DEEP LEARNING NETWORKS WITH ADAPTIVE ATTENTION MECHANISMS FOR MALARIA DETECTION IN THICK SMEAR BLOOD SAMPLES

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Adriana Mónica Renjifo Herrera

In partial fulfilment

of the requirements for the degree in

Telecommunications Technologies and Services Engineering

Advisor: Elisa Sayrol Clols

Barcelona, October 2022

Abstract

This project aims to develop an end-to end deep network to detect and count malaria parasites from thick smear blood sample images with object detectors. The dataset is provided by Institute of Research of Vall d'Hebron-Drassanes.

Malaria is a severe disease caused by parasites of the genus Plasmodium. [3] Paludism is the most common disease in resource-poor settings, with 241 million malaria cases in 2020.[4]

This project uses different objects detectors such as Faster-RCNN, SSD, RetinaNet and YOLO. The model with the best results is YOLO, so adaptive attention mechanisms are implemented in this model, such as Squeeze-and-Excitation (SE) and Convolution Block Attention Module (CBAM).

Revision history and approval record

Revision	Date	Purpose
0	25/05/2022	Document creation
1	12/10/2022	Document revision
2	18/10/2022	Document approved

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Adriana Mónica Renjifo Herrera	adriana.monica.renjifo@estudiantat.upc.edu
Elisa Sayrol Clols, Project Supervisor	elisa.sayrol@upc.edu

Written by:		Reviewed and approved by:	
Date	18/10/2022	Date	18/10/2022
Name	Adriana Mónica Renjifo Herrera	Name	Elisa Sayrol Clols
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

1.1. Statement of purpose

The purpose of this project is to develop an end-to end deep network to detect malaria parasites from thick smear blood sample images with object detectors with adaptive attention mechanisms and see the performance.

1.2. Requirements and specifications

Project requirements:

- Deep learning neural networks:
  - Faster R-CNN
  - RetinaNet
  - SSD
  - YOLO
- Libraries python:
  - PyTorch
  - Torchvision
- Adaptive attention mechanisms:
  - Squeeze and Excitation
  - Convolutional Block Attention Model
- Database
  - Images and labels (json and txt format)

1.3. Methods and procedures

This project is part of a larger project in which the BIOCOMSC, DTIM and GPI research groups of the UPC, in addition to the Probitas Foundation and the Vall d'Hebron Research Institute, whose general objective is to development of an effective, affordable and quality system for the automatic detection of malaria, in which they use the YOLO algorithm.

This project is also continuation of Josep Blazquez's thesis part of a series of works at University's Image Processing Group, of deep learning networks for malaria detection in thick smear blood samples using SSD, RetinaNet and Faster R-CNN. The model with the best performance is Faster-RCNN.

This project aims to integrate adaptive attention mechanisms to YOLO algorithm and Faster-RCNN. The database is from Drassanes Center tropical diseases (part of the Vall d'Hebron Hospital)

The experiments were done in python scripts and Google Collab, were done on servers of the UPC, CALCULA.

1.4. Work plan, Milestones and Gantt diagram

1.4.1. Workplan

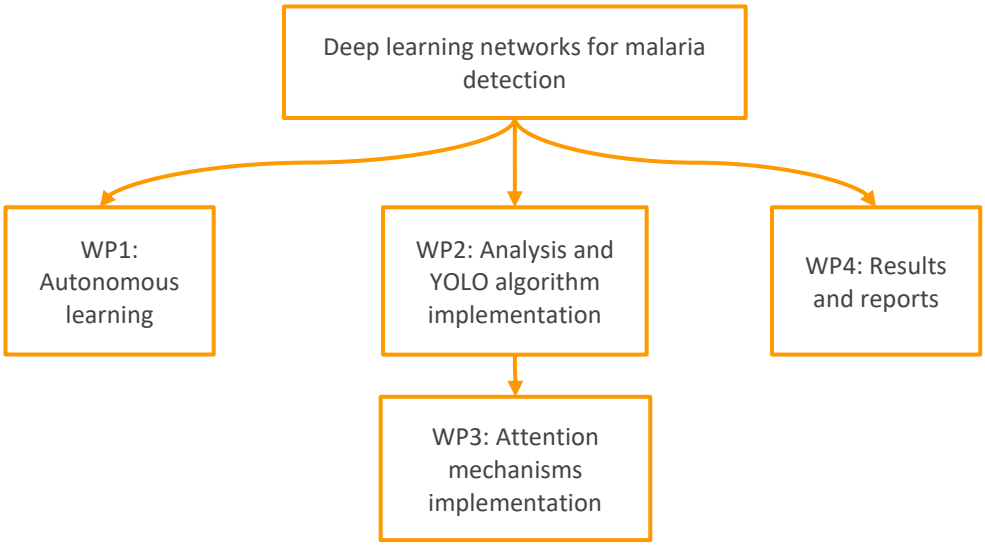


Figure 1 Work Plan

It was decided to separate the work into 5 Work Packages. In the first one, it was necessary to learn concepts about Deep Learning. The second package for the analysis of the previous work of Josep Balzquez (SSD, RetinaNet & Faster R-CNN) and implementation of YOLO algorithm. The third package for applying attention mechanisms to YOLO and Faster-RCNN. To finish, the fourth package for writing the thesis with the development, results and conclusions. In Figure 1.2 can be seen the Gantt followed.

Table 1 Work Packages

Project: Autonomous learning	WP ref: WP1
Major constituent: Learning	Sheet 1 of 4
Short description: - Learn concepts about Deep Learning. - Connecting to computing servers, CALCULA of the GPI with VPN. - Research YOLO algorithm and adaptive attention mechanisms	Start – end date: All semester
Internal task T1: Stanford Course CS231n Convolutional Neural Networks for Visual Recognition, on Youtube. Internal task T2: Labs of the Master course in Deep Learning for Artificial Intelligence from the UPC. Internal task T3: Connecting and configure the VM (virtual machine) Internal task T4: Research YOLO algorithm Internal task T5: Research adaptive attention mechanisms	Deliverables: Do the labs and watch the videos. And access correctly to the VM.

Project: Analysis and YOLO algorithm implementation	WP ref: WP2
Major constituent: Software and programming	Sheet 2 of 4
Short description:	Start date: 22/02/2022 End date: 13/05/2022

- Analysis of the previous work of Josep Balzquez (SSD, RetinaNet & Faster R-CNN) - Use the test dataset to see the performance and choose the better one for work later. - Implementation of the YOLO algorithm, train and test	
Internal task T1: Prepare the database and load Internal task T2: Understand the code Internal task T3: Execute the scripts Internal task T4: Analyze the hyperparameters Internal task T5: Test the algorithm for select the best option Internal task T6: Make modifications if necessary Internal task T7: Prepare YOLO algorithm Internal task T8: Train and test YOLO	Deliverables: The hyperparameters. Find the best model. Python scripts.

Project: Adaptive attention mechanisms implementation	WP ref: WP3
Major constituent: Software and programming	Sheet 3 of 4
Short description: Implement the adaptive attention mechanisms (AAM) in the model with highest performance (found previously) and also with the YOLO.	Start date: 13/05/2022 End date: 15/06/2022 Start date: 12/09/2022 End date: 9/10/2022
Internal task T1: Prepare AAM algorithms Internal task T2: Train and test AAM	Deliverables: Pytorch scripts

Project: Results and reports	WP ref: WP4
Major constituent: Analysis and write the documentation	Sheet 4 of 4
Short description: Write the Critical review and the Final Report document.	Start date T1: 9/04/2022 End date T1:14/04/2022 Start date T2:25/05/2022 End dateT2: 12/10/2022
Internal task T1: Critical review document Internal task T2: Final Report document	Deliverables: Reports

1.4.2. Milestone

WP#	Short title	Milestone / deliverable	Date (week)
1	Autonomous learning	Research	All semester
2	Analysis and YOLO algorithm	Model	09/05/2022
3	Adaptive attention mechanisms	Models with AAM	03/10/2022
4	Results and reports	Documents	12/10/2022

Table 2 Milestone

1.4.3. Gantt diagram

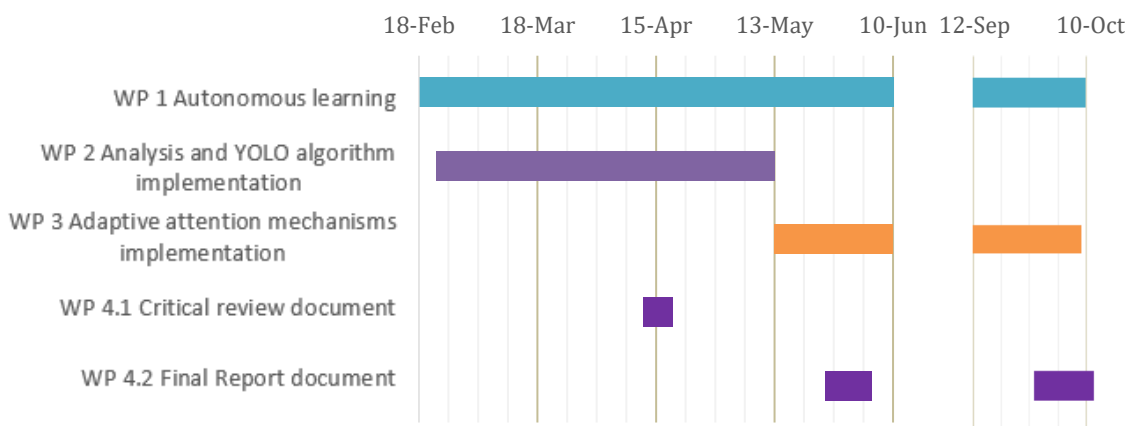


Figure 2 Gantt diagram

1.5. Deviations and Incidences

- The implementation of adaptive attention mechanisms in Faster-R CNN and YOLO requires a modification in the architecture, in the case of Faster-RCNN it is more complicated so I could not be completed.
- For reasons unrelated to the project, it has not been addressed the Bluetooth communication between the computer and the mobile, to transmit malaria images taken from the mobile through the microscope.

2. Malaria

Malaria is one of the most common infectious diseases worldwide. It is caused by parasites of the genus Plasmodium, which is transmitted to humans by a bite of an infected female mosquito of the species Anopheles [3].

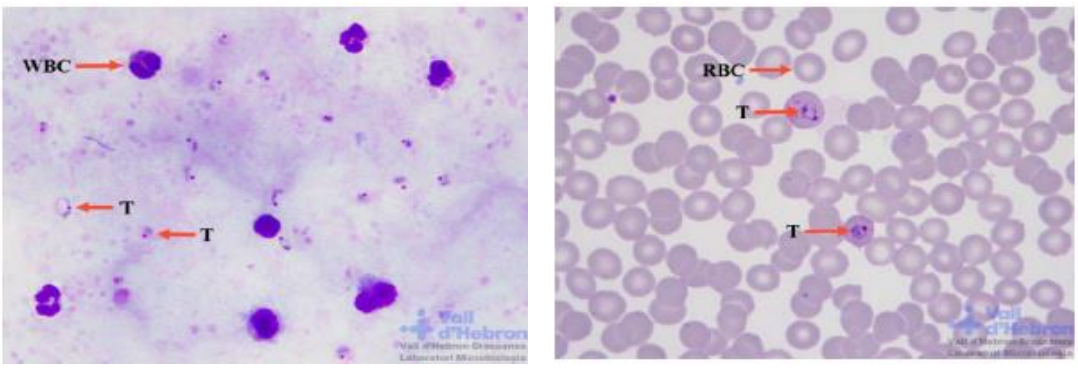
Malaria is a severe disease caused by parasites of the genus Plasmodium, which is transmitted to humans by a bite of an infected female mosquito of the species Anopheles. The World Health Organization carries out a malaria control program on a global scale, focusing on local strengthening of primary health care, early diagnosis of the disease, timely treatment, and disease prevention. Globally, the burden of malaria is lower than ten years ago [4].

Globally, there were an estimated 241 million malaria cases in 2020 in 85 malaria endemic countries (including the territory of French Guiana), increasing from 227 million in 2019 [4]. Malaria-related mortality has a very high association with poverty rates, and the disease is most prevalent in low and middle-income countries. [5].

The COVID-19 pandemic has resulted in unprecedented challenges to health systems worldwide, including the control of non-COVID-19 diseases. Malaria cases and deaths may increase due to the direct and indirect effects of the pandemic in malaria-endemic countries, particularly in sub-Saharan Africa [6].

2.1. Thick and thin blood smear

- **A thick blood smear** is a drop of blood on a glass slide. Thick blood smears are most useful for detecting the presence of parasites, because they examine a larger sample of blood. (Often there are few parasites in the blood at the time the test is done.) [2]
- **A thin blood smear** is a drop of blood that is spread across a large area of the slide. Thin blood smears help doctors discover what species of malaria is causing the infection.[2]



(A)Thick blood smear *(B) Thin blood smear*
White Blood Cell nuclei (WBC), Red Blood Cell (RBC) and Trophozoites (T)

Figure 3 Examples of thick and thin blood smear [1]

Microscopic visualization of thin blood smears permits to distinguish the species of Plasmodium, due to the morphology of erythrocytes and the distinctive features depending on the type of

specimen infection. Thick blood smears are more efficient and have more sensitivity than thin blood smears [7].

2.2. Diagnosis methods

Several techniques are available and used to detect directly or indirectly the presence of malaria parasites in blood.

- Microscopic examination
- Quantitative Buffy Coat (QBC)
- Rapid Diagnostic Tests (RDTs)
- Polymerase Chain Reaction (PCR)
- LAMP

This thesis focuses on microscopic examination, this way we have a database of images of thick blood smear samples for deep learning models.

Comparative table of malaria diagnostic techniques with its main advantages and disadvantages. QBC: Quantitative Buffy Coat, RDTs: Rapid Diagnostic Tests, PCR: Polymerase Chain Reaction, LAMP: Loop-Mediated Isothermal Amplification. [1]

Diagnostic technique	Advantages	Disadvantages
Microscopic examination	Availability, low cost, parasite level calculations, specie identification.	Expert personal requirement, difficult to reproduce.
Quantitative Buffy Coat (QBC)	Fast preparation, high sensitivity.	Expert personal requirement, fluorescence microscopy, specific instrumentation.
Rapid Diagnostic Tests (RDTs)	Fast preparation, easy handling, low cost, specie identification.	False-negative and false-positive results. Mainly, low parasite density, HRP2/3 gene deletions, prozone effect, low sensitivity with P.ovale and P.malariae infections.
PCR	High sensitivity and specificity, specie identification.	Specific instrumentation, difficult to implement in endemic areas and expensive.
LAMP	High sensitivity and specificity, specie identification, does not need thermocyclers.	Specific instrumentation and expensive.

Table 3 Comparative table of malaria diagnostic techniques with its main advantages and disadvantages. [1]

3. State of the art in object detection

3.1. Image processing techniques

Optical microscopy examination of blood smears is the "gold standard" technique for malaria diagnosis, although is a time-consuming method, depends on the experience of the microscopist and results are difficult to reproduce. New techniques based on digital imaging analysis by deep learning and artificial intelligence methods are a challenging alternative tool for the diagnosis of infectious diseases. Systems based on Convolutional Neural Networks for image malaria parasite detection emulate the microscopy visualization of an expert. Automation provides a faster and low-cost diagnosis, requiring less supervision. Smartphones are a suitable option for microscopic diagnosis, which permits image capture and software identification of parasites. In addition, image analysis techniques could be a fast and optimal solution for the diagnosis of malaria, tuberculosis, or Neglected Tropical Diseases in endemic areas with low resources. The implementation of automated diagnosis by using mobile phone applications and new digital imaging technologies in low-income areas is a challenge to achieve. Moreover, automating the movement and focusing of the samples in the microscope by hardware implementation would systemize the procedure. These new diagnosis tools will join the global effort to fight against the malaria pandemic.[1] State of the art of the technology used or applied in this thesis

3.2. Deep learning detection models

3.2.1. Faster RCNN

Faster R-CNN is a deep con deep convolutional network used for object detection, based two-stage detection. Faster R-CNN improves Fast R-CNN by utilising a region proposal network (RPN) with the CNN model. It is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position.[10]

Faster RCNN has two networks. The first is the RPN for generating region proposals and the second network for using these proposals to detect objects. The time cost of generating region proposals is much smaller in RPN than selective search, when RPN shares the most computation with the object detection network. The architecture is as follows.[17]

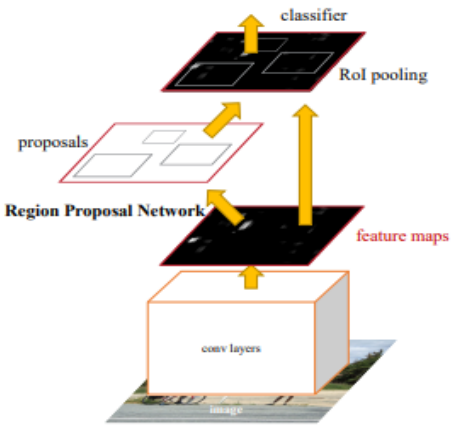


Figure 4 Faster R-CNN [10]

The anchor boxes play an important role in Faster R-CNN. It is a reference box of a specific scale and aspect ratio. By the default there are 9 anchors (3 scales x 3 aspect ratios) at a position of an image. Some of the anchor variations are shown in the next figure.[17]

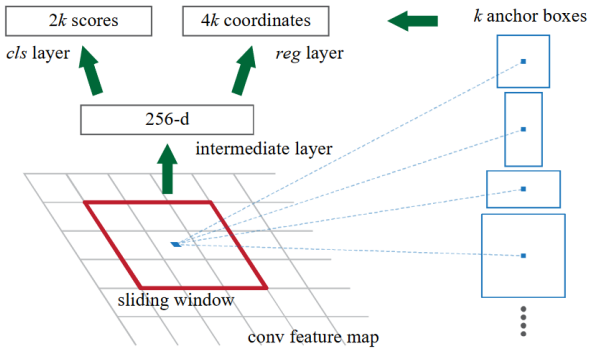


Figure 5 Region Proposal Network (RPN) [10]

3.2.2. RetinaNet

The highest accuracy object detectors to date are based on a two-stage approach popularized by R-CNN, where a classifier is applied to a sparse set of candidate object locations. In contrast, one-stage detectors that are applied over a regular, dense sampling of possible object locations have the potential to be faster and simpler but have trailed the accuracy of two-stage detectors thus far. The researchers discovered that the extreme foreground-background class imbalance encountered during training of dense detectors is the central cause. Loss focuses training on a sparse set of hard examples and prevents the vast number of easy negatives from overwhelming the detector during training. To evaluate the effective-ness of the loss, they design and train a simple density detector, RetinaNet.[12]

RetinaNet is a one-stage object detection model, unified network composed of a backbone network and two task-specific subnetworks. The backbone is responsible for computing a convolutional feature map over an entire input image and is an off-the-self convolutional network. The first subnet performs convolutional object classification on the backbone's output; the second subnet performs convolutional bounding box regression [12].

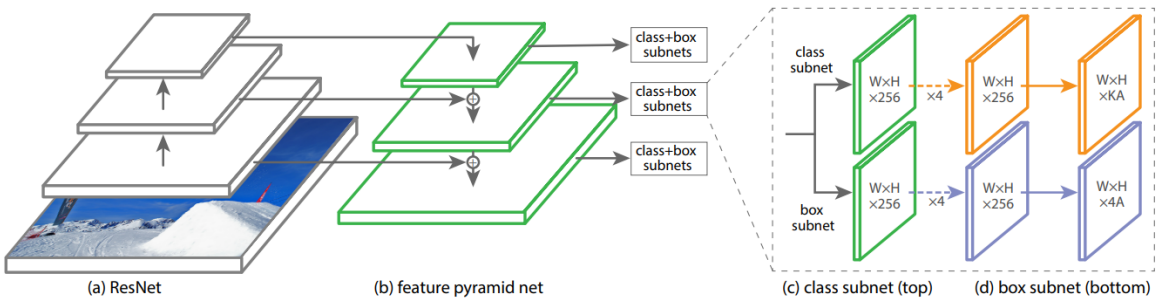


Figure 6 The one-stage RetinaNet network architecture [12]

The one-stage RetinaNet network architecture uses a Feature Pyramid Network (FPN) backbone on top of a feedforward ResNet architecture (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). The

network design is intentionally simple, which enables this work to focus on a novel focal loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN while running at faster speeds. [12]

3.2.3. SSD: Single Shot MultiBox Detector

The SSD is a single-stage detection. SSD discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes.2 [9]

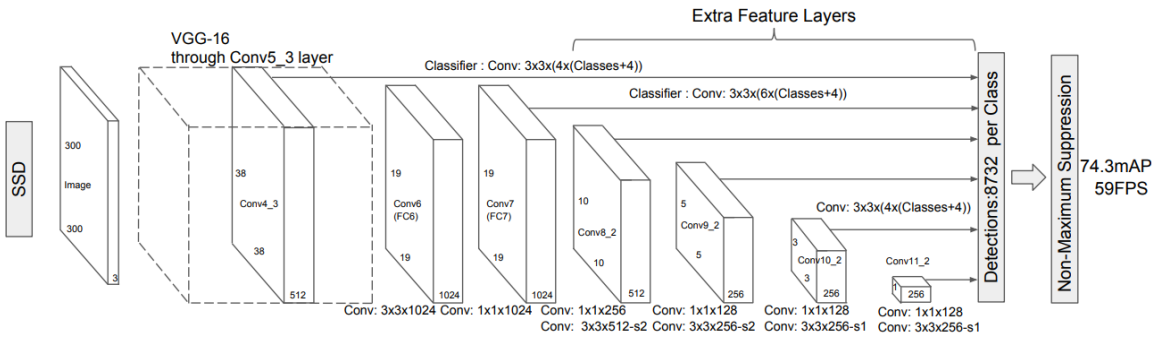


Figure 7 SSD architecture [9]

The early network layers are based on a standard architecture used for high quality image classification (truncated before any classification layers), called the base network. Then it is the auxiliary structure to the network to produce detections with the following key features [9]:

- Multi-scale feature maps for detection.
  Convolutional feature layers to the end of the truncated base network (backbone model). These layers decrease in size progressively and allow predictions of detections at multiple scales.
- Convolutional predictors for detection.
  Each added feature layer (or optionally an existing feature layer from the base network) can produce a fixed set of detection predictions using a set of convolutional filters. These are indicated on top of the SSD network architecture in Fig. 7. The bounding box offset output values are measured relative to a default box position relative to each feature map location
- Default boxes and aspect ratios.
  The default boundary boxes are equivalent to anchors in Faster R-CNN. The default bounding boxes are associated with every feature map cell at the top of the network. These default boxes tile the feature map in a convolutional manner such that the position of each box relative to its corresponding cell is fixed.

3.2.4. YOLO: You Only Look Once

YOLO is a single shot detector, which means that it only looks at the image once to make the prediction in the bounding boxes. YOLO frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts

bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.[8]

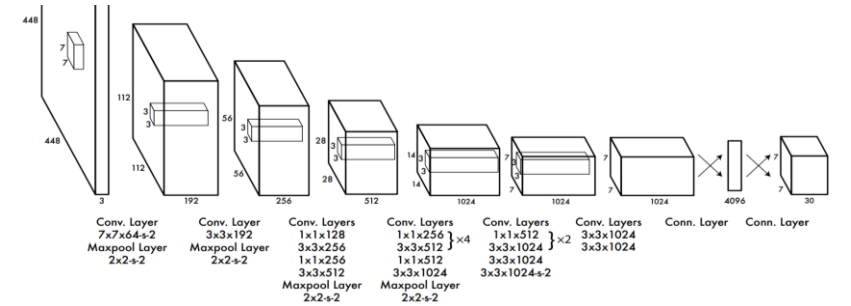


Figure 8 YOLO architecture [8]

The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates.

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. Since the model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. The model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU.[8]

3.3. Adaptive Attention Mechanisms

The attention mechanism is inspired by the human vision, which tends to selectively focus on parts of information and ignore others. In deep learning, attention mechanisms have been widely used in conjunction with the existing neural network models to assign different weights to the different parts of the model by which more critical feature representations *(what* and/or *where*) can be obtained to optimize the model.

3.3.1. Channel Attention

The channel attention focuses on “what” is meaningful given an input image and it is usually in the higher layers since there is abundant semantic information but less positional information. With object of different sizes, may have a negative impact on other objects. To compute the channel attention efficiently, the spatial dimension of the input feature map is squeezed. [15]

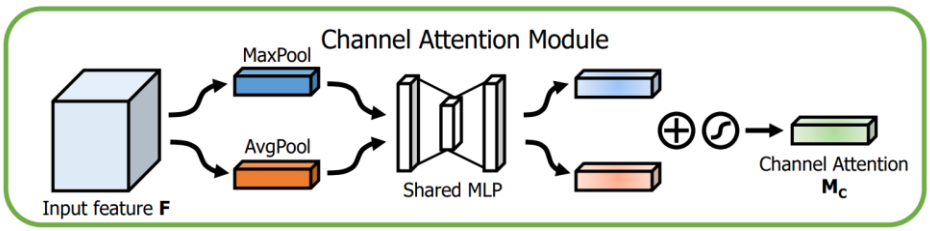


Figure 9 Channel Attention Module [15]

Channel attention performs feature recalibration with respect to the channel dimension and utilizes both max-pooling outputs and average-pooling outputs with a shared network. With the average-pooling we can identify discriminative regions of an object and with the max-pooling it

can be more beneficial for small object. The squeeze-and-excitation module uses average-pooling [14] and convolutional-block-attention-module uses both [15].

We first aggregate spatial information by using both average-pooling and max-pooling operations, generating two different spatial context descriptors: $\boldsymbol{F}_{avg}^{\boldsymbol{C}}$ and $\boldsymbol{F}_{max}^{\boldsymbol{C}}$. Both descriptors are then forwarded to a shared network to produce our channel attention map $M_C \in \mathbb{R}^{C\times1\times1}$.The shared network is composed of multi-layer perceptron (MLP) with one hidden layer. To reduce parameter overhead, the hidden activation size is set to $\mathbb{R}^{C/r\times1\times1}$, where r is the reduction ratio. After the shared network is applied to each descriptor, we merge the output feature vectors using element-wise summation.[15]

The channel attention is computed as:

$$M_c(\boldsymbol{F}) = \sigma\left(MLP\left(AvgPool(\boldsymbol{F})\right) + MLP\left(MaxPool(\boldsymbol{F})\right)\right)$$
$$= \sigma\left(\boldsymbol{W_1}\left(\boldsymbol{W_o}\left(\boldsymbol{F}_{avg}^{\boldsymbol{C}}\right)\right) + \boldsymbol{W_1}\left(\boldsymbol{W_o}\left(\boldsymbol{F}_{max}^{\boldsymbol{C}}\right)\right)\right) \quad (1)$$

where σ denotes the sigmoid function, $\boldsymbol{W_o} \in \mathbb{R}^{C/r\times C}$, and $\boldsymbol{W_1} \in \mathbb{R}^{C\times C/r}$.

3.3.2. Spatial Attention

The spatial attention focuses on “where” as an informative part. Lower layers of a network contain abundant positional information but less semantic information, we only apply spatial attention to several lower layers of a detection network, is complementary to channel attention. [15]

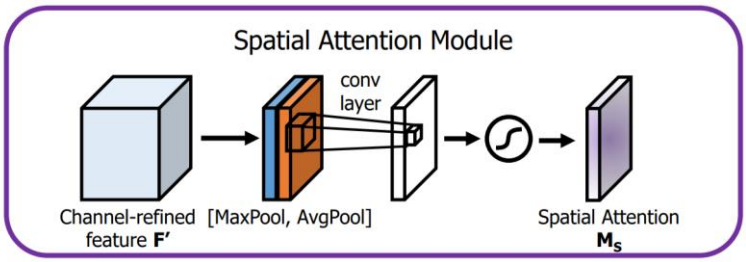


Figure 10 Spatial Attention Module [15]

We first apply average-pooling and max-pooling operations along the channel axis and concatenate them to generate an efficient feature descriptor. Applying pooling operations along the channel axis is shown to be effective in highlighting informative regions. On the concatenated feature descriptor, we apply a convolution layer to generate a spatial attention map $M_S(\mathbf{F}) \in \mathbb{R}^{H\times W}$. which encodes where to emphasize or suppress. We describe the detailed operation below. We aggregate channel information of a feature map by using two pooling operations, generating two 2D maps: $\boldsymbol{F}_{avg}^{\boldsymbol{S}} \in \mathbb{R}^{1\times H\times W}$ and $\boldsymbol{F}_{max}^{\boldsymbol{S}} \in \mathbb{R}^{1\times H\times W}$ [15].

The spatial attention is computed as:

$$M_S(\boldsymbol{F}) = \sigma\left(f^{7\times7}([AvgPool(\boldsymbol{F}); MaxPool(\boldsymbol{F})])\right) = \sigma\left(f^{7\times7}\left([\boldsymbol{F}_{avg}^{\boldsymbol{S}}; \boldsymbol{F}_{max}^{\boldsymbol{S}}]\right)\right) \quad (2)$$

where σ denotes the sigmoid function and $f^{7\times7}$ represents a convolution operation with the filter size of 7×7.

3.3.3. SE: Squeeze and Excitation

Squeeze-and-Excitation Networks introduce a building block for CNNs that improves channel interdependencies at almost no computational cost. It can be easily added to existing architectures. SE add parameters to each channel of a convolutional block so that the network can adaptively adjust the weighting of each feature map.[18]

SE module allows the network to perform feature recalibration whereby the network learns to use global information to selectively emphasize informative features and suppress less useful ones [14]. The block is connected after the final convolutional layer, to the block before the residue is added to the jump connection. The intuition behind this is to keep the jump connection branch as clean as possible to facilitate identity learning.

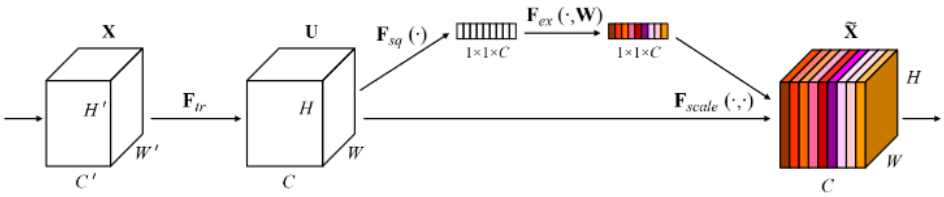


Figure 11 Squeeze and Excitation block [14]

First, a feature transformation (such as a convolution operation) is performed on the input image X to get features U. Next, we perform a squeeze operation to get a single value for each channel of output U. After, we perform an excitation operation on the output of the squeeze operation to get per-channel weights.

Finally, once we have the per-channel weights, the final output of the block is obtained by rescaling the feature map U with these activations. [19]

The role this operation performs at different depths differs throughout the network. In earlier layers, it excites informative features in a class-agnostic manner, strengthening the shared low-level representations. In later layers, the SE blocks become increasingly specialised, and respond to different inputs in a highly class-specific manner. As a consequence, the benefits of the feature recalibration performed by SE blocks can be accumulated through the network. [14]

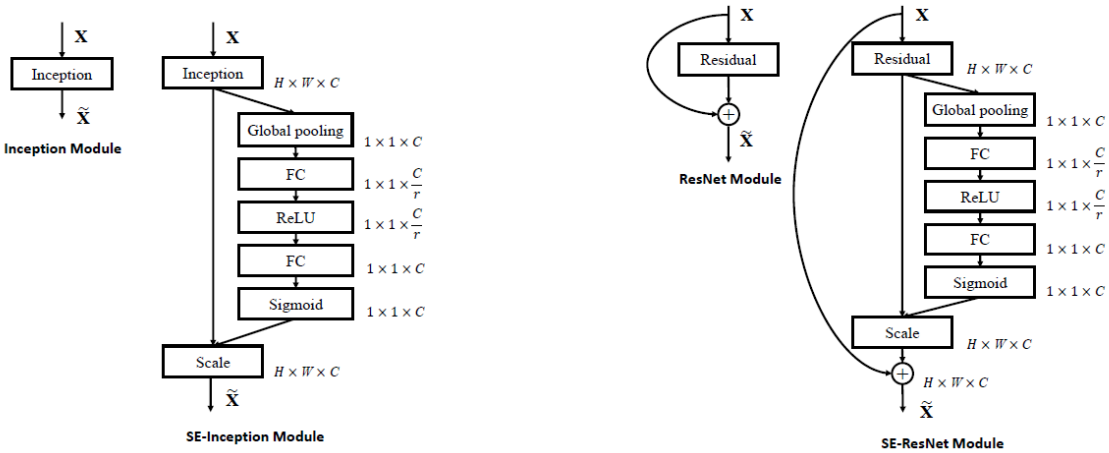


Figure 12 SE-Inception and SE-ResNet module [14]

To integrate SE blocks with existing state-of-art architectures (Fig. 13). The performance improvements produced by SE units are fairly robust to their location, provided that they are applied prior to branch aggregation. [14]

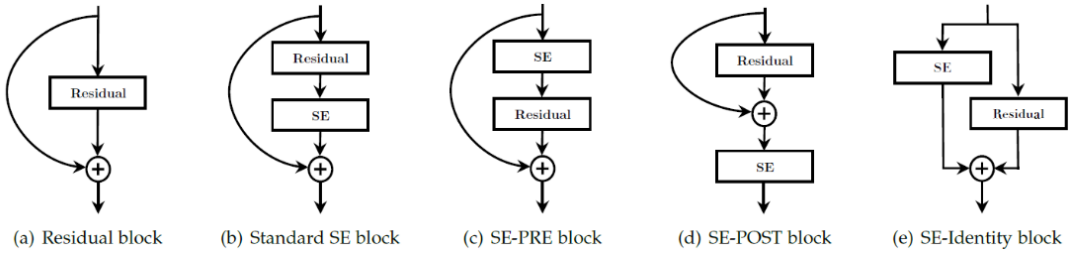


Figure 13 SE block integration designs [14]

In Fig. 13 We can see different integration strategies for the SE block. The standard SE block is applied right after the final convolutional layer of the architecture, in this case of a Residual Network, right before the merging of the skip connection. The SE-PRE configuration was constructed by placing the SE block at the start of the block, before the first convolutional layer, while SE-POST did the opposite by placing it at the end of the block (after the merging of the skip connection). Finally, the SE-Identity block applied the SE-module in the skip connection branch itself, parallel to the main block, and is added to the final output as a normal residual.

3.3.4. CBAM: Convolutional Block Attention Module

The CBAM has two sequential sub-modules: Channel Attention and Spatial Attention. The intermediate feature map is adaptively refined through the CBAM module at every convolutional block of deep networks. [15]

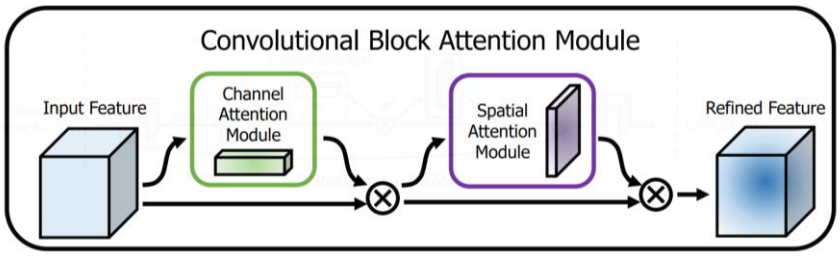


Figure 14 CBAM module [15]

CBAM takes in a tensor containing the feature maps from the previous convolutional layer and first refines it by applying channel attention. Subsequently this refined tensor is passed to spatial attention, thus resulting in the output refined feature maps.

Two modules can be placed in a parallel or sequential manner. It is found that the sequential arrangement gives a better result than a parallel arrangement. For the arrangement of the sequential process, experimental result shows that the channel-first order is slightly better than the spatial-first. In Fig. 15 shows the exact position of our module when integrated within a ResBlock. We apply CBAM on the convolution outputs in each block.[15]

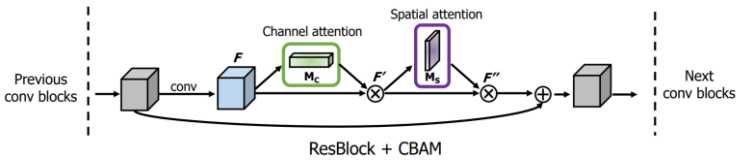


Figure 15 CBAM integrated with a ResBlock in ResNet [15]

4. Methodology / project development

4.1. Dataset and annotations

The database is composed of a total of 2570 images with their respective labels in JSON format and was provided by the Institute of Research of Vall d'Hebron-Drassanes.
It is necessary to split randomly the database one time into three:

- 80% for training dataset, 2057 images.
- 15% for validation dataset, 384 images.
- 5% for testing dataset, 129 images.

It is verified that all the images have the file of the labels, in this project 3 images were found without labels that are not considered for the rest of the rpoject.

The JSON format is like the Fig. 16.


```
{"py/object": "model.model_data.ModelData",
"uuid": "fdfc313d0959c615b7caf2909ce311b6",
"filePath": "C:/Users/Pictures/Fotos_App/MAIG 2021/26-05-2021/image1431gota72.jpg",
"fileName": "image1431gota72.jpg", "healthFacility": "Unitat de Salut Internacional, Drassanes-Vall d'Hebron",
"microscopist": "IMAGING Annotator",
"preparation":
    {"py/object": "model.preparation.Preparation",
                "waterType": 1, "usesGiemsa": 1, "giemsaFP": 1, "usesPbs": 1, "usesAlcohol": 1, "reusesSlides": false},
"microscopeQuality":
    {"py/object": "model.microscope_quality.MicroscopeQuality", "isDamaged": false, "magnification": 1000},

"annotations": [
    {"py/object": "model.annotation.Annotation",
        "id": 1, "posX": 1411, "posY": 158, "width": 130, "height": 137,
        "color": 4284871417, "disease": "Unclear/Undefined", "label": "Leukocyte"},
    {"py/object": "model.annotation.Annotation",
        "id": 2, "posX": 531, "posY": 70, "width": 124, "height": 136,
        "color": 4286740359, "disease": "Unclear/Undefined", "label": "Leukocyte"},
    {"py/object": "model.annotation.Annotation",
        "id": 3, "posX": 12, "posY": 106, "width": 143, "height": 141,
        "color": 4286592253, "disease": "Unclear/Undefined", "label": "Leukocyte"},
    {"py/object": "model.annotation.Annotation",
        "id": 4, "posX": 513, "posY": 301, "width": 121, "height": 137,
        "color": 4290076784, "disease": "Unclear/Undefined", "label": "Leukocyte"},
    {"py/object": "model.annotation.Annotation",
        "id": 5, "posX": 505, "posY": 468, "width": 128, "height": 120,
        "color": 4289392508, "disease": "Unclear/Undefined", "label": "Leukocyte"},
    {"py/object": "model.annotation.Annotation",
        "id": 6, "posX": 553, "posY": 581, "width": 139, "height": 146,
        "color": 4283955548, "disease": "Unclear/Undefined", "label": "Leukocyte"},
    {"py/object": "model.annotation.Annotation",
        "id": 7, "posX": 441, "posY": 930, "width": 80, "height": 95,
        "color": 4288475478, "disease": "+malaria mature trophozoite plasmodium spp", "label": "Parasite"}],

"bloodSmears": "Thick Blood Smears", "appVersion": "1.0.0", "device": "",
"createdOn": "26/05/2021 11:54:43", "lastModified": "26/05/2021 12:02:17",
"createdBy": "Carles", "exists_in_drive": true, "pending_changes": false}
```


Figure 16 JSON annotation example

The JSON file contains different information, such as the name, the quality of the microscope, annotations, among others.
We use the 'uuid' that matches the image name to match it with its respective JSON file. In the 'annotations' section, we have the list of the annotations, each with their bounding box coordinates, the disease, and the label.

Table 4 shows the annotations of the labels in all three datasets and in total. The 'Leukocyte', 'Uncertain' and 'Artifact' label have 'Unclear/Undefined' as disease, instead the 'Parasite' has 6 types of malaria as disease.

Annotation - label	Training	Validation	Test	Total
Leukocyte	7532	1376	464	9372
Parasite	16415	2717	863	19995
Uncertain	3472	549	207	4228
Artifact	2202	355	125	2682
TOTAL	**29621**	**4997**	**1659**	**36277**

Table 4 Annotations - label

The Table 5 shows the annotations of each disease in all three datasets, and in total.

Annotation - disease	Training	Validation	Test	Total
Malaria small trophozoite plasmodium spp	15168	2506	773	18447
Malaria mature trophozoite plasmodium spp	780	137	52	969
Malaria schizont plasmodium spp	289	40	24	353
Malaria trophozoite plasmodium spp	37	6	3	46
Malaria gametocyte plasmodium falciparum	113	22	7	142
Malaria gametocyte plasmodium spp	28	6	4	38
Unclear/Undefined.	13206	2280	796	16282
Total malaria	**16415**	**2717**	**863**	**19995**
TOTAL	**29621**	**4997**	**1659**	**36277**

Table 5 Annotations - disease

There are not enough annotations of all types of diseases. Therefore, it has been decided to summarize the labels and disease in 4 classes:

0. Background
1. Leukocyte
2. Malaria trophozoite
3. Malaria mature trophozoite

In the background goes 'Uncertain' and 'Artifact' labels because there are irrelevant for the training. The Leukocytes are all white blood cells annotations. Malaria trophozoite and Malaria mature trophozoite because there are the most relevant parasites.

This project also needs the labels in txt format for the YOLO model, in the following format:

`Class  X_center  Y_center  width_yolo  height_yolo`

Where:

$$X_{center} = \frac{\text{posX} + \frac{\text{width}}{2}}{\text{img}_{\text{width}}} \qquad Y_{center} = \frac{\text{posY} + \frac{\text{height}}{2}}{\text{img}_{\text{height}}} \tag{3}$$

$$\text{width}_{\text{yolo}} = \frac{\text{width}}{\text{img}_{\text{width}}} \qquad \text{height}_{\text{yolo}} = \frac{\text{height}}{\text{img}_{\text{height}}} \tag{4}$$

$$\text{img}_{\text{height}}, \text{img}_{\text{width}}, \text{img}_{\text{channels}} = image.shape \tag{5}$$

An example of the labels in txt format:

```
0 0.922 0.189 0.081 0.114
0 0.371 0.115 0.077 0.113
0 0.052 0.147 0.089 0.117
0 0.358 0.308 0.076 0.114
0 0.356 0.44 0.08 0.1
0 0.389 0.545 0.087 0.122
2 0.301 0.815 0.05 0.079
```

Figure 17 txt annotation example

For the YOLO implementation, we consider the following classes:

0. Leukocyte
1. Malaria trophozoite
2. Malaria mature trophozoite

4.2. Adaptative attention Mechanism

To add the SE and CBAM modules to YOLO, we must modify the architecture and add the code to make it work since it is not integrated in YOLOv5. We must modify three files: common.py, yolo.py and the yaml file of the architecture and each one should have a different yaml file of the architecture.

4.2.1. SE - YOLO

In this project we follow some the integration proposals of Squeeze-and-Excitation, Fig 13, like the standard SE block, SE pre block, SE post block, and a combination of the pre and post block. And two others according to some examples and papers [16]. In total, 6 different architectures were made:

1. Standard SE: SE after the residual and before the concatenation.
2. SE pre block: SE before the residual.
3. SE post block: SE after the concatenation.
4. SE pre and post block: Combination between pre and post block
5. SE other: according to paper [16] about an Attention Mechanism for Small Object Detection on Satellite Images.
6. SE backbone: SE at the end of the backbone.

In Fig 18 we can see the example of the architecture that must be followed for the SE backbone, all the architecture diagrams are in the Appendices A, as well the code implementation.

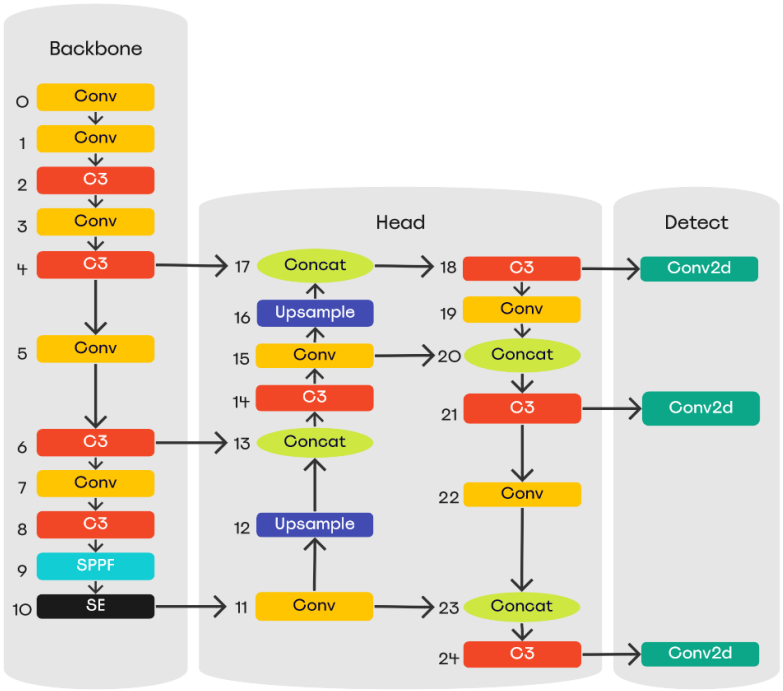


Figure 18 SE backbone architecture

4.2.2. CBAM - YOLO

The details of the implementations of the code files are in the Appendices B.

CBAM module needs to go after the convolutions, C3 block, to facilitate the architecture and avoid channels errors, a C3_CBAM block is created, this block will only be in the backbone, see Fig 19.

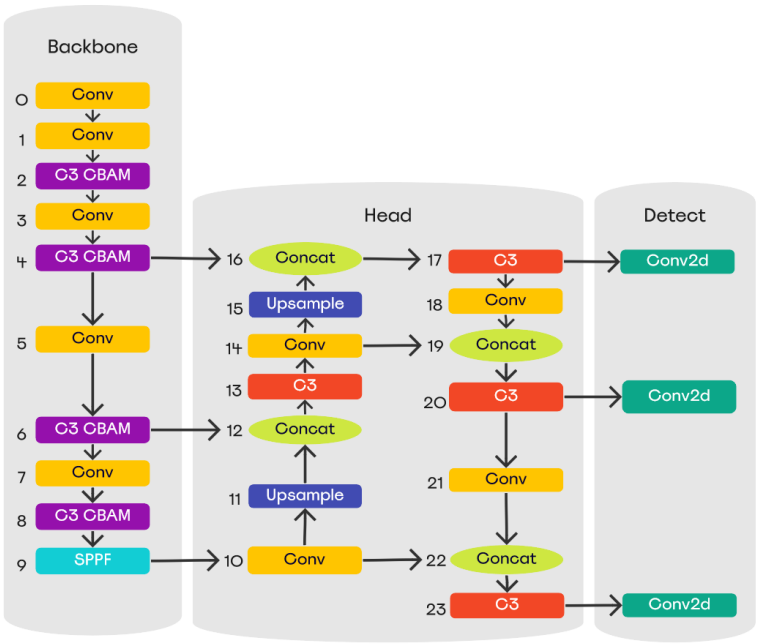


Figure 19 CBAM architecture

4.3. Model Evaluation

F_score and mean Average Precision (mAP) are adopted as the evaluation criteria. F_score finds the most optimal confidence score threshold where precision and recall give the highest F1 score. The F1 score calculates the balance between precision and recall. If the F1 score is high, precision and recall are high, and vice versa. The precision P (Equation (6)) and the recall R (Equation (7)) are defined as:

$$P = \frac{TP}{TP + FP} \tag{6}$$

$$R = \frac{TP}{TP + FN} \tag{7}$$

$$F_{measure} = 2 * \frac{P \times R}{P + R} \tag{8}$$

where TP, FP and FN are the numbers of true positive cases, false positive cases and false negative cases, respectively. A detection map will be considered to be a TP when the IoU exceeds the given threshold (e.g., 0.5). The IoU is the overlap ratio between the predicted box and the ground truth, defined as:

$$IoU = \frac{\text{area(A)} \cap \text{area(B)}}{\text{area(A)} \cup \text{area(B)}} \tag{9}$$

where A and B are the ground truth and the predicted box.

5. Results

5.1. Faster RCNN, RetinaNet and SSD results

For development of the models, is necessary use a pre-trained backbone on the COCO dataset, other wise a large database would be needed.

This project uses the hyperparameters of Josep Blazquez's thesis:

Model	Optim.	Lr	Momen.	Weight decay
Faster RCNN	SDG	1,0E-03	0,900	5,00E-04
Retina Net	SDG	1,0E-03	0,900	5,00E-03
SSD	SDG	1,0E-04	0,900	5,00E-04

Table 6 Hyperparameters from Josep Blazquez's work

In Appendices C are the graphs of the losses and metrics of each model.

This project has 218 more images in the database than Josep Blazquez's project, but we still come to the same conclusions. Faster RCNN works better than the SSD and RetinaNet. And we reach this result with an epoch of 120 for faster RCNN and SSD, instead for RetinaNet we need an epoch of 60, RetinaNet achieve very fast compared to the other models.

In the following table show the summary of the metrics, for validation and testing.

Validation					
Model	**Precision**	**Recall**	**F-score**	**mAP@0.50**	**mAP@0.50:0.95**
Faster RCNN	0.8753	0.9331	0.9033	0.9194	0.6152
RetinaNet	0.9369	0.8155	0.8720	0.9180	0.6101
SSD	0.9501	0.4789	0.6368	0.8491	0.5351
Test					
Model	**Precision**	**Recall**	**F-score**	**mAP@0.50**	**mAP@0.50:0.95**
Faster RCNN	0.8913	0.9638	0.9261	0.9412	0.6390
RetinaNet	0.9407	0.8719	0.9050	0.9489	0.6513
SSD	0.9562	0.5599	0.7063	0.9133	0.5842

Table 7 Summary of metrics, Faster RCNN, Retina Net and SSD

If we compare with Josep's work, we will see that the metrics improves, since we have more images, and we have to take into account that the distribution of the images is not exactly the same, for example, an image that he had for training dataset I could have it in the test dataset, that also influences the result of the metrics. The comparation is in the following table:

Test						
Model	**Josep F-score**	**F-score**	**Josep mAP@.50**	**mAP@.50**	**Josep mAP@.50:.95**	**mAP@.50:.95**
Faster RCNN	0.9120	**0.9261**	0.9320	**0.9412**	0.6290	**0.6390**
RetinaNet	0.9030	**0.9050**	0.9370	**0.9489**	0.6480	**0.6513**
SSD	**0.7150**	0.7063	0.8810	**0.9133**	0.5690	**0.5842**

Table 8 Comparison of metrics with the results of Josep Blazquez

5.2. YOLO results

For development of the model, is necessary use a pre-trained weight, yolo5x.pt. since we do not have a large dataset.

We have three options for the optimizer, SDG, adamW and adam, in the image below we can see that works much better with SDG optimizer.

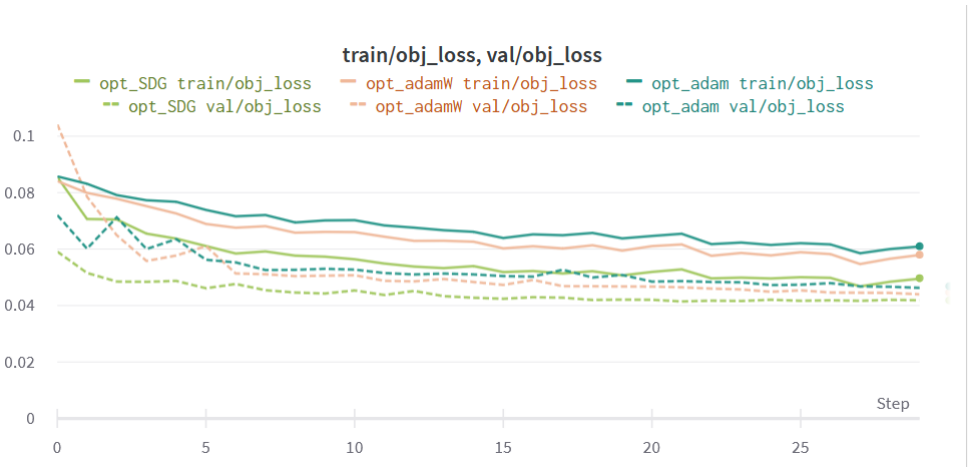


Figure 20 YOLO - optimizer loss

For the hyperparameters, YOLO has by default (hyp.scratch-low) :

- $lr0 = 0.01$ the initial learning rate
- $lrf = 0.01$ the final OneCycleLR learning rate ($lr0 * lrf$)
- $momentum = 0.937$
- $weight\_decay = 5e - 4$, among others.

But there is another type of configuration, integrate in YOLO , hyp.scratch-med, that we can use $lrf = 0.1$. In the Fig 21, **we ca**n see that we have a better performance with the $lrf = 0.1$ of the hyp.scratch-med.yaml file.

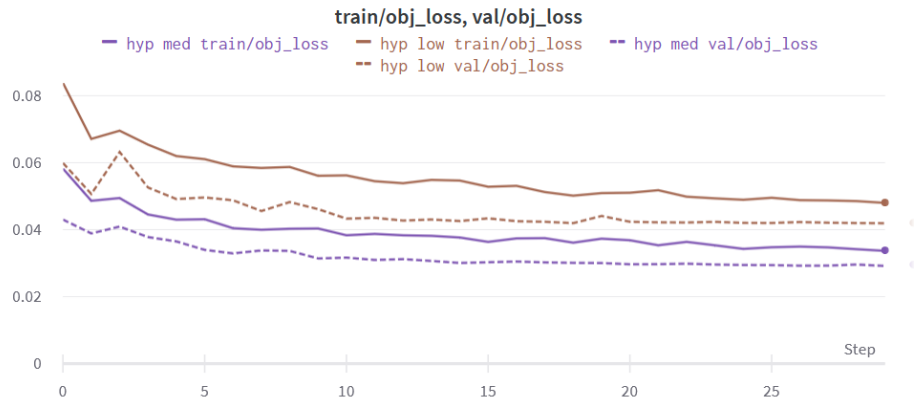


Figure 21 Hyperparameter search

We train the model until 30 epochs with a batch of 8 images, with the hyperparameters from YOLO.

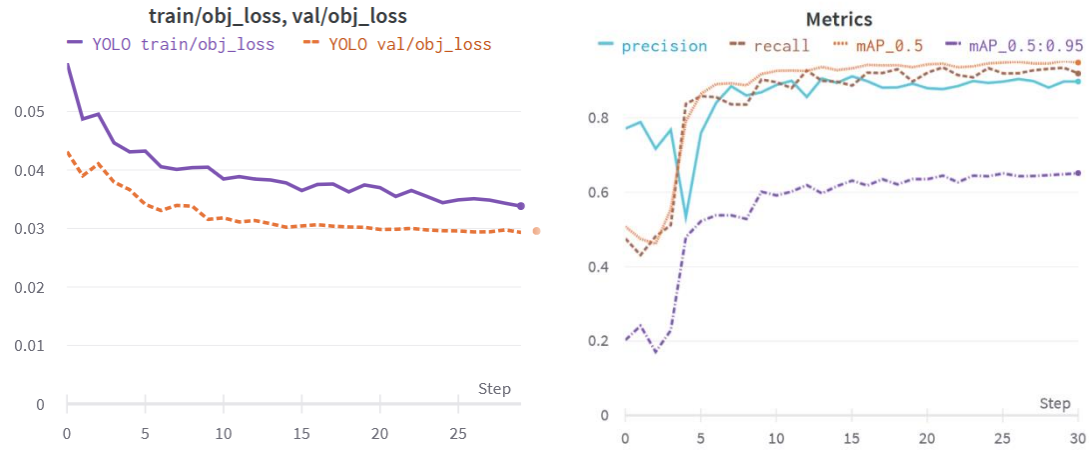


Figure 22 YOLO - Loss and metrics

YOLO provides two models: the latest model and the best model. In this experiment the best model is found in the epoch 25. The values of the metrics for each model:

Validation						
Epoch	**Dataset**	**Precision**	**Recall**	**F-score**	**mAP@0.50**	**mAP@0.50:0.95**
Best -25	**Validation**	0.8975	**0.9198**	**0.9085**	**0.9490**	0.6513
Last-30	**Validation**	**0.8978**	0.9192	0.9084	**0.9490**	**0.6516**
Test						
Epoch	**Dataset**	**Precision**	**Recall**	**F-score**	**mAP@0.50**	**mAP@0.50:0.95**
Best -25	**Test**	0.9210	**0.9350**	**0.9279**	**0.9440**	**0.6910**
Last-30	**Test**	**0.9370**	0.9020	0.9192	0.9320	0.6820

Figure 23 YOLO - metrics numbers

We can see that most of the higher values of F-score are in the best model, now we can compare it with the other models in the next table:

Validation					
Model	**Precision**	**Recall**	**F-score**	**mAP@0.50**	**mAP@0.50:0.95**
Faster RCNN	0.8753	**0.9331**	0.9033	0.9194	0.6152
RetinaNet	0.9369	0.8155	0.8720	0.9180	0.6101
SSD	**0.9501**	0.4789	0.6368	0.8491	0.5351
YOLO	0.8975	0.9198	**0.9085**	**0.9490**	**0.6513**
Test					
Model	**Precision**	**Recall**	**F-score**	**mAP@0.50**	**mAP@0.50:0.95**
Faster RCNN	0.8913	**0.9638**	0.9261	0.9412	0.6390
RetinaNet	0.9407	0.8719	0.9050	**0.9489**	0.6513
SSD	**0.9562**	0.5599	0.7063	0.9133	0.5842
YOLO	0.9210	0.9350	**0.9279**	0.9440	**0.6910**

Table 9 Test - Comparison of model metrics

YOLO is the one who has the best performance with the highest F-score of 0,9279.

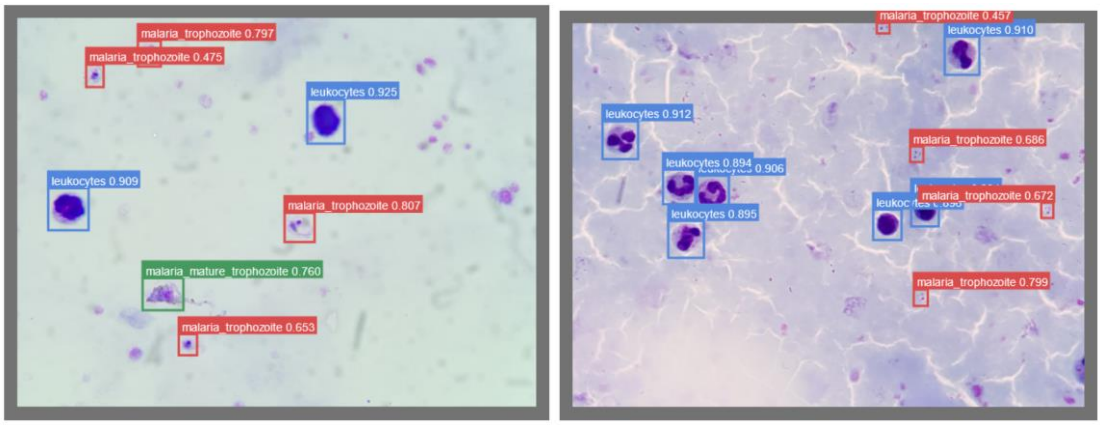


Figure 24 Examples of detections

5.3. Attention Module results

By adding the SE and CBAM, the architecture is changed, therefore it needs more training than in the case of the normal YOLO. For this reason, training experiments are carried out up to 60 epochs, and we use the test database for detection in the 2 models that YOLO gives us, the best model, and the last model.

In total there are 7 experiments, 6 are from SE and 1 from CBAM. Putting them all together in a graph would be complicated, that is why the table is very detailed.

- 1st column: The experiment name.
- 2nd column: If it is the validation or testing database.
- 3rd column: YOLO gives us a 'best' and a 'last' file, so we do tests with each of them.
- 4th column: The number of the epoch.
- And the other columns the metrics.

In the tables [10] [11], those values that are marked with green, is because those values are equal or greater than the metric of the YOLO in the epoch 25, the one in the table in red colour.

Those marked in yellow only have a difference of 0.005 with the value we are comparing, this way we know that it is a value that is very close.

And both tables are sorted by F-score values, from higher to lower.

In the table [10] of validation, we see that the models with best performance are:

- **SE other** with F-score of 0,9157 for the best epoch 59, and 0,9155 for the last epoch.
- **SE backbone** with F-score of 0,9121 for the best epoch 58, and 0,9120 for the last epoch
- **CBAM** with F-score of 0,9112 for the best epoch 55, and last epoch.
- **SE standard** with F-score of 0,9101 for the last epoch and 0,9098 for the best epoch 59.

In the case of the test evaluation, in table [11], do not happen the same thing, the best performance is the basic YOLO architecture, but the most near to this is the **CBAM** architecture with F-score of 0, 9259 in the best epoch 55.

Validation							
Model	Epoch		Precision	Recall	F-score	mAP@.50	mAP@.50:.95
SE other	best	59	0,9118	0,9196	0,9157	0,9481	0,6490
	last	60	0,9139	0,9171	0,9155	0,9479	0,6486
SE backbone	best	58	0,9018	0,9226	0,9121	0,9431	0,6428
	last	60	0,9015	0,9227	0,9120	0,9414	0,6421
CBAM	last	60	0,8858	0,9380	0,9112	0,9463	0,6555
	best	55	0,8857	0,9382	0,9112	0,9460	0,6552
SE standard	last	60	0,9094	0,9108	0,9101	0,9432	0,6506
	best	59	0,9101	0,9096	0,9098	0,9432	0,6502
YOLO	best	25	0,8975	0,9198	0,9085	0,9490	0,6513
SE post	last	60	0,9066	0,9100	0,9083	0,9497	0,6467
	best	49	0,9064	0,9101	0,9083	0,9497	0,6468
SE pre	best	59	0,9338	0,8839	0,9082	0,9494	0,6422
	last	60	0,9326	0,8848	0,9081	0,9494	0,6424
SE pre and post	last	60	0,8896	0,9204	0,9047	0,9468	0,6426
	best	58	0,8898	0,9201	0,9047	0,9466	0,6427

Table 10 Compare validation - Attention modules

Test							
Model	Epoch		Precision	Recall	F-score	mAP@.50	mAP@.50:.95
YOLO	best	25	0,9210	0,9350	0,9279	0,9440	0,6910
CBAM	best	55	0,9350	0,9170	0,9259	0,9420	0,6940
SE other	best	59	0,9040	0,9380	0,9207	0,9450	0,6960
	last	60	0,9040	0,9380	0,9207	0,9450	0,6960
SE pre and post	last	60	0,9390	0,9020	0,9201	0,9350	0,6810
CBAM	last	60	0,9060	0,9320	0,9188	0,9480	0,7010
SE standard	best	59	0,9160	0,9160	0,9160	0,9370	0,6870
	last	60	0,9160	0,9160	0,9160	0,9370	0,6870
SE backbone	best	58	0,9200	0,9060	0,9129	0,9390	0,6870
	last	60	0,9070	0,9130	0,9100	0,9380	0,6750
SE post	best	49	0,8930	0,9250	0,9087	0,9330	0,6800
	last	60	0,9130	0,8970	0,9049	0,9280	0,6790
SE pre and post	best	58	0,8950	0,9130	0,9039	0,9280	0,6740
SE pre	best	59	0,8920	0,8930	0,8925	0,9230	0,6670
	last	60	0,8920	0,8930	0,8925	0,9230	0,6670

Table 11Compare test - Attention modules

6. Budget

Consider the following for the budget:

- All the software used for this project, like Visual Studio Code, are open Source.
- The project has taken 460 hours, and a salary as a junior engineering of 9€/h.
- Medium-high range laptop to be able to work comfortably and quickly. The approximate cost of a laptop with these characteristics is 1000 €.
- The project requires many hours of training for the models and testing; therefore, we consider a virtual machine with a 16 GB GPU for 4 months, 45€/month.
- The dataset is a resource created by the Institute of Research of Vall dHebron-Drassanes and It is not included.

The total cost can be summarized as follows:

Item	Total
Software	0 €
Junior engineer	4320 €
Laptop	1000 €
VM GPU 16GB	185 €
Total Project Cost Estimate	**5325 €**

Table 12 Project budget

The total cost estimate is 5325€.

7. Conclusions and future development

After carrying the experiments and obtaining the results of the different models and architectures, we can say that *the Convolutional Block Attention Module* in the backbone of the YOLO architecture provides the best performance, reaching an F-score of 0,9257 and mAP 0.5 of 0,9420.

Some approaches of future work would be:

- Combine different attention modules, like the SE with CBAM for a better performance.
- Add transformers into the architecture.
- Increase the database with open-source dataset.

Bibliography:

[1] Advances and challenges in automated malaria diagnosis using digital 1 microscopy imaging with artificial intelligence tools – Carles Rubio Maturana, Allisson Dantas de Oliveira, Sergi Nadal Francesch, Besim Bilalli, Francesc Zarzuela Serrat, Mateu Espasa Soley, Elena Sulleiro Igual, Mercedes Bosch, Anna Veiga Lluch, Alberto Abelló, Daniel López-Codina, Tomàs Pumarola Suñé, Elisa Sayrol Clols, Joan Joseph Munné.

[2] HealthLink BC, thick and thin blood smear malaria. [Link]

[3] Jasminka Talapko, Ivana Škrlec, Tamara Alebi´c, Melita Juki´c and Aleksandar V˘cev. Malaria: The Past and the Present. [Link]

[4] World Health Organization WHO. World malaria report 2021. [Link]

[5] Minghui Ren. Greater political commitment needed to eliminate malaria [Link]

[6] Anna-Katharina Heuschen, Guangyu Lu, Oliver Razum, Alhassan Abdul-Mumin4 Osman Sankoh, Lorenz von Seidlein, Umberto D'Alessandro and Olaf Müller. Public health-relevant consequences of the COVID-19 pandemic on malaria in sub-Saharan Africa: a scoping review [Link]

[7] Wangai, L. N. et al. (2011) 'Sensitivity of microscopy compared to molecular diagnosis of P. 860 Falciparum: Implications on malaria treatment in epidemic areas in Kenya', African Journal of 861 Infectious Diseases, 5(1), pp. 1–6. doi: 10.4314/ajid.v5i1.66504.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. 2016. arXiv:1506.02640v5 [Link]

[9] Wei Liu, Dragomir Anguelov2, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. 2016. arXiv:1512.02325v5 [Link]

[10] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2016. [Link]

[11] Ross Girshick. Fast R-CNN. 2015. arXiv:1504.08083v2 [Link]

[12] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollar. Focal Loss for Dense Object Detection. 2018. arXiv:1708.02002v2

[13] Wei Li, Kai Liu, Lizhe Zhang and Fei Cheng. object detection based on an adaptive attention mechanism. 2020 [Link]

[14] Jie Hu, Li Shen, Samuel Albanie, Gang Sun and Enhua Wu. Squeeze and Excitation Networks. 2019. [Link]

[15] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: Convolutional Block Attention Module. 2018. arXiv:1807.06521v2 [Link]

[16] Gong, H.; Mu, T.; Li, Q.; Dai, H.; Li, C.; He, Z.; Wang, W.; Han, F.; Tuniyazi, A.; Li, H.; et al. Swin-Transformer-Enabled YOLOv5 with Attention Mechanism for Small Object Detection on Satellite Images. Remote Sens. 2022, 14, 2861. https:// doi.org/10.3390/rs14122861

[17] Faster RCNN Explained. https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8

[18] Squeeze-and-Excitation Networks. https://towardsdatascience.com/squeeze-and-excitation-networks-9ef5e71eacd7

[19] Squeeze and Excitation Networks Explained with PyTorch Implementation. https://amaarora.github.io/2020/07/24/SeNet.html

[20] yolov5 added CBAM, SE, CA, ECA attention mechanism, pure code. https://www.codetd.com/en/article/13710152

[21] Attention Mechanisms in Computer Vision: CBAM. https://blog.paperspace.com/attention-mechanisms-in-computer-vision-cbam/

[22] Channel Attention and Squeeze-and-Excitation Networks (SENet). https://blog.paperspace.com/channel-attention-squeeze-and-excitation-networks/

[23] Attention Mechanisms in Computer Vision: CBAM. https://blog.paperspace.com/attention-mechanisms-in-computer-vision-cbam/

Appendices:

A. SE implementation

- Let's start with the modification of common.py, this file contains all the classes of the blocks that we can use in YOLO, we just have to add the SE class.

```
class SE_Layer(nn.Module):
    def __init__(self, c1, r=16):
        super(SE_Layer, self).__init__()
        # squeeze
        self.avgpool = nn.AdaptiveAvgPool2d(1)
        # excitation
        self.l1 = nn.Linear(c1, c1 // r, bias=False)
        self.relu = nn.ReLU(inplace=True)
        self.l2 = nn.Linear(c1 // r, c1, bias=False)
        self.sig = nn.Sigmoid()

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.avgpool(x).view(b, c)
        y = self.l1(y)
        y = self.relu(y)
        y = self.l2(y)
        y = self.sig(y)
        y = y.view(b, c, 1, 1)
        return x * y.expand_as(x)

    def forward(self, x):
        return self.relu(x + 3) / 6
```

Figure 25 SE class

- yolo.py: To modify the parse_model function, just add a condition for the SE module, otherwise we would have a problems with the channel numbers.

```
elif m is SE_Layer:  #modification: SE_Layer module
    channel=args[0]
    channel=make_divisible(channel*gw,8)if channel != no else channel
    args=[channel]
```

Figure 26 SE channel code

- yaml file: In this file we have the architecture, we must add the layers of SE.
  In this project 6 different architectures were made:
  1. SE backbone: SE at backbone.
  2. Standard SE: SE after the residual.
  3. SE pre block: SE before the residual.
  4. SE post block: SE after the concatenation.
  5. SE pre and post block: Combination between pre and post block
  6. SE other: according to paper [16].

Each architecture must have a different yaml file. In Fig 17 we can see the example of the architecture that must be followed for the SE backbone. For other architectures see below this section.

According to the backbone architecture, Fig. 17, only one SE layer should be added in layer 10, we also have to change from which layer it has to be concatenated (Concat block) and detect (Detect block), since now we have more layers.

The code of the yaml file would be the following:

```yaml
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
   [-1, 1, SE_Layer, [1024, 4]] # New SE layer
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 14

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 18 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 15], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 21 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 11], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 24 (P5/32-large)

   [[18, 21, 24], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Figure 27 SE backbone architecture code

Next, we show the other architectures with SE that are used in this project:

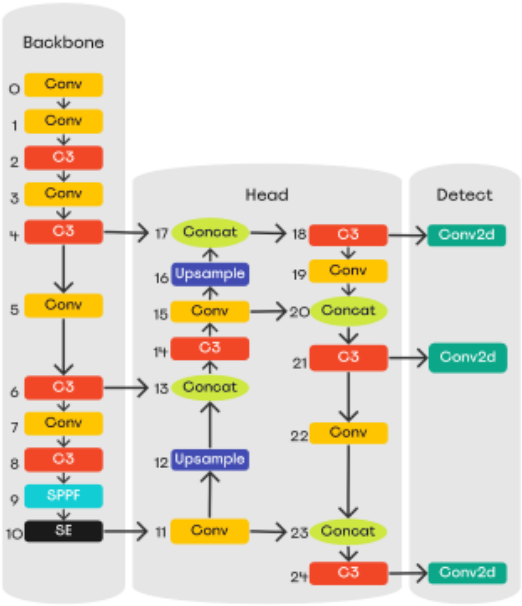


Figure 28 SE backbone

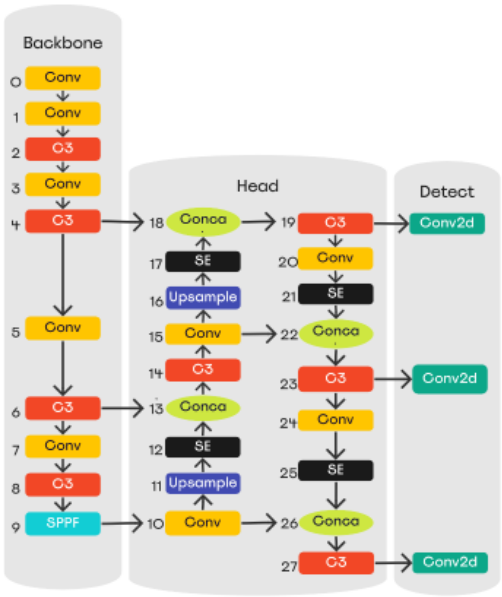


Figure 29 Standard SE

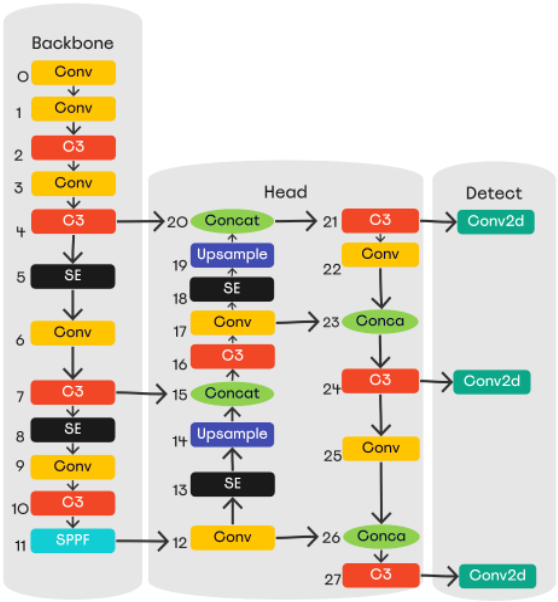


Figure 30 SE pre

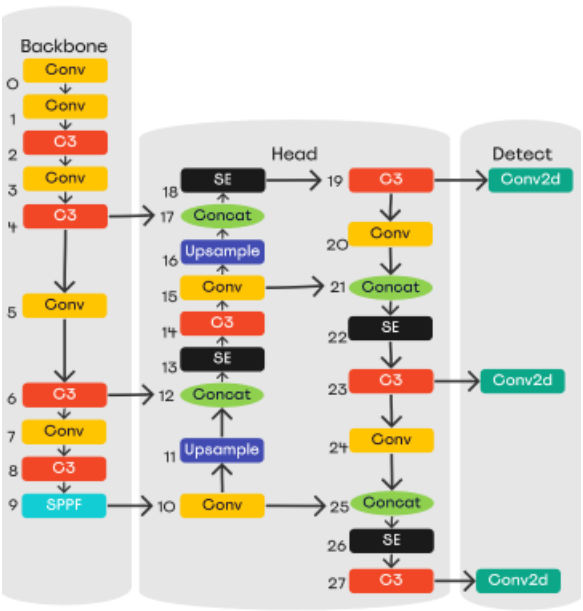


Figure 31 SE post

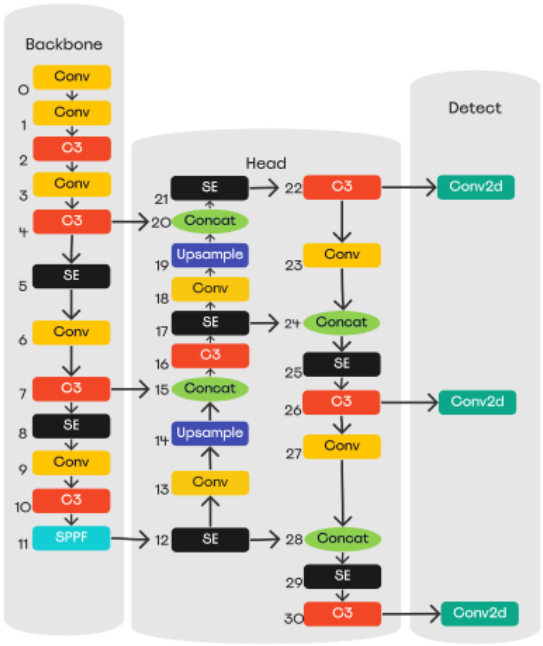


Figure 32 SE pre and post

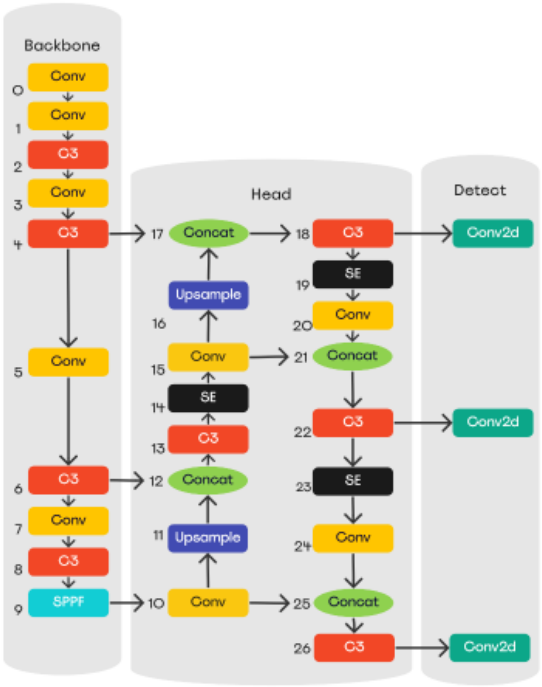


Figure 33 SE other [16]

B. CBAM implementation

For the implementation of CBAM, a procedure similar to SE is followed, with some differences.

- Modifications in common.py: To use the CBAM module we need to implement Channel and Spatial Attention, so in common.py three classes are added.

```python
class ChannelAttention(nn.Module):
    def __init__(self, in_planes, ratio=16):
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)

        self.f1 = nn.Conv2d(in_planes, in_planes // ratio, 1, bias=False)
        self.relu = nn.ReLU()
        self.f2 = nn.Conv2d(in_planes // ratio, in_planes, 1, bias=False)

        #  Write two , Sequential containers can also be used
        # self.sharedMLP = nn.Sequential(
        # nn.Conv2d(in_planes, in_planes // ratio, 1, bias=False), nn.ReLU(),
        # nn.Conv2d(in_planes // rotio, in_planes, 1, bias=False))

        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = self.f2(self.relu(self.f1(self.avg_pool(x))))
        max_out = self.f2(self.relu(self.f1(self.max_pool(x))))
        out = self.sigmoid(avg_out + max_out)
        return out
```

Figure 34 Channel Attention class

```python
class SpatialAttention(nn.Module):
    def __init__(self, kernel_size=7):
        super(SpatialAttention, self).__init__()

        assert kernel_size in (3, 7), 'kernel size must be 3 or 7'
        padding = 3 if kernel_size == 7 else 1

        self.conv = nn.Conv2d(2, 1, kernel_size, padding=padding, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = torch.mean(x, dim=1, keepdim=True)
        max_out, _ = torch.max(x, dim=1, keepdim=True)
        x = torch.cat([avg_out, max_out], dim=1)
        x = self.conv(x)
        return self.sigmoid(x)
```

Figure 35 Spatial Attention class

```python
class CBAM(nn.Module):
    # ch_in, number, shortcut, groups, expansion
    def __init__(self, c, ratio=16, kernel_size=7):
        super(CBAM, self).__init__()
        self.channel_attention = ChannelAttention(c, ratio)
        self.spatial_attention = SpatialAttention(kernel_size)
    def forward(self, x):
        out = self.channel_attention(x) * x
        out = self.spatial_attention(out) * out
        return out
```

Figure 36 CBAM class

As CBAM has to go after the convolutions, specifically after the YOLO C3 block, we can have a block in which everything is already together, for this we create a C3_CBAM class as in Fig 32

```
class C3_CBAM(nn.Module):
    # CSP Bottleneck with 3 convolutions
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):
        # ch_in, ch_out, number, shortcut, groups, expansion
        super(C3_CBAM, self).__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c1, c_, 1, 1)
        self.cv3 = Conv(2 * c_, c2, 1)
        self.m = nn.Sequential(*[Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)])
        #CBAM
        self.channel_attention = ChannelAttention(c2, 16)
        self.spatial_attention = SpatialAttention(7)

        # self.m = nn.Sequential(*[CrossConv(c_, c_, 3, 1, g, 1.0, shortcut) for _ in range(n)])

    def forward(self, x):
        x = self.cv3(torch.cat((self.m(self.cv1(x)), self.cv2(x)), 1))
        out = self.channel_attention(x) * x
        #print('outchannels:{}'.format(out.shape))
        out = self.spatial_attention(out) * out
        return out
```

Figure 37 CBAM with C3 block

- yolo.py: To modify the parse_mode function, just add the class name to the condition like the others blocks, no need to change the channel numbers.

```
n = n_ = max(round(n * gd), 1) if n > 1 else n  # depth gain
if m in (Conv, GhostConv, Bottleneck, GhostBottleneck, SPP, SPPF, DWConv, MixConv2d, Focus, CrossConv,
         BottleneckCSP, C3, C3TR, C3SPP, C3Ghost, CBAM, C3_CBAM):  #modification: CBAM module
    c1, c2 = ch[f], args[0]
    if c2 != no:  # if not output
        c2 = make_divisible(c2 * gw, 8)

    args = [c1, c2, *args[1:]]
    if m in [BottleneckCSP, C3, C3TR, C3Ghost, CBAM, C3_CBAM]: #modification: CBAM module
        args.insert(2, n)  # number of repeats
        n = 1
```

Figure 38 CBAM channel code

- yaml file: In the case of the CBAM, we just have to change the name of the C3 block, to C3_CBAM in the blocks of the backbone, see the architecture diagram in Fig. 19 and the code in the next figure.

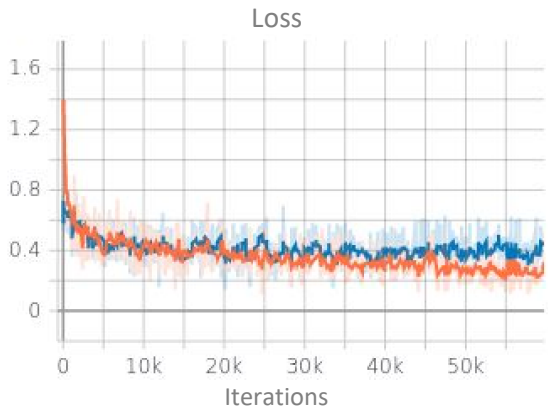
```
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3_CBAM, [128]],       # CBAM
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3_CBAM, [256]],       # CBAM
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3_CBAM, [512]],      # CBAM
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3_CBAM, [1024]],  # CBAM
   [-1, 1, SPPF, [1024, 5]]  # 9
  ]
```

Figure 39 CBAM architecture code

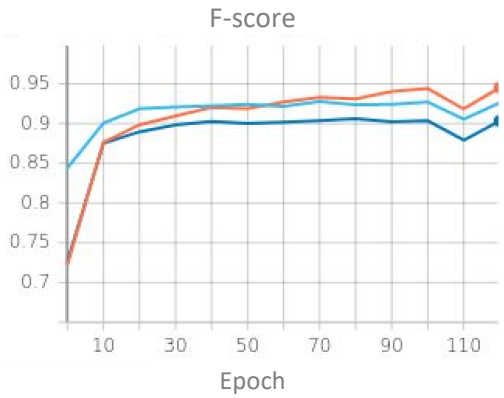
C. Faster RCNN, RetinaNet and SSD

Loss graph and metrics of the three models, Faster RCNN, Retina Net and SSD.

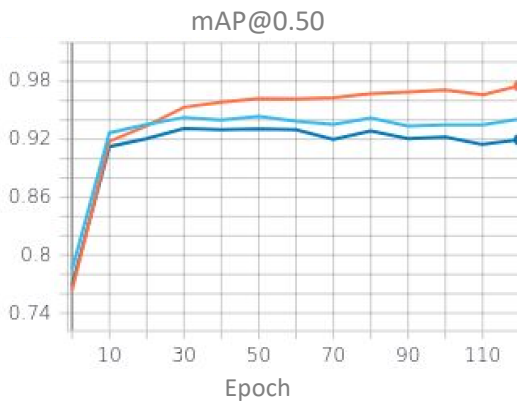
Faster RCNN



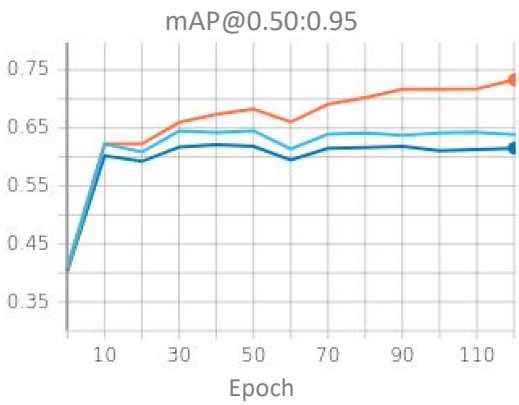
(a) Loss with smoothing



(b) F-score with 0.62 conf. threshold



(c) PASCAL VOC mAP



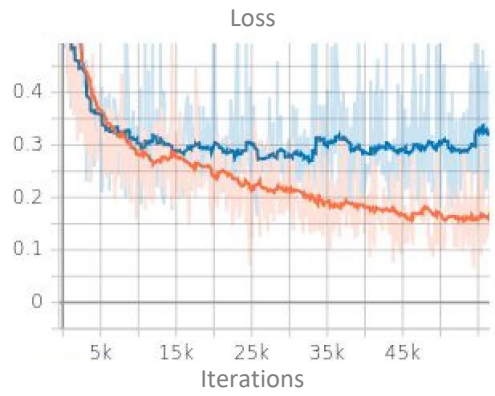
(d) COCO mAP

Figure 40 Faster RCNN – Loss and metrics
Orange: training, Blue: validation, Red: test

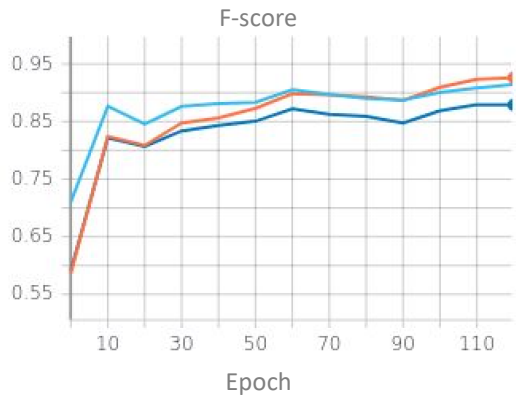
Faster RCNN	Precision	Recall	F-score	mAP@0.50	mAP@0.50:0.95
Validation	0.8753	0.9331	0.9033	0.9194	0.6152
Test	0.8913	0.9638	0.9261	0.9412	0.6390

Table 13 Faster RCNN - metric numbers

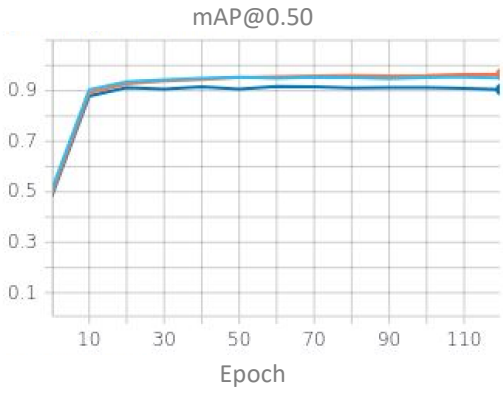
Retina Net



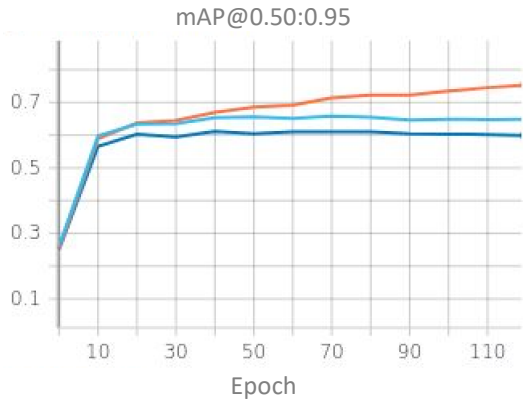
(a) Loss with smoothing



(b) F-score with 0.62 conf. threshold



(c) PASCAL VOC mAP



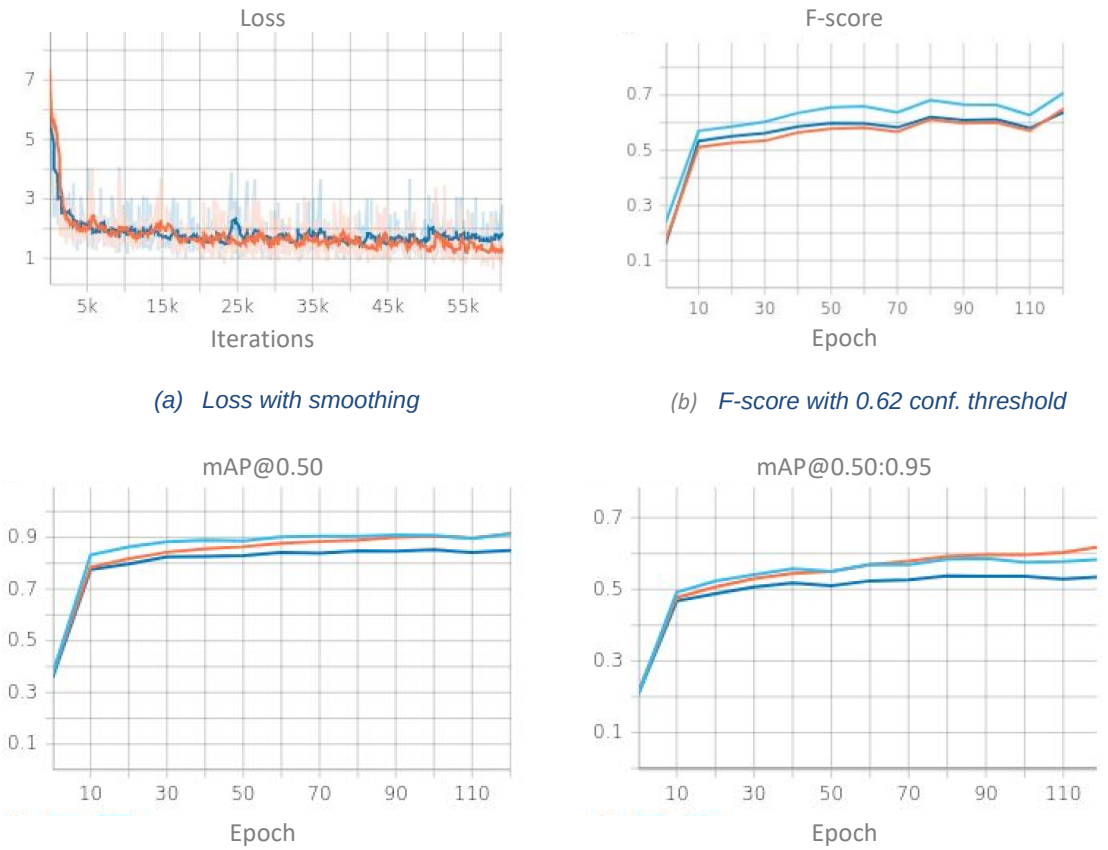
(d) COCO mAP

Figure 41 Retinanet - Loss and metrics
Orange: training, Blue: validation, Light blue: test

RetinaNet	Precision	Recall	F-score	mAP@0.50	mAP@0.50:0.95
Validation	0.9369	0.8155	0.8720	0.9180	0.6101
Test	0.9407	0.8719	0.9050	0.9489	0.6513

Table 14 Retina Net - metric numbers

SSD



(a) Loss with smoothing

(b) F-score with 0.62 conf. threshold

(c) PASCAL VOC mAP

(d) COCO mAP

Figure 42 SSD - Loss and metrics
Orange: training, Blue: validation, Light blue: test

SSD	Precision	Recall	F-score	mAP@0.50	mAP@0.50:0.95
Validation	0.9501	0.4789	0.6368	0.8491	0.5351
Test	0.9562	0.5599	0.7063	0.9133	0.5842

Table 15 SSD - metric numbers

Glossary

Abbreviations

BIOCOMSC	Computational Biology and Complex Systems, research group in Universitat Politècnica de Catalunya.
DTIM	Database Technologies and Information Management Group, research group in Universitat Politècnica de Catalunya.
ETSETB	Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona.
FN	False negative
FP	False positive
GPI	Video and Image Processing group at Signal Theory and Communications Department in Universitat Politècnica de Catalunya.
GPU	Graphics Processing Unit
GT	Ground truth
IoU	Intersection over Union
JSON	JavaScript Object Notation
mAP	mean Average Precision
RCNN	Region based convolutional neural network
RPN	Region Proposal Network
SSD	Single Shoot MulitBox Detector
TN	True negative
TP	True positive
UPC	Universitat Politècnica de Catalunya
VHIR	Vall d'Hebron Research Institute
YOLO	You Only Look One