIS)* 7.3 (1989), pp. 205–229.
- [63] Jason Brownlee. *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*. <https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/>. (Accessed on 05/17/2020). Aug. 2018.
- [64] Nathalie Japkowicz and Shaju Stephen. «The class imbalance problem: A systematic study». In: *Intelligent data analysis* 6.5 (2002), pp. 429–449.
- [65] Chuanqi Tan et al. «A survey on deep transfer learning». In: *International conference on artificial neural networks*. Springer. 2018, pp. 270–279.