COMPARATIVE ANALYSIS OF DEEP LEARNING FRAMEWORKS FOR AUTOMATED FRUIT DETECTION IN SMART AGRICULTURE

José Luis SANDOVAL ALAGUNA

Master's Thesis

Department of Computer Engineering

Programme in Computer Software

Supervisor: Prof. Dr. Serkan GÜNAL

Eskişehir

Eskişehir Technical University

Institute of Graduate Programs

August 2020

II. ABSTRACT

COMPARATIVE ANALYSIS OF DEEP LEARNING FRAMEWORKS FOR AUTOMATED FRUIT DETECTION IN SMART AGRICULTURE

José Luis SANDOVAL ALAGUNA

Department of Computer Engineering

Programme in Computer Software

Eskişehir Technical University, Institute of Graduate Programs, August 2020

Supervisor: Prof. Dr. Serkan GÜNAL

Recently, deep learning-based object detection approaches have come into prominence to automatically determine the fruits on trees for smart agriculture. However, the suitability of various frameworks to different fruit types has not been adequately analyzed in such studies. Taking this into consideration, in this thesis, the performance of different deep learning frameworks for object detection in detecting different fruit types has been comprehensively analyzed. For this purpose, RetinaNet, YOLOv3, and YOLOv4, which are among the highest performing deep learning frameworks for object detection according to the current literature, and 5 different datasets for mango, almond, grape, and apple images were used. After a through experimental analysis, it has been observed that RetinaNet is the best framework in terms of detection accuracy and training and detection time. Also, it has been found that the factors such as atmospheric conditions, properties of the object, and time of day while constituting the image datasets have great impact on the detection performance. Finally, it has been noted that documenting the conditions of the experiments facilitates the reproducibility of the experiments and comparison between different frameworks and datasets.

Keywords: Smart Agriculture, Deep Learning, Computer Vision, Object Detection, Fruit Detection.

III. ÖZET

AKILLI TARIMDA OTOMATIK MEYVE TESPİTİ İÇİN DERİN ÖĞRENME ÇERÇEVELERİNİN KARŞILAŞTIRMALI ANALİZİ

José Luis SANDOVAL ALAGUNA

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Yazılımı Bilim Dalı

Eskişehir Teknik Üniversitesi, Lisansüstü Eğitim Enstitüsü, Ağustos 2020

Danışman: Prof. Dr. Serkan GÜNAL

Son zamanlarda, akıllı tarım için ağaçlardaki meyveleri otomatik olarak belirlemeye yönelik derin öğrenme tabanlı nesne algılama yaklaşımları ön plana çıkmıştır. Bununla birlikte, çeşitli çerçevelerin farklı meyve türlerine uygunluğu bu tür çalışmalarda yeterince analiz edilmemiştir. Bunu göz önünde bulundurarak, bu tezde, farklı meyve türlerini tespit etmek için nesne tespitine yönelik farklı derin öğrenme çerçevelerinin performansı kapsamlı bir şekilde analiz edilmiştir. Bu amaçla, nesne tespiti için güncel literatüre göre en yüksek performanslı derin öğrenme çerçeveleri arasında yer alan RetinaNet, YOLOv3 ve YOLOv4 çerçeveleri ile mango, badem, üzüm ve elma görselleri için 5 farklı veri seti kullanılmıştır. Kapsamlı bir deneysel analiz sonunda, tespit doğruluğu, eğitim süresi ve tespit süresi açısından RetinaNet'in en iyi çerçeve olduğu gözlemlenmiştir. Ayrıca görüntü veri setlerini oluştururken atmosferik koşullar, nesnenin özellikleri ve günün saati gibi faktörlerin algılama performansı üzerinde büyük etkisi olduğu bulunmuştur. Son olarak, deneylerin koşullarının belgelenmesinin, deneylerin tekrarlanabilirliğini ve farklı çerçeveler ve veri kümeleri arasında karşılaştırmayı kolaylaştırdığı belirtilmiştir.

Anahtar Sözcükler: Akıllı Tarım, Derin Öğrenme, Bilgisayarla Görme, Nesne Tespiti, Meyve Tespiti.

IV. ACKNOWLEDGEMENTS

I want to thank my thesis supervisor, Prof. Dr. Serkan GÜNAL, who provided me with an interesting topic for development and was supervising me, giving me recommendations and suggestions, and other useful activities throughout the development of this dissertation.

I also want to thank Jonathan Hui, Software Architect & Lead at InnoTrekker, his writings on deep learning and the different concepts involved in it, as well as the dissertations on different deep learning frameworks helped me a lot to understand how they work and to be able to evaluate them for this study.

José Luis SANDOVAL ALAGUNA

17/08/2020

V. STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES

I hereby truthfully declare that this thesis is an original work prepared by me; that I have behaved in accordance with the scientific ethical principles and rules throughout the stages of preparation, data collection, analysis and presentation of my work; that I have cited the sources of all the data and information that could be obtained within the scope of this study, and included these sources in the references section; and that this study has been scanned for plagiarism with "scientific plagiarism detection program" used by Eskişehir Technical University, and that "it does not have any plagiarism" whatsoever. I also declare that, if a case contrary to my declaration is detected in my work at any time, I hereby express my consent to all the ethical and legal consequences that are involved.

José Luis SANDOVAL ALAGUNA

VI. CONTENTS

Page

VII. LIST OF TABLES

Page

VIII. LIST OF FIGURES

Page

IX. LIST OF IMAGES

Page

X. GLOSSARY OF SYMBOLS AND ABBREVIATIONS

IoU	:	Intersection over Union
CNN	:	Convolutional Neural Network
mAP	:	Media Average Precision

1. INTRODUCTION

This thesis aims to be a guide for future developments in the field of smart agriculture, knowing in advance which deep learning technique is the most suitable for the crop on which researcher wants to work can save time and effort that can be used to get the most out of the chosen framework; a careful study of the current state of art researching on the application of deep learning in the field of agriculture provides a general idea of the potential of current techniques and their suitability for each type of study subject that agriculture can present, and in this specific case, for the type of crop product in its different agricultural contexts.

1.1. Background

The field of agriculture has great importance throughout the world and also in Turkey; the increasing number of inhabitants in the world makes it imperative to cover the demand for food for a constantly growing population (Kitzes, 2008); this situation is negatively affected by the increasingly severe depopulation of the countryside, which contrasts with the growth of cities (McNicoll, 2002) (Perry, 2015), which are the ones that supply the highest demand for food globally, and which is the largest provider of food, and therefore, in the agricultural sector par excellence. Fortunately, with scientific development comes also technological development in the field of agriculture, fostered by the industrialization of the field, and what has enabled the productivity of the countryside to have increased significantly and independently of human labor; today it is possible to find miles of commercial technological solutions that facilitate and enhance agriculture (Schmitz, 2015).

For the 1990s, studies carried out in the field of agriculture intending to boost production while saving material resources led to the creation of the concept of "precision agriculture" (McBratney, 2005), which consists of the farm's management based on the observation, measurement, and actuation in the variability of the crops; with the development of new technologies related to the field of electrical and electronic engineering as well as systems and computing engineerings, such as the Internet of Things, home automation, robotic platforms, remote sensors, etc., the term "smart agriculture", which is the application of these technologies to the field of agriculture, to face the challenges that arise in agriculture in terms of productivity, environmental impact, food security and long-term sustainability (Walter, 2017).

To face these challenges it is necessary to have a complete panorama of the field to which these technologies are applied: crops are changing living ecosystems, which is why it is necessary to analyze and understand them, which implies defining certain variables that will be constantly monitored; this monitoring process generates enormous amounts of data, which, depending on the farmer's requirements, must be stored and processed in real-time or not (Kamilaris A. a.-B., 2017).

The type of data that, it can be said, receives the most attention are images: sets of thousands and thousands of photographs are taken in the orchards, to be processed for multiple purposes, among which it is found the identification of crops' types (fruits, leaves, flowers, etc.), identification of diseases in crops, classification of crops, yield estimation, etc., in different agricultural contexts; these image sets are processed with different image processing techniques, some of which are based on Machine Learning processes, and for instance, it is found K-Means Clustering, Support Vector Machines (SVM), or Artificial Neural Networks (ANN), among others; all of which are known as traditional image processing techniques (Saxena, 2014) (Singh, 2015).

On the other hand, Deep Learning (Schmidhuber, 2015), a recent approach within the field of machine learning, and which is a further development within the Artificial Neural Network and which has a better learning capacity and therefore greater classification precision (Kamilaris A. a.-B., 2018), has been tested quite successfully within the field of agriculture and is enjoying increasing popularity in recent years; to provide these capabilities, Deep Learning models require greater computational capacity and longer training time for the models, a disadvantage that has led to studies that seek to both reduce computational requirements and reduce training and inference time of said models (Wang C. a., 2016) (Cui, 2016).

These disadvantages, however, do not diminish the ever-growing interest in the applicability of Deep Learning in the field of agriculture, which is evident in the ever-increasing number of studies applied in recent years (Santos L. a., 2020); reviews such as those by Kamilaris et. Al. (Kamilaris A. a.-B., 2018) (Kamilaris A. a.-B., 2018) on the application of Deep Learning in the field of agriculture, some others such as Saxena et. Al. (Saxena, 2014) (Liakos, 2018) that also include Deep Learning techniques and others that go further and analyze the different aspects that involve the application of the Deep learning technique in this field of study (Koirala A. a., 2019).

1.2. Motivation

The research field tends to focus more on the framework than on the object of study, namely, the object whose model trained in the framework is intended to detect and/or classify; in our case, this subject is the crop product, the fruit, but it is not the fruit by itself since it must be known that the same model trained in a study subject with certain characteristics will not be able to detect the same subject of study in different characteristics, a different context, but it is the fruit in the tree canopies, before its harvest.

In many studies where Deep Learning is applied to the field of agriculture, the data set used is described and in many cases the restrictions or conditions with which it affects the study, however, the direct goal is to know the performance of the framework of work in terms of detecting and classifying the object of study; this is also reflected in research studies whose purpose is to do a field's review (application of Deep Learning in Agriculture), where there are questions about the constitution of the image datasets, among others.

However, as the question is made clear in Koirala et. Al (Koirala A. a., 2019), these studies, being focused on the performance of the frameworks, do not pay much attention to the data sets themselves, nor to the suitability of the framework for the study subject (many times they do not justify the choice of the framework based on the study's object, but rather in the current performance of the selected framework), or the specificity of the chosen dataset (even when they describe the process of constructing the dataset). Furthermore, as Santos et. At (Santos L. a., 2020) says, these research papers also do not pay attention to the restrictive time and hardware characteristics imposed by the complexity of the Deep Learning frameworks.

Since, in the field of agriculture, and more specifically, in the field of the fruit crops industry, both precision and speed are necessary when estimating total yield per harvest, and that each crop has its characteristics, It is necessary to have information in advance that allows us to properly choose the framework that will be implemented when estimating crop production. This is why the following questions are asked: ¿how does the type of study object affect the training of deep learning models concerning each of the different techniques that make up the standard today? And in turn, according to the characteristics of each object, ¿which of the standard frameworks best performs its function of detecting and classifying the object in a given set of images?

1.3. Thesis Objectives

The present thesis aims to provide the research field with an overview of the performance of the most important current frameworks implemented in different datasets/objects of study, evaluating the suitability of each framework concerning each study's object, specifical fruits in tree canopies, to facilitate the choice of frameworks for future researches; having an overview of the performance of the technique on the object of study, the process of choosing a framework is accelerated, likewise, knowing that the framework works better with that object of study, subsequent investigations can be conducted to exploit the full potential of the framework concerning the study's object, or conduct research that allows to discover and cover shortcomings in the frameworks which offer less performance.

A second goal, no less important, is to conduct an investigation that takes into account several different factors that are quite neglected by researchers in general, such as the size of the models trained, the characteristics of the hardware used to train the models (since in the present study no field implementations are carried out), the training and inference time of the models, etc., aspects that are quite important in the research field, since they affect one of the fundamental principles of modern science: the reproducibility of the experiments that support the conclusions of the studies.

1.4. Thesis Methodology

Given the goals that determine our study, it is conducted as follows: initially, a review is carried out on the current state of art researching on the application of Deep Learning in the field of agriculture, and especially in reviewing studies, in order to understand the panorama, gather a set of images enough to carry out the experiments and outline the methodology of experimentation for this thesis; the same study is carried out on the standard frameworks within Deep Learning in order to choose the techniques with the best performance, rather than the most used techniques, and their application within the agriculture's field; from these same reviews the guidelines are defined (standard measurements in the field and others not so used but that also provide important information) that will be used in the evaluation of the performance of each technique contrasted with each images' dataset; once these preliminary steps are completed, the image datasets are processed, to standardize the training process it is necessary to standardize the inputs to the frameworks, which in this case are the image datasets that

will feed the technique that trains the Deep Learning models; then, a machine is supplied with the necessary software to run the experiments, after which the training of the Deep Learning models is carried out one by one, framework against image datasets, and records of the process are obtained, which will be used later for analysis, the resulting models are collected for subsequent evaluation; finally, each model is evaluated in order to obtain the metrics that will be used to contrast and discuss the suitability of each framework.

All the products derived from the methodology of the present study can be found in the appendix section of this thesis, which includes scripts for supplying the necessary software, processing scripts for the data sets, etc.; these are available in a private repository, authored by the author of this thesis, and will be made available to the public once permission is obtained.

1.5. Thesis Structure

The remaining of this thesis is organized as follows: Chapter 2 recounts the current state of the art in the field through the review of recent studies on the subject, the importance of Deep Learning in the agriculture industry is taken as a starting point, with its development over the last decades to culminate in the report of studies specifically focused on the review of Deep Learning application in the agriculture industry, an important point that defines the methodology to use in the thesis.

Chapter 3 describes the methodology used in the development of this study, the datasets that define the study objects, the frameworks used to contrast dataset/framework performance, the work environment (hardware and software), the measures that will be used to evaluate performance, as well as the reasons why both the datasets, the frameworks, and other restrictions derived from the objective to be achieved with this thesis have been chosen.

Chapter 4 focuses on exposing and analyzing the data, based on the records obtained from the work frameworks, the defined measures used to evaluate performance are obtained, as well as various graphs that allow us to further analyze the performance of the frameworks; subsequently, based on these data, it is proceeded to discuss the suitability of each framework concerning the study's object, based on the approaches underlying the frameworks and subjects of study from which results are assumed, these will be

contrasted and discussed with the results of the experiments carried out in the present study.

Chapter 5 discusses the outcome of the experimental work based on the theoretical part, finding the reasons that explain the results obtained, thus outlining the possible answers to the questions that start this dissertation.

Finally, based on the agreements reached in the previous section, the conclusions drawn from the study are shown in Chapter 6, as well as to give some guidelines that may give way to future research or that may help in ongoing research.

2. LITERATURE REVIEW

Deep learning has a great boom in machine learning by providing multiple advantages over techniques that for many years were made with complex algorithms and that often only solutions in certain steps of the process; well, many of these processes are executed automatically by CNN models, thanks to their "learning" capacity, facilitating and speeding up those processes that were once done separately. It is not difficult to realize the popularity that deep learning reaches today in terms of its application in agriculture.

Detection of diseases in plants (stems, leaves, fruits), identification of fruits, leaves, plant varieties, etc., detection of weed, the composition of the soil, among others, are among the various agricultural contexts on which traditionally it has been applied machine vision, and which have also been the focus of studies in the application of deep learning; object detection, one of the most popular fields for the use of deep learning, is of particular importance to farmers: by identifying fruits within the treetops, for example, the number of objects can be counted In the image, and by taking images of the entire crop, the total amount of fruits produced by said crop can be estimated, and from there, the production of fruits per crop can be estimated; yield estimation is thus one of the most important agricultural contexts of study for the application of deep learning. And if there are vehicles, UAV or not, with good computing capacity, the possibility of automating the production estimation process and reducing human effort becomes evident.

Although it is easy to understand the large number of possibilities that arise within agriculture and that could be supplied by deep learning, it is surprising to realize that very few studies have been carried out in its application to various agricultural contexts; its use in agricultural yield estimation is even more limited, this may find its reason in the fact that deep learning models are computationally expensive and require powerful hardware to function. This, however, does not diminish the possibilities existing in the field, and that is that the possibilities are enormous and the constant development of the hardware allows each day to have devices capable of greater computation that require less electrical power (some developments even focus directly on reducing drastically the electrical requirement, or build more capable UAVs, for instance).

In this section, deep learning and object detection are first briefly explained. Then, a tour of the field of smart agriculture is carried out, describing the process of conception

of the term, its initial development through computer vision and other disciplines, to finally move on to the application of deep learning techniques in its development.

2.1. Deep Learning

Deep learning is a subfield of Machine Learning, which is, in turn, a part of the broad field of artificial intelligence. This makes use of Artificial Neural Networks, inspired by the biology of the human brain; a perceptron, the most basic unit of a neural network, is the computational version of a neuron, and is basically a linear classifier, activated once the threshold is reached; each perceptron has a weight for each input, giving importance to some inputs more than others, and each perceptron learns by adjusting the weights, for this, the perceptron adds all the inputs and multiplies them by their weights to then apply an activation function, RELU for the input layer perceptrons, and sigmoidal or softmax for the last capable perceptrons.

In summary, the Deep Learning process consists of building neural network architectures, grouping perceptrons into hidden layers so that by learning, they can identify patterns. The training process begins by feeding the neural network, whose weights in its perceptrons are initialized with values close to zero randomly, with information; each neuron will change its weight only if the sum of all the weights exceeds the threshold, a process known as forwarding propagation; later the obtained values are compared with the real values during the training; a cost function (square difference) is used to check how many incorrect results were obtained and then backpropagation (using Stochastic Gradient Descent functions) is used to readjust the weights. This process is repeated until the total error is minimized, this is when the network has learned.

As a development of classical Artificial Neural Network (ANN), the Convolutional Neural Networks (CNN) add a convolution layer that, using filters, creates a simplified map representing the main feature from an image with less data, keeping the entropy; to reduce dimensionality and linearity, CNNs apply Max Pooling layers, then, the entire feature map is flattened to a one-dimensional vector, and finally the layer is connected to an ANN.

After finishing this short introduction, in this section, the state of the art of the application of deep learning to the field of agriculture is reviewed, taking its "historical" development as well as the results presented so far.

2.1.1. Object Detection

In 2010, ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky, 2015), a benchmark in object category classification and detection on hundreds of object categories and millions of images, who "evaluates algorithms for object detection and image classification at large scale" and which its goals are "One high-level motivation is to allow researchers to compare progress in detection across a wider variety of objects - taking advantage of the quite expensive labeling effort" and "to measure the progress of computer vision for large scale image indexing for retrieval and annotation", as it says on its website, is launched.

ILSVRC was born following the PASCAL VOC challenge precedent (Everingham, The Pascal Visual Object Classes (VOC) challenge, 2010) , established in 2005 and which set the precedent for standardized evaluation of recognition algorithms in the form of yearly competitions. Like PASCAL VOC, ILSVRC consists of two components: a physically available dataset, consisting of a set of manually annotated training images and also, a manually annotated set of test images, with the manual annotations with-held, it can be added that, in the latest revisions, the format also includes a set of annotations validate images to validate the model's training, in addition to the set of annotations for training, and the set of annotations to evaluate performance (test); as a second component, each year a competition is held, with its corresponding workshop, where the participants propose their solutions, frameworks, algorithms, techniques, and train them with the set of images (training and validation), then the trained models they are evaluated (test); the results (predicted annotations) are sent to the evaluation server; the results of this evaluation are published at the end of the competition and competitors are invited to share their opinions about the workshop.

The dataset, with its standard format and a large number of images, allows the development and comparison of categorical object detection algorithms, by offering an evaluative framework that imposes certain goals to achieve, and by offering a fairly large dataset for model's training; for their part, the competition and the workshop facilitate the way to track progress in the development of these algorithms, techniques, frameworks, etc., and discuss the lessons learned from the most successful and innovative ideas of each year.

In 2012, Krizhevsky et al. Published "ImageNet Classification with Deep Convolutional Neural Networks" (Krizhevsky, 2012), an article that detailed the solution they had presented in the ILSVRC challenge since that same year, winning the challenge, and that completely broke the paradigms that until then, had predominated in the field of object detection; AlexNet (Krizhevsky, 2012), as this solution would be known from then on following its author name, was a CNN that could obtain a Top-5 error rate (rate of not finding the true label of a given image among its top 5 predictions) of 15.3%, when next best result trailed far behind (26.2%). Thereafter the use of CNN for object detection and classification would dominate the field.

As the first approaching for object detection in an image, it can be thought in sliding windows from left and right, and from up to down in the image to identify objects using classification; if it is wanted to detect different object types at different sizes and distances in the same image, varied sized windows with different aspect ratios can be used. Now, from these sliding windows, it will be extracted patches that will be warped (many classifiers take fixed-size images only to restrict memory's consumption, and, since here, these classifiers are trained to handle these images without losing classification accuracy). Each warped image patch is put as input in a CNN classifier to extract features; then, an SVM classifier is applied to identify the class of the detected object and a linear regressor for getting the boundary box. For detecting different objects in different locations from the picture, many windows will be created, controlling the number of them to improve the performance.

2.1.1.1. Region-Based Object Detectors

Instead, to use Sliding-Windows detection (Dalal, 2005), a region proposal method also can be used to create a region of interest (ROI) (Brinkmann, 1999); it is stated taking each pixel as its group; after, the texture for each group is calculated and that are closest are combined, combining smaller groups first to prevent a single region swallowing others; this is done until everything is combined. The resulting groups will be taken as the region of interest. This process is known as Selective Search (Uijlings, 2013).

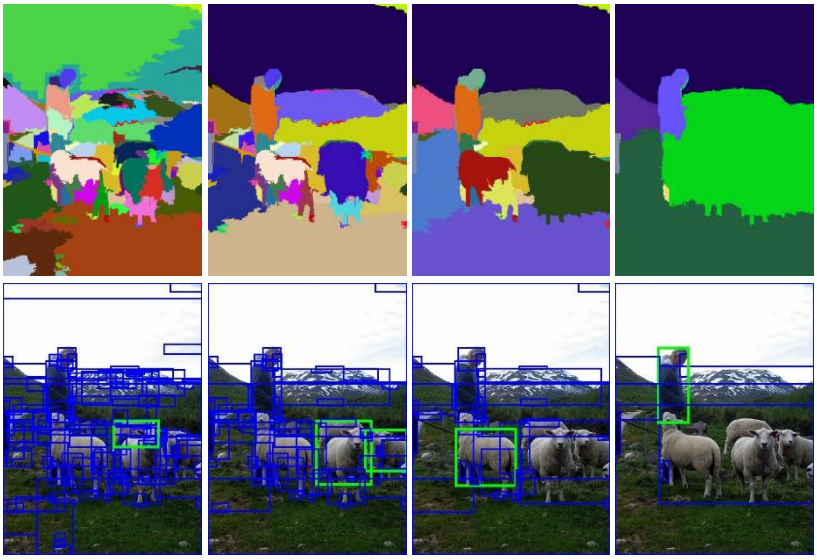


Image 2.1. *Selective search example, (Uijlings, 2013)*

Before it was said many different sliding windows or region of interest can be created when the objects to detect are very different and they are located in very different places, as a solution of the next problem: as the number of objects of interest is not fixed when a standard CNN followed by a fully connected layer is used, the output layer's length will be variable; but as the objects of interest might have different spatial locations within the image and different aspect ratios, for instance, a huge number of region of interest will have to be selected, causing a computationally blow up.

2.1.1.1.1. Region proposal Convolutional Neural Network (R - CNN)

To solve this problem, Girshick et al. (Girshick R. B., 2014) proposed a method who limits the region of interest to just 2000; using selective search 2000 region of interest are extracted from the image and they are called region proposals, the algorithm could be described as follows: first, generating an initial segmentation, many different candidate regions are obtained; second, similar candidates are merged into larger regions using greedy algorithms; finally, the resulting one is used to create the region proposals.

Therefore, these region proposals are warped into square sized images and feed into a CNN which outputs a 4096- dimensional feature vector individually; since here, it can be understood that CNN acts a feature extractor and the output layer, consisting in the extracted features, feeds an SVM classifier which detects the presence of the object of interest in the region proposal; the algorithm also predict the four offset values to increase

the bounding box's precision(helping to adjust the bounding box for the region proposal). Then, having a less amount of region of interest and improving the bounding boxes, R-CNN, the name by which this technique is known, runs faster and more accurately than using selective search.

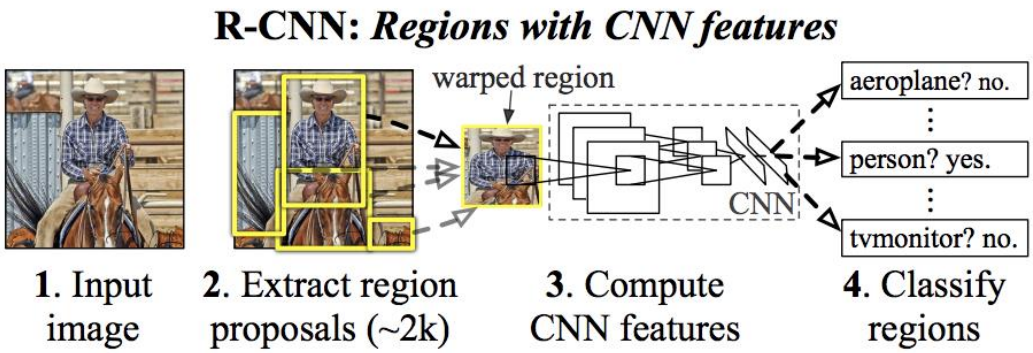


Image 2.2. *R-CNN algorithm process.*

Region of interest methods are computationally intense, as it was mentioned before; as a result, training to classify 2000 region proposal per image take also much time, testing also takes much time since it takes 47 seconds for an image, making it impossible to use for real-time implementation. To speed up the process of generating the region proposals, often less expensive region proposal methods for creating ROIs are selected, followed by using a linear regressor to tune the boundary box as possible. Also, as the selective search algorithm is fixed, there is no learning in this step, leading to a generation of bad candidates to the region proposal.

2.1.1.1.2. Fast R - CNN

As a solution, initially, a feature extractor (CNN) can be used to extract the features from whole image set; this feature map is used to identify region proposals using an external method like selective search, which are later combined with their respective feature maps (which gave rise to them), to create patches; using an ROI pooling layer, the patches are warped to a square fixed size to later feed fully connected layers. Using the region of interest (ROI) feature vector, a softmax layer can predict the class (object of interest) of the proposal region (classification) and the bounding box's offset values (localization, detecting the object's location).

As Fast R-CNN (Girshick R. B., 2015) just feed the CNN with the 2000 region proposal only once per image instead to feed it every time, Fast R-CNN reduces process time substantially (ten times faster than R-CNN in training and one hundred fifty times faster in inferencing); Fast R-CNN also improves the accuracy since its whole architecture (feature extractor, classifier, and boundary box regressor) is trained end-to-end with multi-task (classification and loss) losses.

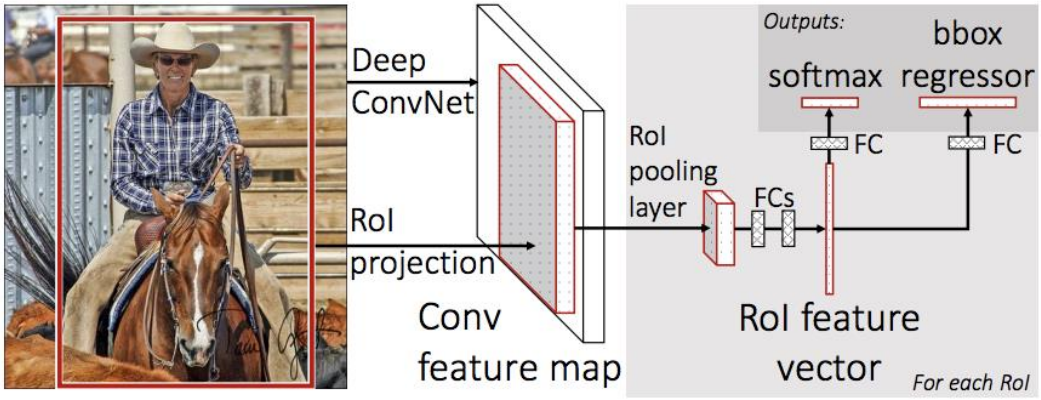


Image 2.3. *Fast R-CNN architecture, (Girshick R. B., 2015)*

Previously described methods, R-CNN and Fast R-CNN depend on external methods to generate the region proposals, selective search, for instance, methods that run in CPU, are slow and time-consuming, and therefore, affect the performance of the method.

2.1.1.1.3. Faster R - CNN

Faster R-CNN (Ren, 2017) removes this dependency by replacing the selective search algorithm by an internal deep neural network and deriving the region of interest (ROI) from the feature map instead, letting the network learn the region proposals.

Following the Fast R-CNN architecture, the images are feed in a convolutional network which produces the feature map; instead of using a selective search algorithm to process the feature map to get the region proposals, a different convolutional network predicts the region proposals; since here, region proposals are the warped in square sized patches using the region of interest (ROI) pooling layer, after which are used to classify and localize the objects(as is done by Fast R-CNN). The region proposal network(RPN) is more efficient since it runs in around ten milliseconds per image generating region of interest (ROI).

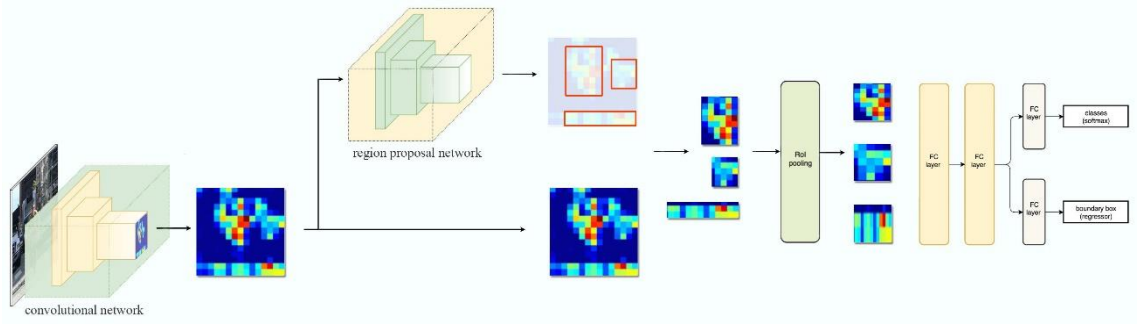


Image 2.4. *Faster R-CNN replaces the external selective search algorithm by an internal convolutional network*

Region proposal network receives the output from the feature maps, resulting from the previous convolutional network, as an input; after it slides 3 × 3 over the feature maps making class-agnostic (object-agnostic) region proposals using a convolutional network (ZF Network, VGG, ResNet, etc.); selecting the most efficient network may mean an increase in computational cost or not in exchange for more complete extraction. For instance, the ZF network produces 256 values as output, they are then fed into two separate fully connected layers which predict a boundary box and two objectness scores. A regressor is used to compute a single objectness score, but Faster R-CNN uses a classifier who detects "have an object" or "have no object" (background class) for simplicity. Objectivity is defined as whether the box contains or not an object.

Region proposal network makes a *k* number of guesses for each location in the feature maps; therefore, it outputs 4 × k coordinates and 2 × k scores per location. From the obtained guesses, it just needs one from them to be correct, so, it is better if each guess has a different shape and size. Faster R-CNN, instead of making random boundary box proposals, predicts δx, δy offsets relative to the top left corner of some reference boxes (anchors, priors, or default boundary boxes); offsets values are constrained to resemble the anchors.

It is needed anchors to be centered at each location since they will be used to make *k* predictions per location, each prediction is related to a specific anchor and different locations share the same anchor shapes. The anchors are pre-selected to get diverse anchors which cover real-life objects with different scales and aspect ratios acceptable well; this leads the initial training with better guesses, allowing each prediction to get specialized in a certain shape; thereby, early training is made more stable and easier.

Faster R-CNN uses nine anchor boxes: three different scales at three different aspect ratio; using nine anchors per location, it gets 2 × 9 objectness scores and 4 × 9 coordinates per location.

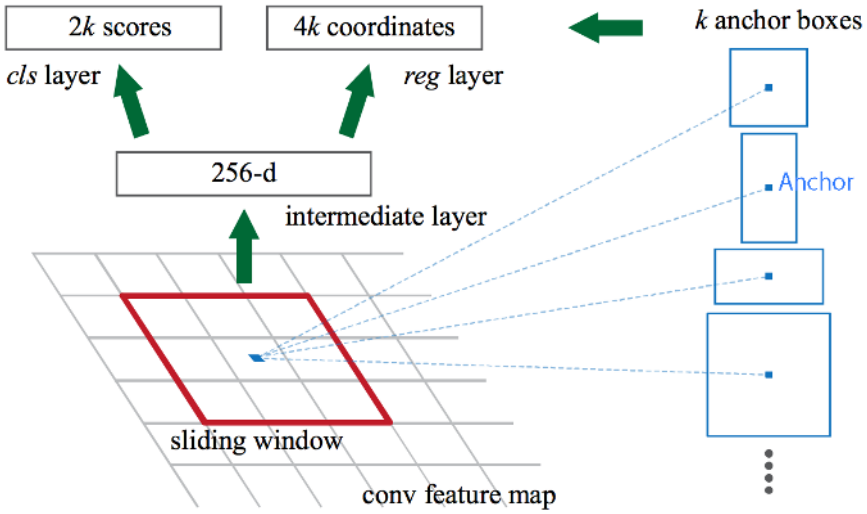


Image 2.5. *Region proposal network (Ren, 2017)*

The analysis can be continued, assuming a characteristics map, for example, that detects the right eye of a face; it could be used to locate the whole face since the right eye should be on the face's top-left corner, enabling it to locate the face. Now, it is supposed another feature maps who detect the left eye, or the nose or the mouth, whose results are going to combine to locate the face more accurately. Faster R - CNN uses a detector which applies multiple fully connected layers to make a prediction, but with 2000 regions of interest, this process can be computationally expensive.

As the aforementioned region-based feature maps are independent of regions of interest they can be computed outside each region of interest(ROI); from here, reducing the work needed for each region of interest(ROI), the further work is simpler, improving the speed and making R-FCN (Dai, 2016) faster than Faster R - CNN.

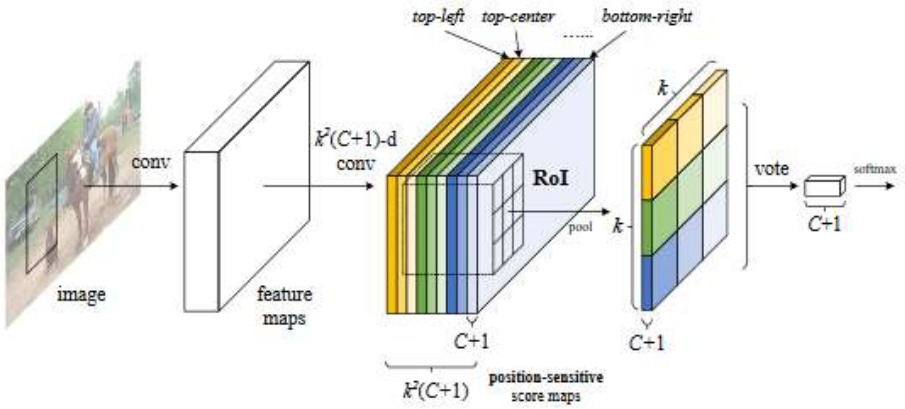


Image 2.6. *The main idea of R-FCN for object detection, (Dai, 2016)*

To understand this better, a 5 × 5 feature map **M** can be considered, for instance; it is divided into 3 × 3 sections, getting nine sections from which nine new specialized feature maps will be created: the first one for the top left corner (TL), the third one for the top right corner (TR), and so forth. Since the square is divided into nine sections, nine feature maps each detecting the corresponding region of the object can be created. These feature maps are called position-sensitive score maps since each map detects (scores) a sub-region of the object.

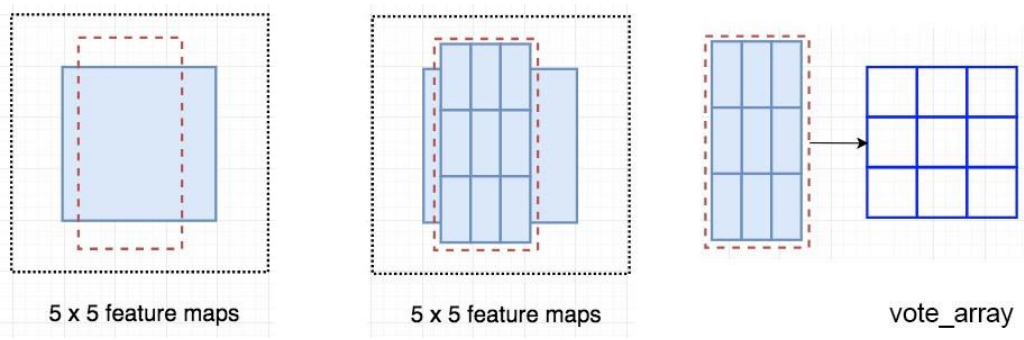


Image 2.7. *R-FCN's process: applying region of interest (ROI) to the feature maps, it gets a 3 x3 array*

Now, dotted red rectangle in the **Image 2.7** will be taken as the proposed region of interest**;** it is divided into a 3 × 3 region and the likelihood each region contains the corresponding section of the object is asked; for instance, how likely the top-left region of interest contains the left eye; the results are stored into a 3 × 3 vote array (final diagram in **Image 2.7**). The process described above is called position-sensitive Region Of Interest - pool and it is very close to the Region Of Interest pool described before.

Once all values for the position-sensitive region of interest pool are calculated, the class (object) score is obtained by averaging all its elements. For instance, for **C** classes to detect, taking C + 1 classes for including a new class for the background (non-object); each class will have its own 3 × 3 score maps and therefore a total of (C+1) × 3 × 3 score maps; using its own set of score maps, a class score for each class is predicted; finally, a softmax is applied on those scores for computing each class's probability.

2.1.1.1.4. Instance Segmentation and Semantic Segmentation

Until here, how these processes divide the image into sections called segments has been seen; since not all segments have objects inside them, it is logical to avoid those segments that do not contain information; selecting important segments for its processing is then called image segmentation. As an image is a collection or set of different pixels, image segmentation is used to group the pixels with similar attributes, similar to selective search.

From these object detection techniques, a set of four coordinates is gotten, bounding box, as a result of the detection algorithms; but, just these bounding box with a class label (object name) is gotten, without any other useful information, for instance, the object shape. Instead, image segmentation creates a pixel-wise mask for each class given in the image, thus giving us easier to analyze results and a much more complete understanding of the objects present in the image (Shapiro, 2001) (United States Patente nº 10/618,543, 2003).

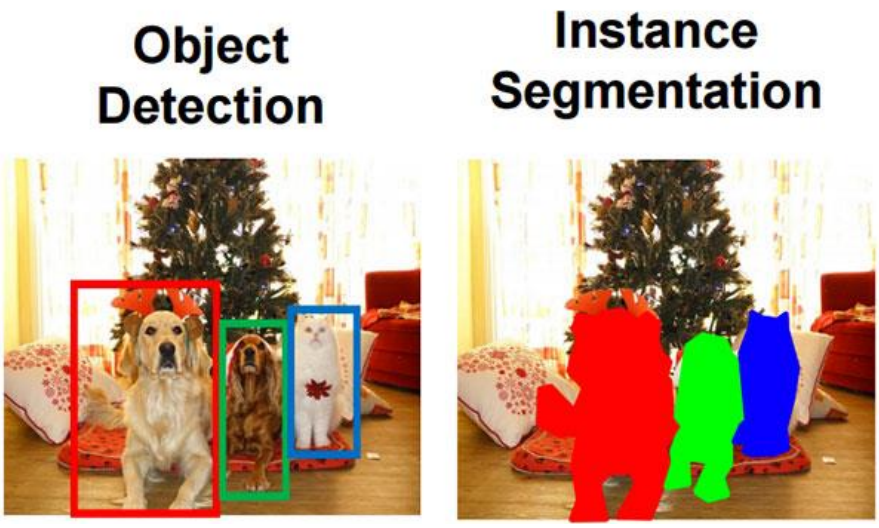


Image 2.8. *Comparison between Object Detection and Instance Segmentation(one of two Image Segmentation types), (University, 2020)*

Such a difference is important in a context in which the shape of each object on the screen, as well as the number of objects on the screen, is of vital importance to interpret the predicted results, for example, in the tomographies performed on cancer patients it is necessary to differentiate each one of the present tumors, to know what type of tumor it is, and if it is cancer, in which stage of advance it is.

Two types of image segmentation can be defined: Semantic Segmentation and Instance Segmentation. Semantic Segmentation detects all the objects present in an image at pixel-wise level, resulting in regions with different classes (objects)**;** it groups pixels in a semantically meaningful way, pixels belonging to different classes are grouped separately. For its part, Instance Segmentation identifies each object instance for every known object within an image assigning a label to each pixel of the image. It is used for tasks such as counting the number of objects. Instance segmentation has to goals: classify and localize each object instance using a bounding box (object detection), and classify each pixel into a fixed set of categories but avoiding to differentiate object instance (segmenting instances).

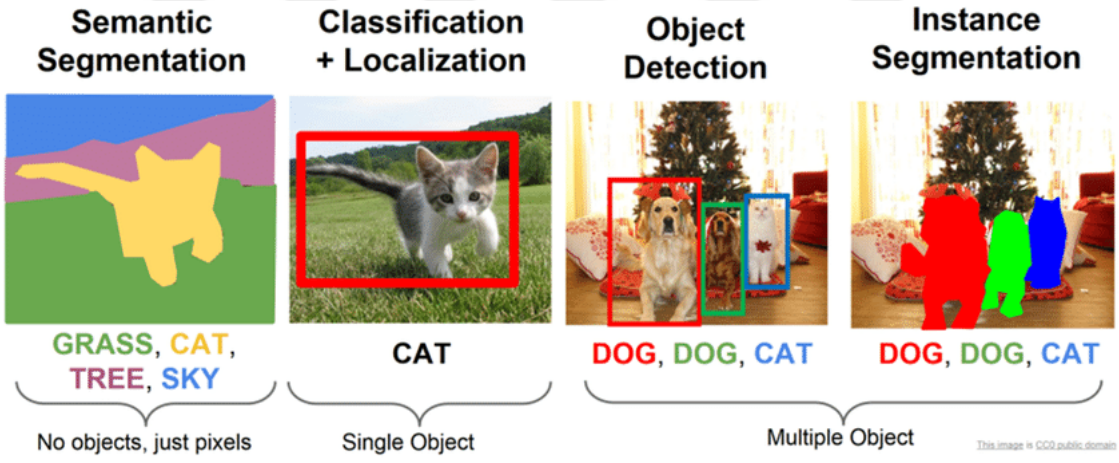


Image 2.9. *Comparison of semantic segmentation, classification and localization, object detection and instance segmentation, (Li, 2017)*

2.1.1.1.5. Mask R – CNN

A couple of things about Faster R-CNN can be said: it is not designed for pixel-wise alignment between network inputs and outputs and for each candidate class it has a label and a bounding box offset, thereupon, it does not provide any kind of shape for each class.

Mask R-CNN (He K. a., 2017) is a development of Faster R-CNN, introducing a third branch who predicts a mask for each region of interest at a pixel-wise level, it outputs an object mask for each object in the given image.

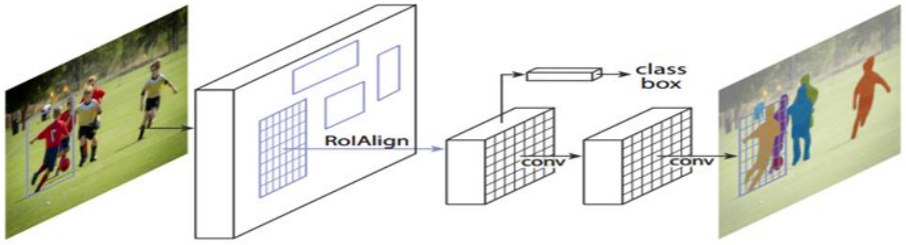


Image 2.10. *Mask R-CNN proposition, (He K. a., 2017)*

Mask R-CNN has two parts: a region proposal network and a binary mask classifier. Like Faster R-CNN, the images are feed in a CNN who generates all feature maps which later go through the region proposal network.

The Region Proposal Network(RPN) generates multiple regions of interest (RoI) with a lightweight binary classifier who build nine anchors over the images to return object/no-object scores; by using anchor boxes to detect objects with variated scales or overlapped objects, Mask R-CNN improve object detection's speed and efficiency; then, all anchor boxes that belong to the background class are removed and all anchor boxes with an intersection over union (IoU) value greater than a threshold value(0.5 by default) are filtered; finally, anchors with high objectness score are selecting by applying Non-Max suppression over the filtered anchors to avoid other bounding boxes detecting the same object. As a result, around 300 region proposals are generated per image.

The Region of Interest (RoI) Align network warp multiple bounding boxes into a fixed dimension and outputs them; then, the warped features are fed into fully connected layers for classification using softmax, and for refining boundary box prediction, using the regression model; warped features are also fed into Mask classifier: a set of two CNN's, one for object detection and a mask prediction network, which output a binary mask for each region of interest (RoI). Since the Mask Classifier runs in parallel to the label and box prediction network, for each generated region of interest (ROI), the network predicts masks belonging to all the classes without competition among classes.

Image 2.11. *Mask R-CNN results: it detects two different classes (airplane and person) and highlights each class instance with a different color (Abdulla, 2017)*

2.1.1.2. Single Shot Detectors

So far, all the algorithms described making use of regions to detect objects within images, being precious but having a computational cost. Also, Faster R-CNN and Mask R-CNN have a dedicated Region Proposal Network followed by a classifier.

Taking back to Sliding-windows detection, as a summary, sliding windows over feature maps to detect objects, different window for shapes different object types, it can be noted a mistake: it uses the windows as the final boundary boxes, hereafter, it needs too many shapes to cover most objects. It can be handled in a more effective way of treating the window as an initial guess, thus, the detector will predict the class and the boundary box from the current sliding window simultaneously.

This way is similar to Faster R-CNN and Mask R-CNN anchors, but it enables the single shot detector to predict both class labels and boundary boxes at the same time. Faster R-CNN and Mask R-CNN, use one convolution filter to produce a five-parameter prediction: four parameters on the predicted box relative to an anchor and one objectness confidence score. A single shot detector, use the convolution filter also to predict **C** class probabilities for classification (one per class).

Single-shot detectors typically change precision for real-time performance (speed), causing them to have trouble detecting small or very close objects.

2.1.1.2.1. Single Shot Detector

SSD (Liu W. a., 2016) is a single shot detector that uses a VGG16 network as a feature extractor (equivalent to the CNN in Mask R-CNN and Faster R-CNN). Convolution layers are added afterward and convolution filters are used to make independent object detection from multiple feature maps to enable the detector to detect not only large objects since the convolution layers reduce spatial dimension and resolutions.

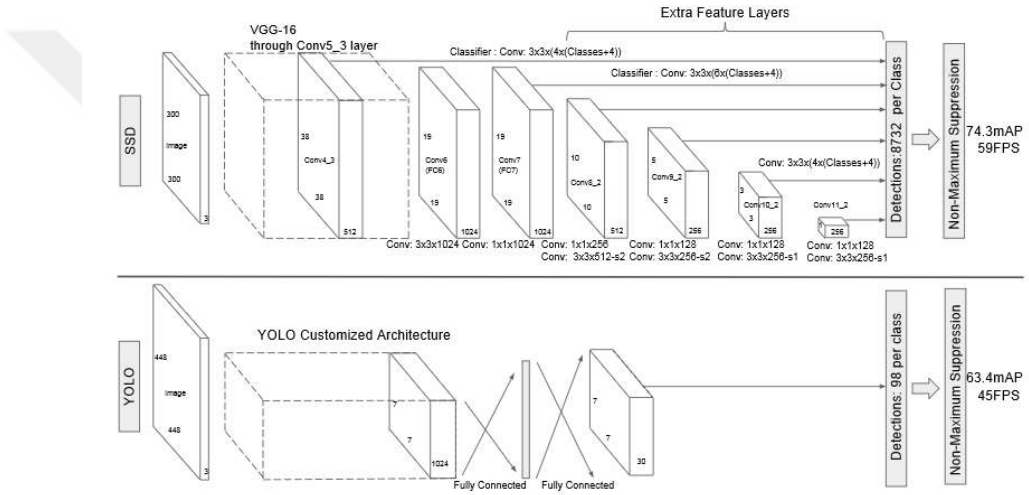


Image 2.12. Comparison between SSD and YOLO (Liu W. a., 2016)

SSD uses layers already deep down into the convolutional network to detect objects. This drops significantly the spatial resolution, leading to missing locating small objects that are too hard to detect in low resolution. In such a case, it is needed to increase the resolution of the input image. SSD does not use a delegated region proposal network and predicts the boundary boxes and the classes directly from feature maps in one single pass. SDD computes both the location and class scores using small convolution filters.

SSD uses **VGG16** to extract feature maps, starting from the left, it also defines a scale value for each feature map, detecting objects at the smallest scale (0.1 or 0.2), increasing linearly to the rightmost layer (0.9), using the **Conv4_3** layer. The width and the height of the default boxes are obtained by computing the scale value with the target aspect ratios.

Once the feature maps are extracted, SSD applies 3 × 3 convolution filters for each cell to make predictions(computing results just like regular CNN filters). For each cell (location), it makes 4 object predictions, each prediction composes of a boundary box and 21 scores for each class (one extra class "0" for no object), and the highest score is picked as the class for the bounded object. Conv4_3 makes a total of 38 × 38 × 4 predictions: four predictions per cell regardless of the depth of the feature maps.

SDD uses an anchor similar concept: default boundary boxes which are clusters of ground truth boundary boxes, instead of making random guesses, SDD guesses based on the default boundary boxes; the default boxes are pre-selected carefully trying to cover multiple objects from real life, keeping a minimum of default boundary boxes(four or six) with one prediction to each one; the boundary box predictions, now, are relative to the default boundary boxes offsetting to a default boundary box per cell.

SSD predictions are positive when the default boundary box intersection over union (IoU) with the ground of truth is greater than a threshold(0.5 by default), and negative if it is lower; hence, respective predicted boundary box for positive matches are use to calculate the cost, this lead to each prediction to predicts shapes closer to the respective default boundary box, making them more stable and diverse at training.

SDD ensures the ration between positive and negative matches (sorted by the confidence loss and selected the higher ones) is at most 3:1 to handle the class imbalance problem(predicting more than the number of objects lead to getting more negatives than positives, tipping the training negatively). Data augmentation is used to improve the accuracy; for it, it is possible to randomly sample a patch with a ½ to 2 aspect ratio, to sample a patch with 0.1, 0.3, 0.5, 0.7, 0.9 intersections over union (IoU), therefore, it will be resized and a one-half of data will be flipped.

Combining both multi-scale feature maps and default boundary boxes enables SDD to detect large and small objects at the same time: higher resolution feature maps detect small objects, even so, SDD performs bad with very small objects, requiring using higher resolution images input to mitigate this problem.

2.1.1.2.2. YOLOv3

YOLO (Redmon J. a., 2015), another Single Shot Detector, uses Darknet (Redmon J. , 2013 - 2016), own proposed architecture, followed by convolutional layers, and it does not use multi-scale feature maps to make independent predictions.

YOLO follows the next methodology: first, it divides the input image into an $\mathbf{S} \times \mathbf{S}$ grid, each grid cell predicts a fixed number of boundary boxes with one box confidence score per boundary box, only one object regardless of the number of boxes and several conditional class probabilities(one per class for the likeliness of object class). However, YOLO has a one-object rule which limits how close detected objects can be; hence, it has some limitations on how close objects can be.

Each boundary box contains 5 elements: (*x, y, w, h*) and a box confidence score which reflects how likely the box contains an object (objectness) and how accurate is the boundary box. It requires to normalize the bounding box width *w* and height *h* by the image width and height, *x* and *y* offset to the corresponding cell; hence, *x, y, w,* and *h* are all between 0 and 1. Each cell has 20 conditional class probabilities which are the probability that detected object belongs to a particular class (one probability per category for each cell).

YOLO builds a CNN network to predict a (7, 7, 30) tensor using the CNN network to reduce the spatial dimension to 7×7 with 1024 output channels at each location. Then, it performs a linear regression using two fully connected layers to make 7×7×2 boundary box predictions. For the final prediction, those with high box confidence scores (greater than 0.25) as our final predictions are kept. The class confidence score measures the confidence in both the classification and the localization (where an object is located).

YOLO has 24 convolutional layers followed by two fully connected layers (FC), some convolution layers use 1 × 1 reduction layers alternatively to reduce the depth of the features maps. The last convolution layer outputs a tensor with shape (7, 7, 1024) and it is then flattened. Using two fully connected layers as a form of linear regression, 7×7×30 parameters are output and then reshaped to (7, 7, 30), i.e. two boundary box

predictions per location. A faster but less accurate version of YOLO, called Fast YOLO (Shafiee, 2017), uses only nine convolutional layers with shallower feature maps.

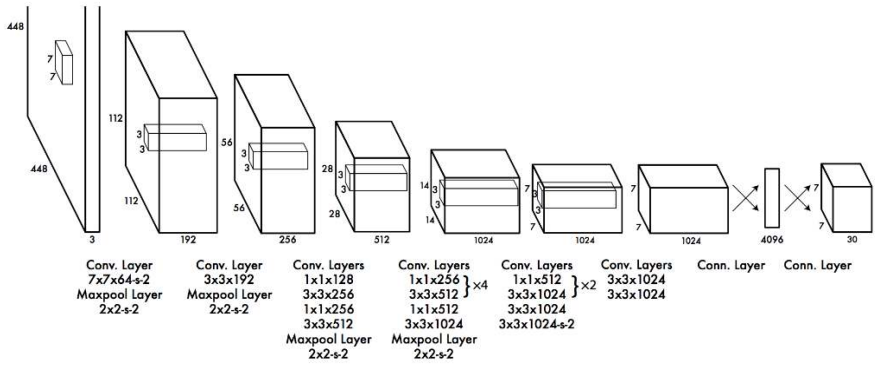


Image 2.13. *YOLO's architecture. (Redmon J. a., 2015)*

To compute the loss for the true positive, the one with the highest intersection over union (IoU) with the ground truth is selected, making it responsible for the object; this strategy leads to specialization among bounding box predictions, it means, each prediction becomes better at predicting certain sizes and aspect ratios. YOLO uses sum-squared error between the predictions and the ground truth to calculate loss which is composed of classification loss, localization loss (errors between the predicted boundary box and the ground truth), and confidence loss (the objectness of the box).

YOLO applies non-maximal suppression to remove duplications with lower confidence to avoid duplicate detections for the same object, for the non-maximal suppression implementation the following steps can be used: first, sorting the predictions by the confidence scores; second, starting from the top scores, ignore any current prediction if previous predictions that have the same class and IoU > 0.5 with the current prediction is found, this step will be repeated until all predictions are checked.

YOLOv2 (Redmon J. a., 2016) is the second version of the YOLO, its address to fix the YOLO's higher localization errors and its lower recall. As improvements it: adds batch normalization in convolution layers, trains a classifier network and the replaces the fully connected layers with a convolution layer and retrains it end-to-end for the object detection, it predicts offsets to each of the anchor boxes instead to predict boundary boxes, constraining the offset values, the diversity of the predictions is maintained and each prediction focuses on a specific shape, making initial training will be more stable; anchors are used as clusters, and K-means clustering is applied on the training data to locate the

centroids of the top-K clusters, a sigma function is applied to constraint the possible YOLO's predicted(five values) offset range, adopts pass through to handle the spatial dimension decreasing problem and applies multi-scale training to force the network to predict well for variated dimension and scale images.

The following changes are made in the architecture: the fully connected layers for predicting the boundary box are removed, the class prediction is moved from the cell level to the boundary box level, the last convolution layer is replaced with three 3×3 convolutional layers each outputting 1024 output channels and then a final 1×1 convolutional layer to convert the $7 \times 7 \times 1024$ output into $7 \times 7 \times 125$ is applied, the input size is reduced by 16, and one pooling layer is removed.

YOLO9000 (Redmon J. a., YOLO9000: Better, Faster, Stronger, 2016) combines samples from COCO (Lin T.-Y. a., 2014) to learns to find objects and uses ImageNet to learn to classify these objects, four ImageNet data for one COCO (Lin T.-Y. a., 2014) data, thus, reaching to detect over 9000 classes classification with a 9418 node WordTree. Since YOLO extracts similar features from related objects it can detect 156 categories, without having been trained directly on detecting them, from these feature values. While evaluates, YOLO test images on categories not trained directly locating them but it trained how classifying.

Most classifiers assume output labels are mutually exclusive. It is true if the output is mutually exclusive object classes. Therefore, YOLO applies a softmax function to convert scores into probabilities that sum up to one. YOLOv3 (Redmon J. a., 2018) uses multi-label classification, replacing the softmax function with independent logistic classifiers to calculate the likeliness of the input belongs to a specific label. YOLOv3 uses binary cross-entropy loss for each label instead of using mean square error in calculating the classification loss. Avoiding the softmax function, the computation complexity is reduced.

YOLOv3 predicts an objectness score for each bounding box using logistic regression. It changes the way in calculating the cost function. If the bounding box prior (anchor) overlaps a ground truth object more than others, the corresponding objectness score should be 1. For other priors with overlap greater than a predefined threshold (default 0.5), they incur no cost. Each ground truth object is associated with one boundary

box prior only. If a bounding box prior is not assigned, it incurs no classification and localization lost, just confidence loss on objectness; tx and ty (instead of by and by)are used to compute the loss.

YOLOv3 uses Darknet-53 (replacing to Darknet-19) as the feature extractor. YOLOv3 represents a significant improvement in detecting small objects, as well as performs very well in the fast detector category when speed is important, though it has higher localization error.

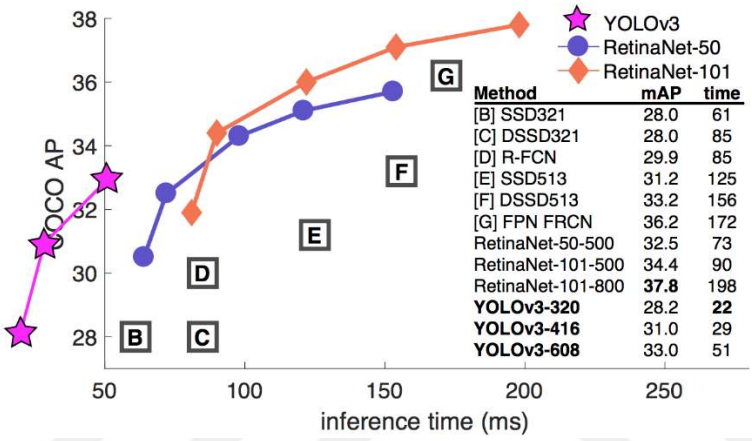
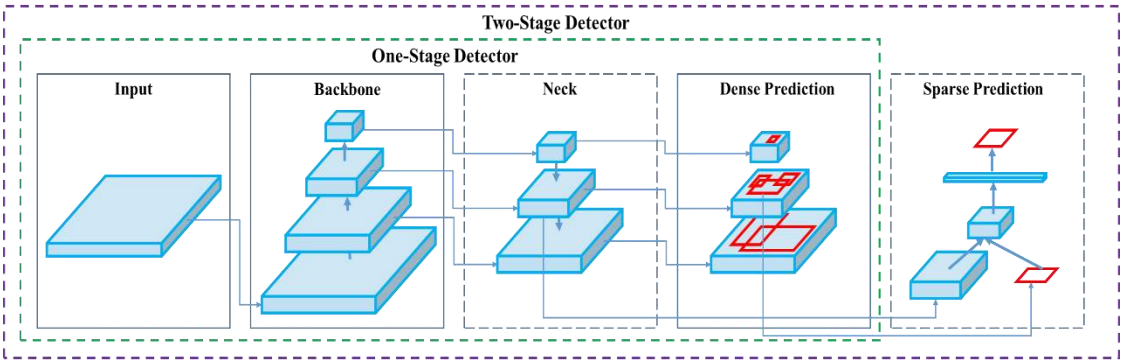


Image 2.14. *YOLOv3 performance (Redmon J. a., YOLOv3: An Incremental Improvement, 2018)*

2.1.1.2.3. YOLOv4

So far, the current state-of-the-moment frameworks can be summarized as constituted by an architecture that is trained to classify (backbone), a layer that is dedicated to predicting the class and boundary boxes of objects (head), and a set of layers, with several bottom-up and top-down paths, located between the backbone and the head whose function is to collect feature maps in different stages (neck). Likewise, most of these methods are trained off-line, and researchers have proposed different methods to increase the precision of detectors without increasing inference time at the cost of increasing the cost of training, changing the training strategy; these methods will be called "bag of freebies" (Zhang , y otros, 2019). Methods have also been proposed to significantly increase accuracy but at the cost of slightly increasing inference time, these plug-ins or post-processing methods will be called "bag of specials".



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Figure 2.1. *General description of state-of-the-art object detectors. (Bochkovskiy, 2020)*

Bochkovskiy et al. (Bochkovskiy, 2020), to build a complete detector capable of being trained with a single GPU, avoiding the difficulty of buying more than one GPU to train in Multi-GPU (state of the art for many modern detectors), which has more than respectable performance As for speed and precision, they developed different experiments to select, among all the proposals that today constitute the state of the art of object detection, the best architecture (backbone), the best detector (head), and a different bag of freebies and bag of special methods that substantially improve the performance of the proposed architecture.

As a backbone, YOLOv4 uses CSPDarknet53 (Wang, y otros, 2020), it used Darknet53 with CSPConnections for feature extraction. A DenseBlock (Huang, Liu , Van Der Maaten, & Weinberger, 2017) is comprised of multiple convolution layers, each Hi layer is made up of batch normalization, ReLU, and are followed by convolutions; This Hi layer receives the output of all previous layers and the original layer as an input, producing four feature maps, increasing the radius of growth. A network composed of Cross-Stage-Partial-connections (CSP) (Wang, y otros, 2020), CSPNet, separates the input feature maps of the DenseBlock into two blocks: the first to bypass the DenseBlock and forms part of the input to the next transition layer, the second part goes through the entire DenseBlock, until it reaches the transition layer; This reduces computational complexity.

Previously, it has been described how FPN (Lin T. a., 2017) is placed between the backbone and the header, connecting different bottom-up and top-down layers, to enrich the information in the head, in this case, FPN is the neck. YOLOv4 uses two components as part of its neck: spatial pyramid pooling layer (SPP) and Path Aggregation Network (PAN).

SPP (He, Zhang, Ren, & Sun, 2015) replaces the last pooling layer, the feature maps are specially divided into $m \times m$ bins (m being 1, 2 and 4), applying a maximum pool to each bin for each channel, forming a representation of fixed length, to be analyzed with FC-layers. YOLOv4 uses YOLOv3 as the detector, head, so it adapts SPP to YOLO (Huang, y otros, 2020): SPP must retain the spatial dimension, so a maximum pool is applied to a nucleus divided by 1×1, 5×5, 9×9, 13×13, whose feature maps are concatenated to form an output.

PAN (Liu, Qi, Qin, Shi, & Jia, 2018) enters a shortcut that only takes about 10 layers to go to the top layer; this concept generates fine-grained localized information available to top layers; feature maps are concatenated together and using element-wise max operation, PAN merges information from all the capable ones. Spatial Attention Module (SAM) (Woo, Park, Lee, & Kweon, 2018) applies attention to the neural network: maximum pool and average pool are applied to a separate input feature map to create two sets of feature maps that are then fed into a convolutional layer followed by a sigmoidal function to create spatial attention, which is applied to a feature input to refine maps of characteristics; for YOLOv4 SAM is used from the point-wise attention rather than spatial-wise, it replaces the PAN shortcut connection to concatenation.

Now, for the backbone, of all the reviewed methods, YOLOV4 makes use of the following bag of freebies methods: CutMix data augmentation, Mosaic data augmentation, DropBlock regularization, and Class label smoothing. CutMix (Yun, 2019) crop a portion of an image and superimpose it on another image, the ground truth labels are adjusted proportionally to the area of each patch, forcing the model to learn to detect objects with different sets of characteristics, thus avoiding overfitting, without losing information or training efficiency. In Mosaic, four images are combined into one when training, improving detection of objects out of their normal context; each mini-batch contains a long variant of the image, reducing the need for large mini-batch sizes at

estimating the mean and variance. Instead of dropping individual pixels, a square-sized block (Ghiasi, Lin, & Le, 2018) is dropped, allowing you to work with a dropoff in convolutional layers. Instead of using 1.0 as the upper limit of a prediction, label smoothing (Müller, Kornblith, & Hinton, 2019) always sets the target to a lower value, 0.9 for instance, thereby mitigating overfitting.

For the detector: CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, multiple anchors for single ground truth, Cosine annealing scheduler, Optimal hyperparameters, and Random training shapes. Complete IoU Loss (CIoU) (Zheng, y otros, 2020) increases the overlapping area of both ground truth and predicted boxes, minimizes their central point distance, and maintain the boxes aspect radio's consistency. CmBN is a modification of the Cross-Iteration Batch Normalization (CBM) (Yao, 2020), which estimates the mean and variance of previous *k* iterations using a proposed fit, but obtaining these values only in the mini-batches within a single batch.

Self-Adversarial Training (SAT) is a data augmentation technique proposed by YOLOv4; here, first, develop a forward pass over a training set, then change the image in such a way that it degrades the performance of the detector as much as possible, this is an adversary attack to the current model, then it is trained with the new image but with boundary box and class original label, thus helping to generalize the model and avoid overfitting.

YOLOv4 uses a scaling factor (>1.0) as a parameter to compute boundary boxes to eliminate grid sensitivity; also it uses multiple anchors for a single ground truth when IoU is upper than a given threshold. The cosine annealing scheduler (Loshchilov, 2016) adjust the learning rate according to a cosine function, starting by reducing the large learning rate slowly, and then, reducing the learning rate quickly halfway to finish with a tiny slope in reducing the learning rate.

To find the optimal hyper-parameters for training, a series of experiments are conducted using genetic algorithms to train with GIoU loss and search 300 epochs for min-value five thousand sets. Finally, to improve generalization, YOLOv4 is trained using different image sizes (Multi-Scale Training).

From the bag of specials for the backbone, it takes Mish activation, Cross-stage partial connections (CSP), and Multi-input weighted residual connections (MiWRC). Mish (Misra, 2019) is an activation function similar to ReLU and Swish (Ramachandran, Zoph, & Le, 2018) which outperforms them and increases the YOLOv4 accuracy. Multi-input weighted residual connections (MiWRC) is a contribution that YOLOv4 borrows from EfficientDet (Tan, Pang, & Le, 2019), considered state-of-the-art right now.

For the detector: Mish activation, modified SPP-block, modified SAM-block, modified PAN path-aggregation block, and DIoU-NMS. The Distance-IoU Loss (Zheng, y otros, 2020) is employed as a factor in non-maximum suppression(NMS), taking the distance between two bounding boxes' central points and the IoU to suppress redundant boxes, making it more robust against the occlusion cases.

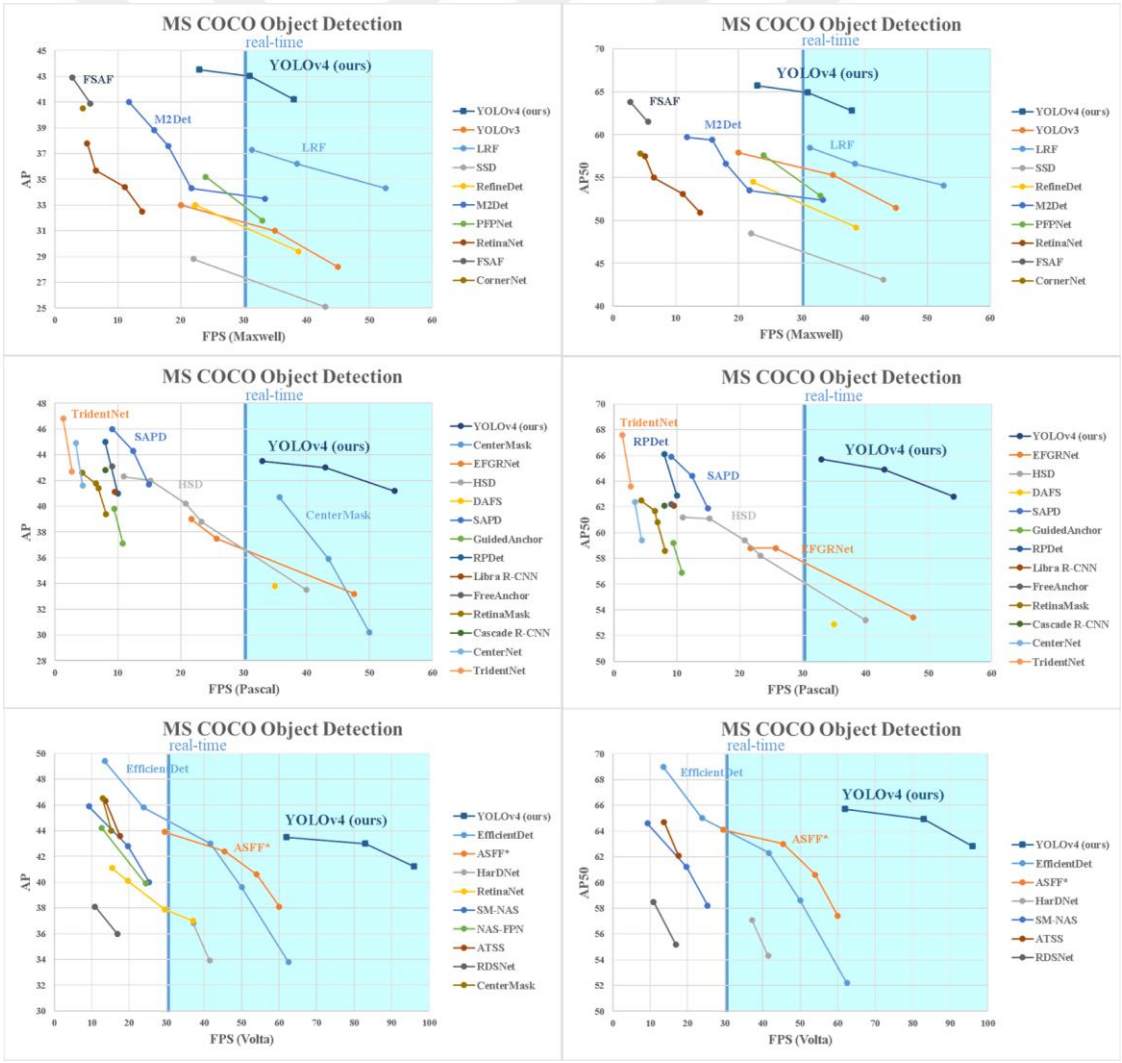


Figure 2.2. *Comparison of YOLOv4 performance against other object detectors. (Bochkovskiy, 2020)*

2.1.1.2.4. RetinaNet

A common problem that can be seen from the above detector is detecting small objects. Feature Pyramid Networks (FPN) (Lin T. a., 2017) replaces the feature extractor for different detectors and generates higher quality feature maps (designed with the feature pyramid concept), improving accuracy and speed.

FPN is composed of two pathways: bottom-up and top-down. Going bottom-up, the spatial resolution decreases while the semantic value increases for each layer; the top-down pathway then constructs high-resolution layers from each semantic rich layer; FPN also adds lateral connections between the constructed layers and their origins (corresponding feature map) to improves the localization prediction, making the training easier.

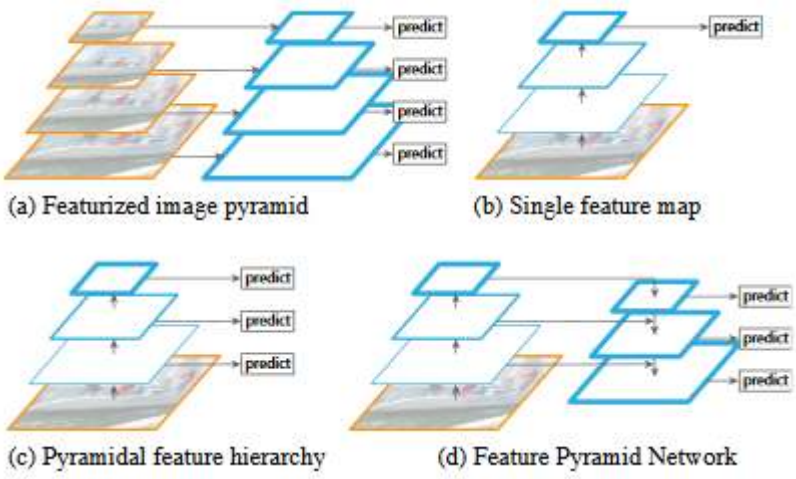


Image 2.15. *Feature pyramids built upon image pyramids. (Lin T. a., 2017)*

Now, going from the top layer module (*C1*) down, a 1 × 1 convolution filter is applied to reduce its channel depth to 256, creating the first feature map(M5); next, the previous layer is upsampled by 2 using nearest neighbors, the corresponding layer module(*Ci*) is filtered again(1 × 1) and added them element-wise, reducing the aliasing effect; a 3 × 3 convolution is applied to all merged layers, getting the pyramid feature map.

The same process is applied till *C1* because its spatial dimension is too large and it could slow down the process too much. Sharing the same classifier and box regression ensures all pyramid feature maps have 256-d output channels.

As a feature detector, FPN can be used with RPN, extracting the feature maps and feeding them into the RPN, also RPN will generate the RoI, that will feed Faster R-CNN, etc. FPN is also good extracting mask from image segmentation.

Another common problem is the class imbalance, which affects negatively the accuracy of the training. Focal Loss (Lin T. a., 2020) reshapes the loss function to focus training on hard negatives by down-weighting the easy negatives; a modulating factor is added to the cross-entropy loss function, giving to Focal Loss two properties: as first one, the modulating factor is near 1 when an example is misclassified and *pt* is small, unaffecting the loss, while *pt* is going onto 1, the factor goes to 0 and the loss for well-classified examples is down-weighted; as a second property, a focusing parameter *which* smoothly adjusts the rate at which easy examples are down-weighted is added, being Focal Loos equivalent to the Cross-Entropy loss function when the parameter is 0 and likewise increasing the modulating factor's effect as the parameter is increased. Besides, an α parameter, which yields slightly improved accuracy, is added and a sigmoid activation function for computing *p is used* resulting in greater numerical stability.

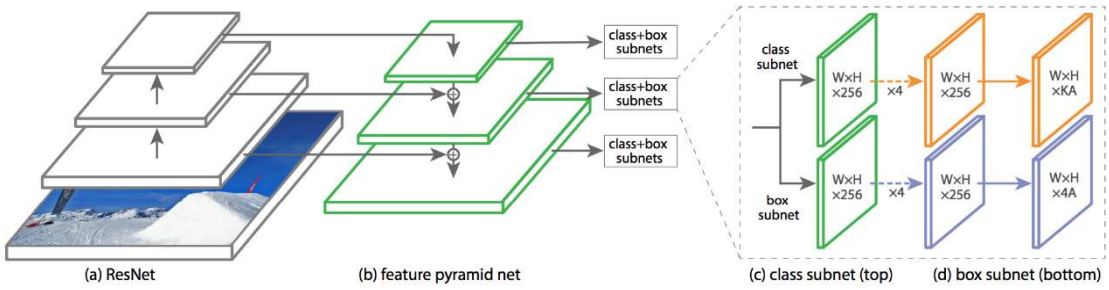


Image 2.16. *RetinaNet architecture using FPN. (Lin T. a., 2020)*

Resnet (Lin T. a., 2020) is a detector proposed in the same paper: FPN is used on top to ResNet to build a rich multi-scale feature pyramid from one single resolution input image but making the following changes: the pyramid feature map is generated from P3 to P7 (for improving the accuracy of large object detection), the P2 is not used due to computational cost and P6 is obtained by stridden convolution (instead of to downsampling).

ResNet also uses anchors in each pyramid feature map with three different aspect ratios and different sizes, at nine per level; a 3×3 convolution layered FCN subnet is used to predict the probability of object presence at each spatial position for each *A* anchors

and *K* classes; another FCN subnet is attached to each pyramid level to regress each anchor offset to a nearby ground of truth when it is possible.

At the training, the total image's focal loss is computed summing all 100000 anchors' focal loss, normalized by the ground-truth box assigned anchors' number. At inference, the confidence is thresholded at 0.05, after, box prediction from at most thousand top-scoring predictions per FPN level are decoded; later, top predictions from all levels are merged, a non-maximum suppression(NMS), thresholded at 0.5, is applied, producing the final detections.

2.2. Smart Agriculture

Agriculture is one of the most important sectors of human work in the history of humanity; Since man learned that cultivating would make his life easier and abandon nomadism for a sedentary lifestyle, agriculture has gone through many important milestones, from the use of the plow, crop rotation, forgotten technologies such as terrace cultivation, etc., and its development continues on par with the development of the societies of man (Walter, 2017).

Starting in the second half of the 20th century, a series of factors led to the rapid industrialization of agriculture throughout the world: the development of new vaccines, medicines and, in general, medicine allowed the exponential growth of the world population, thus increasing the demand for food and therefore the need to produce food efficiently and in the shortest possible time. Consequently, various scientific advances in the field of agriculture made their appearance: artificial fertilizers to increase crop production, chemical pesticides to eliminate pests, the replacement of the old plow by tractors, etc., are among the technological advances who saw its origin in the 20th century.

In the last decade of the last century, various studies were carried out to improve the efficiency of agricultural production, giving way to a concept known as precision agriculture. Precision agriculture has a wide variety of definitions, among which it can be stated: such agriculture which increases the number of correct decisions per unit area of land per unit time with associated net benefits (McBratney, 2005); the former definition focuses in the fineness of the decision regarding both space and time and it does not involve any particular technology or set of technologies, although it makes a lot of use of it.

Another definition is given to us by the European Parliamentary Research Service: “precision farming is a modern farming management concept using digital techniques to monitor and optimize agricultural production processes” (Daheim, Poppe, & Schrijver, 2019); as a complement to this definition, the study adds some examples: instead of using a certain amount of the same fertilizer in all crops, or feeding the same amount of a specific food to different types of animals, the precision agriculture constantly monitors variations in crop conditions or herd/animal group conditions to carry out a strategy of farming/raising animals that drives production while reducing the cost of agricultural inputs. Again, it's about making decisions right here.

From the 20th century, thanks to the accelerated development of computing and electronics, the industry underwent an accelerated change towards automation, mechanization gave way to computerized control and the massification of data led to the emergence of new technologies such as machine learning; With electronics, together with computing, technologies such as the Internet of Things (IoT), remote sensing, or robotic platforms, home automation, etc. emerged.

Of course, many of these technologies are applied to agriculture; in particular, the development and improvement of the sensor, acoustic, and visual devices (you can count normal cameras, cameras with bulbs for different spectrums, hyperspectral snapshot cameras, etc.), among others, as well as unmanned vehicles, have facilitated the monitoring of cultivated fields as well as herds, animal farms and others in multiple ways; Besides, modern analysis techniques, such as decision tree models, allow agriculture to be managed more efficiently.

All these changes within the field bring with them many improvements but also quite a few challenges that all those involved in agriculture must face, the most important being farmers and states, who must supply the food demand of their countries, and who must be one of them in constant dialogue to find a point of agreement that directs agriculture constantly towards its improvement and not towards stagnation: “Only if aspects of technology, diversity of crop and livestock systems, and networking and institutions (i.e. markets and policies), are considered jointly in the dialogue, should farming in the digital era be termed “Smart Farming.” (Walter, 2017)

2.2.1. Computer vision for agricultural field

One of the main data yielded by the agricultural field is images since they are the basic method for the physical classification of groceries and agricultural products. Various factors are involved in the process of classification/selection of agricultural products, for instance, among which are the shape, size, color, texture, for fruits, vegetables, plants; the contrast of the color between one section of the image and another, the contrast between the texture of one section or another and the shape of the section serve, for example, they are used to detect any disease present in fruits, stems or leaves; these and other characteristics are also used in contexts such as classifying soils, ecosystems, livestock, as well as diseases, states (ripening, harvesting, diseases, etc.), to

give other examples and many times the factors to be taken into account go beyond that can be counted on the fingers.

Such classification processes, with such several factors involved, enabled thanks to modern technological advances (in sensors) that allow precision and scope when acquiring the necessary information for said processes, make manual development impossible, at least if you are looking for objectivity and infallibility in its results; and often these processes become continuous tasks according to the workflow of the sector involved (for example, farming).

To give us a broad picture of the development of visual computing in agriculture, Bhargava and Bansal (Bhargava, 2018) present a very complete study about computer vision techniques applied in the field of agriculture for evaluating fruit and vegetable quality.

The first stage in this study focuses on describing the image processing technique, application of computer vision; This technique consists of five steps: image acquisition, pre-processing, image segmentation, feature extraction, and classification.

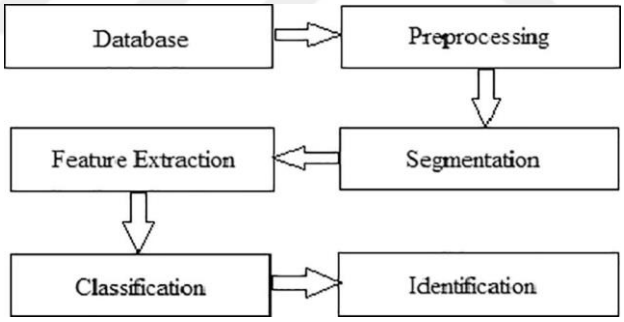


Image 2.17. *General diagram of the process of detecting objects in images. (Bhargava, 2018)*

For the acquisition of images, use is made of visual computing systems that respond to the peculiarities of the problem to be solved; Zhang et al. (Zhang B. a., 2014) describe five fundamental components: illumination, an image capture board(digitizer or frame grabber), a camera, and computer hardware; the image capture board is generally used in-ground capturing process(at crop field), also, the illumination tool can be replaced by the sunlight at in-ground capturing, but some studies as (Koirala A. a., 2019) and (Koirala A. a., 2019) recommends to take images at night with artificial illumination, to control the illumination density, angles, etc. Background control and lighting are of vital importance since they affect the quality of the image, showing shape, color, and texture, as well as

defects, changes, and contrasts within the surface of the object and which must be well distinguishable from what that it would come from being the background in the image (that is why it is used, but it is difficult to get in images taken in crops); light systems are structured as front and backlighting.

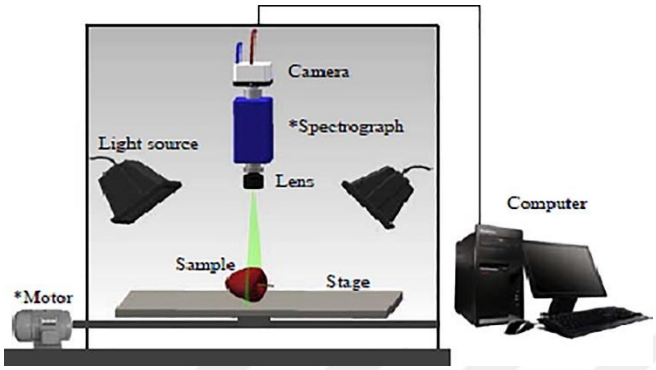


Image 2.18. *General scheme of a visual computing system for agriculture. (Zhang B. a., 2014)*

As hardware to capture the images or data, different technologies are used, among which it can be found cameras, ultrasound, magnetic resonance imaging (MRI), electrical tomography, and computed tomography (CT). Charged coupled device (CCD) and complementary metal-oxide-semiconductor (CMOS) image sensors are used to digital images.

As the captured images continue noise, it is necessary to apply techniques to eliminate it, to remove distortions and better outline the characteristics present in the photo, techniques that are divided into pixel pre-processing, "converts an input image into an output image such that each output pixel is correlated to the input pixel having the corresponding coordinates" (Bhargava, 2018), and local pre-processing (Filtration) "use a small neighborhood of a pixel in an input image to produce a new brightness value in the output image" (Bhargava, 2018).

The next step consists of processing the tunned images to separate the image in different areas to get out the background from the significant areas during the object evaluation: image segmentation; an improper imagen segmentation will affect negatively the performance of the classifier. There are a lot of different techniques that intend to offer a proper imagen segmentation, and the study lists: "The thresholding classify each pixel in the image into two classes i.e. interest area and background area. The pixels with particular gray level belonged to the class of interest area while pixels with the equivalent gray level belonged to the class of background" (Bhargava, 2018) improved my the Otsu

method (Otsu, 1979); the clustering technique comes as a solution of the computational cost of thresholding technique and it has two types: hard, which segments the image based on pixels belonging to identical clusters, and soft, more realistic as noise exact division does not exist in real life; fuzzy clustering techniques are another approaching to image segmentation (Ghabousian, 2012) (Alavi, 2012) as well as the rule-based segmentation method (Jamil, 2014); k-means algorithm (Lee, 2015) (Mehra, 2016), minimum spanning tree algorithm, pattern recognition (Jhawar, 2016), hybrid/mixed techniques among different other techniques are among the many image segmentation algorithms.

To extend the recognition radius the feature vectors are extracted, effective data for image perceptive, interpretation, object classification; these feature vectors define the shape object uniquely and precisely (Bhargava, 2018). In agriculture, three types of characteristics are analyzed: color, morphological and textural features. Color feature, the first and most widely used visual features in image retrieval and indexing, is the most natural way to recognize the current state of fruit, leaf, plant, etc.; HSI space, CIELab space, RGB color space, are commonly used for color analysis, the latter being the most common in image acquisition, requiring various techniques to standardize RGB (since each RGB device produces different RGB combinations, each image of the same reference is different in terms of RGB). Morphological features are shape and size; the shape is a critical visual feature for image content description which cannot be defined precisely because it is difficult to measure the similarity between shapes, it has two categories: region-based (based on the integral area of object) and contour-based (boundary segmented using local features), and they are measured by roundness, aspect ratio and compactness (Bhargava, 2018); the size is quantified by length and width of fruits and vegetables, which are the longest line across the object, that is gotten by the distance of every two boundary pixels, the major axis and the longest line drawn across the object perpendicular to the major axis, the minor axis. For its part, the texture represents the distribution of elements and the appearance of the surface in the images, is useful for predicting surface in the form of roughness, contrast, entropy, orientation, etc.; it can be analyzed qualitatively(contrast, correlation, entropy, and energy) and quantitatively(contrast, coarseness, line-likeness, directionality, roughness, and regularity) (Bagri, 2015); its type are statistical texture, model-based texture, structural texture, and transform-based texture.

The extracted features are used to form a training set to feed a classification algorithm applied to extract the knowledge base which decides in the unknown case. In a computer vision system, a wide variety of methods have been proposed as these classifiers: KNN, SVM, Artificial Neural Network (ANN), Deep Learning(Convolutional Neural Network - CNN). "KNN targeted on the similitude of samples measured by a distance metric. It firstly selects K number of neighbors then based on Euclidian distance in each category, several data point counted and then a new point is elected where the new point can be counted. SVM is a powerful classification algorithm in which both linear and non-linear data is classified. It non-linearly maps data to a high dimensional space by using kernel functions. For 2-class problems, SVM finds the linear optimal hyperplane such that the distance between support vectors (extreme points of both the classes) can be maximized. ANN is the computer programs that are biologically influenced to simulate how the human brain processes instruction" (Bhargava, 2018).

Subsequently, the study recounts different pioneering techniques in the application of ANN as classifiers, the background of the application of deep learning. Since here, the study makes a comparison between more than 40 papers, most of which use traditional techniques, comparing the performance of the proposal in terms of the performance measured in precision, resulting in very little difference between the more complex techniques used; each section presents different tables comparing the reviewed papers about proposals in each computer vision image processing stage and the accuracy obtained by them.

In their study, reviewed in the next section, Kamilaris et Al. (Kamilaris A. a.-B., 2018) list eighteen computer vision applications, traditional techniques, in agriculture; phenology, monitoring, yield estimation, etc. of crops/weeds, pest detection and management, fruits detection, and soil monitoring were found between the main sources of interest; they also found Support-Vector Machines (SVM), Normalized Differential Vegetation Index (NDVI), linear regression analysis, decision trees, and linear polarization as the most popular hand-crafted techniques in agriculture.

2.2.2. Deep learning in Smart Agriculture

Contrary to what one may think, the introduction of deep learning techniques in the field of agriculture does not date back more than three years ago: the earliest report of its application in the field of agriculture dates from 2018 (Kamilaris A. a.-B., 2018) and it

reports 40 published studies taking a very wide deep learning's definition; from here it follows that the number of review papers for deep learning applications in the agricultural field can be counted on the fingers.

These types of reviews are very important to track the development of deep learning in the field of agriculture and among other things, Although the advance of Deep Learning offers more than satisfactory results, traditional techniques are still being used, so first, one of the most current revisions as an introduction is presented, and then review some of the Deep Learning reviews; hence, the present study focuses on reviewing these types of studies specifically.

It was mentioned the review work of Kamilaris et Al. (Kamilaris A. a.-B., 2018): it is an extensive study that tries to cover all the studies on the application of deep learning to agriculture until its year of publication, taking 40 relevant papers, to offer a broad and descriptive vision of the state of the art of deep learning in agriculture.

These forty papers are grouped by purpose (the problem for which a solution based on deep learning is presented), among which it is highlighted: crop/weed detection and classification count with 8 papers, each of which has a CNN developed by their authors, and also with different datasets; fruit counting counts with 4 papers, two of with use Fast R-CNN as deep learning framework; crop yield estimation counts with 2 papers that use a custom author proposed CNNs; plant recognition counts with 4 papers that uses AlexNet CNN and author proposed CNNs; land cover classification counts with 4 papers that use author proposed CNNs and other different computer vision techniques; plant disease detection counts with 3 papers, first one uses CaffeNet CNN, second one tests both AlexNet CNN and GoogleNet CNN and the third one uses LeNet CNN: from this follows the statement that until then, not many authors dared to experiment outside of conventional CNN.

The study also delineates what would be the measures by which the performance of deep learning solutions offered by each paper is evaluated and which would become the state of the art standards: classification accuracy (CA), Intersection over Union (IoU), F1(harmonic mean of precision and recall), Root Mean Square of Error (RMSE) and Mean Residual Error (MRE).

The study also finds several advantages of deep learning compared to traditional techniques: deep learning offers better performance(outperforms) the hand-crafted image processing techniques in terms of processing time and accuracy, it also reduces the effort implementing its solutions, taking less time to implements deep learning solutions(for instance, it takes less time to extract dataset features); deep learning also offers the possibility to simulate datasets to train the model (for instance, data augmentation).

But deep learning has also disadvantages: the most important problem it requires large datasets to train (initially, almond 20000 images would be needed to get a satisfactory accuracy in predictions), even when data augmentation is used, more than a few hundred of images are required to build a fair enough training dataset, depending on the problem's complexity; another problem, which it is verified in the study, it is the very pronounced difference between dataset: for instance, a solo mango trained model cannot predict mangoes in tree canopies; besides, as it is further verified in the present study, finding complete (extensive), publicly accessible datasets, especially those related to tree canopies is a very difficult task.

The study ends by listing some possible fields of study that can be visited by deep learning, such as the identification and classification of seeds, nitrogen content in soil and leaves, irrigation, soil erosion, detection of pests and diseases, identification of effects harmful caused by pesticides, etc .; it also proposes the use of more current deep learning techniques, more complex models and architectures, different methods for obtaining images, among others. Additionally, it indicates that some of the solutions present in the study may have commercial use.

The next year, Koirala et Al. (Koirala A. a., 2019) take the study mentioned above, starting from the considerations proposed by the study and extending it in different aspects. First, it presents a review of the development of deep learning and makes a brief description of the components of a framework for detecting objects (fruits) in general, especially highlighting the role that CNN has as feature extractors, later, it goes on to explain the different Deep learning framework for object detection and its standard classification.

In the fifth section, the study focuses on deep learning model training: generally, the number of images for the data set responds to the complexity of the object to be

detected, each data set is normally divided into three sets (PASCAL VOC style, for instance), training, validation and verification, and each data set have the annotations corresponding to each image in each group, containing the vertices of the bounding boxes of the objects contained in said image. The quality of the training images is an important factor that inside the precision of the predictions of the trained model, the validation set is important to tune parameters within the model that will later be verified by the verification set.

More than the number of images, ranging from between a thousand and two thousand images per data set, what matters is that the set is carefully selected so that it is varied enough to cover all possible cases in the context of the problem that you want to solve (deployment context); the more complex CNNs (longer ones), although more precise, requires a greater amount of images for their training. This study again mentions the lack of publicly accessible data sets, which would facilitate comparison between models, respecting the scientific method, by validating different models with the same data sets, but which, unfortunately, until today are quite a few.

Despite what is thought, the use of transfer learning seems not to be necessary, as several of the papers reviewed report that there was no significant increase in performance in models that used this technique for their training. To end this apathy, although most object detection frameworks handle size standards for training images, this can be changed, at the cost of increasing the computational expense.

The next section starts by reminding us of something implicit in the scientific method but which is often, as mentioned, overlooked: all deep learning-based solutions for image detection should be properly documented and should be publicly accessible, to the extent of which it is possible so that each researcher could replicate these experiments and verify their conclusions or buy their developments.

As the main parameter for performance assessment, the study describes the intersection over union (IoU) parameter, also known as *Jaccard Index* (Jaccard, 1901), defined as the size of the intersection divided by the size of the union of the sample sets(the union area detected and the ground-truth bounding boxes); it used to evaluate the results in the detection of objects, to wit, false positives, false negatives, etc. *Precision* (number of true detection out of total detections) and *Recall*(number of true detection out

of total ground-truth annotations) are useful measures to characterize an algorithm performance on a dataset, being *Precision* sought when false positives have a high cost associated, and *Recall* sought when false negatives have a high cost associated; in general, a balance is always sought between the two measures. *F1 score* (Sasaki, 2007) is the harmonic mean between *Precision* and *Recall,* recommended as an overall performance metric to be a single metric to compare and benchmark models and methods. Besides, the *Average Precision* is used as a single metric to summarize the models and methods performance.

More than comparing different architectures and models on different applications, research studies should try to optimize these models and architectures or offer new alternatives, always well documented in terms of their development and comparison of their performance against existing alternatives, all within order to facilitate its replication. Very few papers report both the training time and the inference (detection) time of the produced models, being also complex to buy since it responds to the type of hardware used for the experiments; Although training time is not of the utmost importance, inference time is vital in terms of real-time solutions, which is why many of the current solutions still lack practical application in real-time.

The study records twelve specific papers on the use of deep learning for the detection of fruit in treetops, being that before 2016 the investigative studies used handcrafted techniques for it. This section describes the studies in which deep learning solutions for fruit detection have been put into practice, most of which implement a certain framework for crops of different types and their performance and usefulness in detecting objects, alternating and/or assisting with various approaches to obtain a complete solution to its purpose (for example, to resolve the occlusion of fruits between the tree canopies, to have a better approximation of the crop production - yield estimation - when counting the fruits automatically employing catches in the open field); here, it describes the different approaches used in each study and how they progressively progress from study to study. For instance, MangoYolo (Koirala A. a., 2019) (Wang Z. a., 2019), redesigns the YOLOv3 architecture, reducing the number of layers, and achieves better performance (memory usage and speed) compared to the YOLO standard, demonstrating that the architecture design inside more in the performance of the models than the number of layers that networks possess.

The study ends by addressing some of the most common problems for the agricultural context that concerns us: fruit yield estimation; among these are the clarity with which the fruits can be differentiated within the tree canopies, something that depends on the angle at which the photo is captured, as well as the angle and position of each fruit, or groups of grottos , the quality of light (artificial or natural and therefore, the time of day) among other factors; the occlusion effect is one of the most affecting the detection of objects, since a fruit, when superimposed on another, totally or partially hides a fruit, making it difficult for the model to correctly detect the fruits within an image, thus affecting the total estimate of agricultural production (obtained by computing the number of total fruits by the average weight of the fruit, which must be chosen carefully); for all these problems, solutions and developments are insight that could enhance the use of deep learning in agriculture in the future.

The last published study (Santos L. a., 2020), in addition to making, once again, a summary of the development and application of deep learning in the field of agriculture, reviews papers related to this context. Initially, there are 43 recent papers related to fourteen study areas, of which only twenty-nine papers are reviewed (list of papers with public access), being disease detection (six papers), plant identification (four papers), identification of weeds and their ground reach (three papers), the most popular study areas for research; besides, CNN solutions are the most widely used (twenty-four papers).

Slowly but surely, the number of papers grows year by year (one paper in 2016, three papers in 2017, eighteen for 2018, eight for 2018). In some cases (12 papers), the authors develop their architectures, although it is not clear if they are modifications of existing ones or if they were developed from scratch. Data augmentation is a common practice to lengthen training data sets. In general, it is found that developments with deep learning perform better than those made with traditional techniques.

It is attested that the complexity of deep learning models imposes restrictions on the hardware necessary to run them: four papers run on CPU while the others turn to the GPU, and even with this additional potential, only eleven papers claim to have performed test operations on real-time; in the other cases, and as usual in these studies, the authors do not provide information on the hardware used, the training and inference time, or the hardware used to acquire the image sets.

In conclusion, the study declares that the implementation of deep learning, booming and with several applications, can be combined with autonomous vehicles, air or land, to automate farm work; however, the restrictions produced by the complexity of the models make it difficult to achieve this goal, since they require great computing power, and currently only GPUs are capable of providing such computing power, at the cost of needing a large amount of electrical power, preventing the use of fully autonomous vehicles, since there are few types of batteries capable of supplying the necessary electrical power.

3. METHODOLOGY

Comparing architectures against different datasets is the main goal of the present thesis, since here, it does not pretend to present an innovative methodology or approaching; instead, the present thesis, then, takes public access data sets derived from the reviewed papers to train deep learning architectures whose models are then evaluated, based on the results of the evaluation and what it is known about these architectures, as well as the results of the reviewed papers, a performance analysis of each architecture for each data set is developed to elucidate which architecture is more suitable for which data set.

Finally, conclusions are presented on the findings and subsequent research paths are suggested for anyone interested in the application of deep learning within the field of agriculture, and specifically, to the detection of fruits and the estimation of production in crops.

3.1. Environment

To begin, and as Koirala et al. (Koirala A. a., 2019) and Santos et al. (Santos L. a., 2020) recommend, the hardware used in experimentation is described, as well as its software; These characteristics are defined by the chosen deep learning architectures, by the time requirements and by the same restrictions imposed by the complexity of the deepening models.

Given the complexity of deep learning models, it becomes unfeasible to train such models using CPU, since they would take weeks or even months to complete their training; That is why it has been decided to choose to use GPU cards to speed up the training time of the models. Likewise, to reduce as much as possible the computational load of the training, updated hardware has been chosen, with a good processor, a high-end GPU card, recurrent in many of the reviewed works, and high RAM.

Most of the deep learning architecture implementations in vogue are implemented in both C ++ and Python, so a friendly Python or C ++ development environment is required, although tools such as Conda or Spider offer complete development environments. For Windows, this system is well known for complicating matters when it comes to native development.

Likewise, these implementations make use of industry-standard libraries, which greatly facilitate their implementation, and which also have their requirements, which are normally base libraries in their latest versions; Because to shorten the time required for each experiment, the most up-to-date implementations have been chosen, it is necessary to have the most up-to-date libraries. These base libraries, CUDA for example, also have their hardware requirements that greatly limit the range of GPU cards that can be used.

Once whole requirements where evaluated, **Google Collaboratory** was selected as the experimental environment since it provides both hardware and software features needed to lead the experiments, as it will be explained in the next two sections.

Indeed, the advice offered by the studies cited above is not fully replicated, whereas the environments reported by the studies whose datasets are used were not possible to be replicated; however, for time issues, putting the reduction of time above these councils was the taken decision for this study; however, what section of the environment consists of is here exactly reported so that subsequent studies can replicate this study's experiments and confront the given results.

3.1.1. Software

Thinking about the ease of implementation of the architectures, a GNU / Linux operating system is chosen to be used as the main environment: Ubuntu 18.04, this version being LTS, and therefore maintaining extended support, libraries as up-to-date as possible but maintaining compatibility (you want to avoid any type error caused by conflicts between libraries). As a base, it has python3, required for the latest versions of python implementations, and GCC, as a native C ++ implementation library.

The Tensorflow (Martín Abadi, 2015) library, used in many deep learning implementations, has restrictions for CUDA versions in its implementation for GPUs (Team, s.f.): 10.1 for Tensorflow versions higher than 2.1.0 (which, although being the most current versions, are already implemented in many architectures, updated), 10.0 for Tensorflow versions 1.13 onwards; CUDA is currently at version 10.2, however, no Tensorflow version implemented so far has support for this version of CUDA.

To avoid conflicts between the different libraries used by Python implementations, the virtual library is used, which is responsible for managing isolated virtual python environments, so that it allows installing the different required libraries in separate

instances, avoiding any type of conflict, and even allowing to install different versions of Tensorflow.

The aforementioned limitation forces us to install, at least to work with CUDA, a different version of the NVidia drivers; other libraries are included, either to further increase the performance of the architecture (decrease training time), which is the case of TensorRT, or to speed up inference and detection of objects, such as OpenCV, processing library of images written in C ++.

Finally, **Table 3.1** shows which software was used to execute the experiments:

Table 3.1. *Software used in the environment:*

Software	Version
Ubuntu	18.4.3 LTS
GCC	7.5.0
Nvidia driver	418.67
CUDA	10.1.243
cuDNN	7.6.5
OpenCV	3.2.0
TensorRT	6.0.1
Tensorflow	2.2.0
Python	3.6.9
Keras	2.3.1

3.1.2. Hardware

The chosen CUDA version imposes hardware restrictions: the latest versions require a computational capacity greater than 3.5, which restricts all available GPUs to those available for five years (Corporation, 2020); NVidia drivers also restrict the possible video cards to be used as well as the number of CUDA cores that each reference has, being that the greater the quantity, the greater the computational capacity they have.

Since the deploying models make use of computational resources to carry out complex mathematical operations, it is necessary to have hardware capable of carrying out the greatest number of operations in the least time, which is why a capable processor is chosen that does not limit or restrict GPU performance; Likewise, although this study aims to work all the time with the GPU, a good RAM card is used as a support for other

tasks that do not depend on the GPU, and as a support for those tasks that do need it when necessary.

As in the previous section, **Table 3.2** reports the hardware used for the experiments:

Table 3.2. *Hardware used in the environments*

Device	Specifications
CPU	Intel(R) Xeon(R) CPU @ 2.30GHz
GPU	Tesla P100-PCIE-16GB, compute capability: 6.0
RAM	13021KB

3.2. Architectures

3.2.1. YOLOv3 and YOLOv4

Darknet (Redmon J. , 2013 - 2016) is an open-source neural network framework written in C and CUDA(it uses Tensor Cores). This is the base model used by the papers related to the YOLO framework (Redmon J. a., 2015) (Redmon J. a., YOLO9000: Better, Faster, Stronger, 2016) (Redmon J. a., 2018) (Bochkovskiy, 2020), most of them developed by Redmon et al.

Darknet is written entirely in C, it supports both CPU and GPU computing, although it was not originally compiled with OpenCV, it is possible to compile it with OpenCV and CUDA (if you want GPU support); initially, it can only be compiled for Linux. Darknet can be configured in multiple ways, both for the network architecture and for its installation and execution: it has tutorials to compile both GPU and CPU with different options that enable compiling with different libraries, also, it allows generating different architectures network from configuration files, whose templates are available in your repository.

However, its original authors discontinued its development and there has been no news of it for two years. Fortunately, Bochkovskiy et al. (Bochkovskiy, 2020) have continued their development since then (fork) offering different innovations and solving some problems that made the original implementation obsolete and incompatible with the most current machine learning libraries.

This fork adds support for compiling to Windows, support for the latest version of CUDA (10.x), as well as OpenCV (3.x and 4.x), GCC, and CLANG and MSVC

(2015/2017/2019). It also adds support for different State-of-Art models: CSP, PRN, EfficientNet, new layers to the neural network architecture, the ability to train recurring models for accurate video detection, data augmentation, different activations (SWISH, MISH, NORM_CHAN, NORM_CHAN_SOFTMAX), the ability for training with GPU-processing using CPU-RAM to increase the mini_batch_size and increase accuracy (instead of batch-norm sync), correct calculation of **mAP**, **F1**, **IoU**, **Precision-Recall**, and the calculation of anchors for training and drawing of the chart of average-Loss and accuracy-mAP.

This fork improves the binary neural network performance 2 to 4 times for Detection on CPU and GPU, the neural network performance approximately 7% by fusing 2 layers into 1, the detection performance twice on GPU Volta / Turing (Tesla V100, GeForce RTX, etc.) using Tensor Cores, performance approximately 1.2 times on FullHD and approximately 2 times on 4K for detection on the video (file/stream), the performance 3.5 times of data augmentation for training (using OpenCV SSE / AVX functions instead of hand-written functions) and the performance of detection and training on Intel CPU with AVX (Yolo v3 approximately 85%); it also optimizes memory allocation during network resizing, GPU initialization for detection.

The repository offers tutorials where they explain how to compile the architecture for both Linux and Windows with different libraries (CMake and Make for Linux, CMake-GUI, vcpkg and Visual Studio for Windows), how to train with GPU and multi GPU, as well as to detect Own objects (instead of using state-of-the-art datasets, to wit; Microsoft COCO (Lin T.-Y. a., 2014), ImageNet, etc.), how to use other models, improve object detection and evaluate network performance.

Regarding the format, YOLO uses its format, which consists of files in text format with the name of the image where each line represents an object located in the image; each line contains the number (integer) corresponding to the class, as well as the center point coordinates and the normal width and height (floating-point values between 0 and 1). Regarding the training, validation, and testing subsets, one file for each must be created, containing the absolute path of each image. The folder does not have a defined structure, all files can be contained in the same folder, although for order it is preferable to put the images and annotations in separate folders; Darknet uses configuration files to indicate where the annotations, images, and subset files are (**Figure 3.1**); a *names* file is

used to store the labels of the objects to train, one per line, indicating the number of that class (integer starting at 0).

```
classes= 2 # number of classes
train  = data/train.txt # path to the training file
valid  = data/test.txt # path to the validation file
names = data/obj.names  # path to the object names file
backup = backup/     # path to the folder where model weights are
                     # saved every certain number of batches before
                     # obtaining the final model.
```

Figure 3.1. *Obj.data containing the training data, where classes' variable is the number of object's classes.*

The repository offers a tutorial to configure the network to train with its objects, as well as a script written in python to convert annotations in PASCAL VOC 2012 (and later) format to YOLO format (**Appendix E**).

3.2.2. RetinaNet

Based on the Lin et al. (Lin T. a., 2020) described object detector, Fizyr's Retinane Net implementation is built over the *Keras* python library, thence, this implementation is full written on python; it currently has support for *Keras* 2.3.x and *Tensorflow* 2.1.x...

This implementation has many of the features present in the state of the art implementations: support for different backbones (architectures, being *ResNet50* (He K. a., 2016) default backbone), support for CPU and GPU calculation, support for multiple annotation formats (Microsoft COCO (Lin T.-Y. a., 2014), PASCAL VOC, free formats, etc.), support for transfer learning, debugging, inference and mAP, precision and recall values calculation (although it does not have the F1 score calculation).

This implementation works training and inference models separately, to wit, the training models are shortened versions that only contain the necessary training layers (regression and classification values), while the inference models include all the necessary capabilities to detect objects; thence, it is necessary to convert the training result model to an inference model to evaluate the model.

The repository has tutorials to install the implementation, evaluate, train, convert training models to inference models, and train with its formats, as well as some answers to common questions about the implementation.

3.3. Datasets

Since our objective is to compare, based on already existing results, using existing and publicly accessible data sets is chosen, intending to corroborate the previously established results and compare them with the results in this study. For this, the different review papers are visited and it was tried to gain access to their datasets, and, like Kimlaris et al. (Kamilaris A. a.-B., 2018) and Koirala et Al. (Koirala A. a., 2019), It was impossible to gain access to many of the data sets.

Despite this, Koirala et al. (Koirala A. a., 2019) offer a list of papers among which five sets of public access data could be located, in image 3.1 you can see the list of said papers. From this list, the data sets belonging to Bargoti and Underwood (Bargoti S. a., 2017), Kestur et al. (Kestur, 2019), and Koirala et al. (Koirala A. a., 2019) were obtained. Remaining papers do not have their datasets available to the public, or they are not minimally easy to access except by Liang et al. (Liang, 2018), which dataset is public by Baidu, the Chinese website, unfortunately, requires a number Chinese cell phone for registration, making it not accessible for this study.

Author	Commodity	Method	Validation statistics
Sa et al (2016)	Sweet pepper	Faster R-CNN	F1 0.84
Bargoti and Underwood (2017a)	Apple, mango, almond	Faster R-CNN ZF	F1 0.89, 0.88, 0.73
	Apple, mango, almond	Faster R-CNN VGG	F1 0.90, 0.91, 0.76
Bargoti and Underwood (2017b)	Apple, mango	Pixel wise CNN	F1 0.86, 0.84
Chen et al. (2017)	Apple, orange	FCN + CNN regression	Ratio counted 0.91, 0.97
Choi et al. (2015)	Citrus (classification)	AlexNet	TP rate 96%
Habaragamuwa et al. (2018)	Strawberry	CNN	AP 88.0%
			Bounding Box Overlap 0.74
Liang et al. (2018)	Mango	SSD VGG	F1 0.91
Peng et al. (2018)	Apple, litchi, navel orange, Huangdi gan	SSD ResNet	F1 0.96
Tao et al. (2018)	Peach, apple, orange	Faster R-CNN	AP 91.5, 94.2, 90.2%
Xiong et al. (2018)	Green citrus	Faster R-CNN	F value 77.5%
			mAP 85.5
Koirala et al. (2019)	Mango	Faster-RCNN ZF	F1 0.94
		Faster-RCNN VGG	F1 0.95
		SSD VGG	F1 0.96
		MangoYOLO	F1 0.97
Kestur et al. (2019)	Mango	CNN	F1 0.84

Image 3.1. *Table of reviewed papers. (Koirala A. a., 2019)*

The list of data sets is completed with Santos et al. (Santos T. a., 2020), a paper that was found while doing the dataset collection, and the literary review that supports this study. From now on each one of the acquired data sets will be explained, their characteristics as well as the processes necessary to implement them, whenever necessary.

3.3.1. ACFR Orchard Fruit Dataset

Throughout several of their different papers, Bargoti et al. Have worked with this dataset, with which they have tested the different stages of the development of their proposals in the application of deep learning to agriculture. For this study the 2017 paper

(Bargoti S. a., 2017) is referred, in which it reviews the applications of deep learning to agriculture, in the context of fruit detection in orchards, to propose the Fast R-CNN architecture as an object detector for the detection of fruits in crops.

This dataset contains images, with their respective annotations, for three fruits: apples, mangoes, and almonds; The photographs were collected by the agriculture team at the Australian Center for Field Robotics at The University of Sydney, Australia, at different farms across Australia. **Table 3.3** details the information on each data set, including the type of image, the type of annotation applied, and notes on the variety of the fruit and the place where the photographs were taken.

Image 3.2. *An example of the images belonging to the ACFR Almonds dataset.*

Image 3.3. *An example of the images belonging to the ACFR Apples dataset.*

Image 3.4. *An example of the images belonging to the ACFR Mangoes dataset.*

The structure of the data set responds to the PASCAL VOC 2007 (Everingham, 2010) format, that is, each fruit has its directory, inside which are the directories for the images, the annotations and one that contains the different files that list the different types of sets, to wit, training, validation, and test.

The structure of the data set responds to the PASCAL VOC 2007 (Everingham, 2010) format, that is, each fruit has its directory, inside which are the directories for the images, the annotations, and one that contains the different files that list the different types of sets. , to wit, training, validation, and test. The annotation files are in comma-separated values (CSV) format, the names of the respective images containing one image per line with their respective annotations; apple dataset images are in Portable Network Graphics (png) format.

As for the annotations, these were taken using the *PychetLabeller* program, both Almond and Mango datasets have rectangular annotations, while apples have circular annotations due to their natural shape; the apple dataset also contains additional pixel-level annotations.

Table 3.3. *ACFR multifruit dataset summary. (Bargoti S. a., 2016)*

Fruit	Image Info	Annotation Info	Notes
Apples	1120 Images PNG Image Data 308 x 202, 8-bit/color RGB Sensor: PointGrey Ladybug3	Circle Annotation (x,y,radius) Pixel-wise annotation	Collected at Warburton, Australia Apple varieties: Pink Lady, Kanzi
Mangoes	1964 Images PNG Image Data 500 x 500, 16-bit/color RGB Sensor: Prosilica GT3300c Strobes Used	Rectangle Annotation (x, y, dx, dy)	Collected at Bundaberg, Australia Mango varieties: Calypso
Almonds	620 Images PNG Image Data 308 x 202, 8-bit/color RGB Sensor: Canon EOS60D	Rectangle Annotation (x,y,dx,dy)	Collected at Mildura, Australia Almond variety: Nonpareil

Given that the standard format implemented by the selected architectures is PASCAL VOC 2012 (Everingham, 2012), as it was already mentioned before, these data sets required "reconversion", or update, of their format to the most current:

- The annotation format has changed both in its type and in its description: files are now handled in Extensible Markup Language (XML) format containing the source directory (dataset folder), the image name, the image path, the source of the image (most of the time unnecessary), image size (width, height, and depth), and an outline consisting of the class label, other auxiliary labels, and the boundary box coordinates for each object annotated in the image, as you can see in the **Image 3.2.**
- In the previous format, the origin coordinates (upper left corner) plus a delta (distance from the origin point to the lower right coordinate) were used to define the boundary box. This new format chooses to have both the origin coordinate and the final coordinate, without using deltas.
- The current format does not have some kind of circular annotation.

Given these restrictions, a conversion algorithm is developed as a *python* script, visible in **Appendix A** and which follows the following scheme:

- The structure of the *CSV* file is previously defined based on the shape of the fruit, circle, or square.
- *Python* goes through each of the directories looking for the annotations folder and the images folder.
- Depending on whether the file is square or circle, the *Pandas* library is used to load the file with the respective *CSV* heads.
- Both the image file name and the final annotation are deduced from the path of the current directory.
- Using *OpenCV* you get the width, height, and depth of the image.
- The python *XML* release would be used to build an XML tree which will then be saved.
- Each of the labels required by the format is created and filled with the variables obtained previously, calculating the final coordinates for each rectangular annotation by adding the deltas to the initial coordinate and calculating the initial and final coordinates of the circular annotations by subtracting or adding the radius of the circle to the coordinate of the center as appropriate.
- Finally, the XML tree is saved as a file in its respective directory.

After this process, the original files are removed to avoid wasting unnecessary disk space.

```
<annotation>
        <folder>Kangaroo</folder>
        <filename>00001.jpg</filename>
        <path>./Kangaroo/stock-12.jpg</path>
        <source>
                <database>Kangaroo</database>
        </source>
        <size>
                <width>450</width>
                <height>319</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>kangaroo</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>233</xmin>
                        <ymin>89</ymin>
                        <xmax>386</xmax>
                        <ymax>262</ymax>
                </bndbox>
        </object>
</annotation>
```

Image 3.5. *PASCAL VOC 2012 (Everingham, 2012) annotation format. (Everingham, 2012)*

The second drawback is that the implementations of the architectures used require by default to know the extension of the image file, which by default is Joint Photographic Experts Group (jpeg), while the images in this dataset normally have png extension; to overcome this, some changes have been made, reported to the original repository as well, that enables passing the image file extension as a parameter to the evaluation and training script, as appropriate.

3.3.2. MangoYOLO dataset

Koirala et al., in his MangoYOLO benchmarking (Koirala A. a., 2019), his deep learning proposal for crop yield estimating, presents his dataset of mangoes. The images were collected from all the trees of five (*Mangifera indica*) orchards (farm management units) located in Queensland, Australia, being varied in the cultivar and canopy structure, after maturation and 6 weeks before harvest. The photos were taken at night, starting the process after sunset, thus allowing consistent lighting conditions, since daylight is highly variable due to ambient conditions and the angle of the sun.

This dataset is in PASCAL VOC 2012 (Everingham, 2012) format, already described before, so, inside the annotations folder there is an *XML* file for each image in the images folder; each annotation file has the name of the image, the absolute path of the image, its attributes (depth, width, and height) as well as the coordinates (initial and final) of the boundary boxes for each object in the annotated image; therefore, this dataset does not require converting the annotated files. The annotations were captured using *labelImg.*

Image 3.6. *An example of the images belonging to the Mango YOLO dataset.*

It also contains three defined files: training, validation, and testing. The scheme used in the study (**Table 3.4**) is not followed, the standard sets for the experiments lead the carried out experiments in the study. Being that most frameworks re-scale the images to a specific resolution when training, usually square, to decrease computational costs, and taking the network stride, the factor by which network scales down the input image to its final feature map, for different frameworks, the images, with 2448 × 2048 pixels original resolution, were divided into smaller images of 612 × 512 pixels, a grid of 4 × 4, maintaining the aspect ratio of the original image, to have an object size greater than the stride size of the final feature map.

Table 3.4. *Description of image datasets. (Koirala A. a., 2019)*

Dataset names	Number of images or tiles	Image size (pixels)	Total # fruits in	Cultivar
Train set 1	1 300	612 × 512	11,820	Calypso
Validation set 1	130	612 × 512	861	Calypso
Test set 1 All	300	612 × 512	2 600	Calypso
Test set 1 Low	100	612 × 512	341	Calypso
Test set 1 Medium	100	612 × 512	636	Calypso
Test set 1 High	100	612 × 512	1 623	Calypso
Test set 2A	34	2 448 × 2 048	2 151	Calypso
Test set 2B	12	2 448 × 2 048	964	Calypso
Test set 2C	24	2 448 × 2 048	842	Calypso
Test set 2D	36	2 448 × 2 048	1 229	Honey Gold
Test set 2E	36	2 448 × 2 048	552	R2E2
Test set 2A-can	34	6 000 × 4 000	2 137	Calypso

Test set 2A-kin	34	1 920 × 1 080	1 746	Calypso
Train set 2-bu	1 154	500 × 500	7 065	Calypso
Test set 3-bu	270	500 × 500	947	Calypso

3.3.3. Embrapa Wine Grape Instance Segmentation Dataset (WGISD)

Santos et al. (Santos T. a., 2020) offer a publicly accessible cloud-hosted repository (Github) with detailed information, structured in the way that Crawford et al. (Crawford, 2018) recommend in their study and supported by the Embrapa SEG Project 01.14.09.001.05.04, *Image-based metrology for Precision Agriculture and Phenotyping*, and the CNPq PIBIC Program (grants 161165/2017-6 and 125044/2018-6).

The set consists of RGB images with their respective annotations, which contain at least one grape cluster located within boundary boxes; a subset of these also have binary masks identifying the pixels belonging to each grape cluster; any grape cluster located far in the background is ignored (no boundary boxes, and therefore classified as background).

The images were collected in the vineyards of Guaspari Winery, located at Espírito Santo do Pinhal, São Paulo, Brazil, after the first year of harvest and for production, thus having canopies of lower density, in April 2017 for *Syrah* and in April 2018 for the other varieties. "A Canon EOS REBEL T3i DSLR camera and a Motorola Z2 Play smartphone, were used to capture the images. The cameras were located between the lines of the vines, facing the vines at distances around 1-2 meters. The EOS REBEL T3i camera captured 240 images, including all *Syrah* pictures. The Z2 smartphone grabbed 60 images covering all varieties except *Syrah*. The REBEL images were scaled to *2048 X 1365* pixels and the Z2 images to *2048 X 1536* pixels. More data about the capture process can be found in the Exif data found in the original image files, included in the dataset." (Santos T. a., 2020)

Image 3.7. *An example of the images belonging to the WGISD dataset.*

In total, there are 300 images with 4432 grape clusters located in boundary boxes, a set of these, 137 images, contain binary mask identifying the pixels of each cluster; **Table 3.5** presents an overview of the dataset.

Table 3.5. *Dataset's general description: varieties, number of images, boxed, and masked clusters.*

Prefix	Variety	Date	Images	Boxed clusters	Masked clusters
CDY	*Chardonnay*	2018-04-27	65	840	308
CFR	*Cabernet Franc*	2018-04-27	65	1,069	513
CSV	*Cabernet Sauvignon*	2018-04-27	57	643	306
SVB	*Sauvignon Blanc*	2018-04-27	65	1,317	608
SYH	*Syrah*	2017-04-27	48	563	285
Total			300	4,432	2,020

The dataset is in YOLO format, which consists of a text format file for each 8-bit RGB image containing one boundary box per line, each line has the class of the identified object (identified by a number, which in this case will always be 0), the coordinate of the center point of the box (x, y) and the width and height of the box, each value separated by a space; these values, likewise, are normalized, so they vary between 0 and 1, and are floating-point type; all these files are in the data folder.

In the same folder, the binary mask is found as files in *Numpy*'s compressed array (*.npz*) format; each array is a *H X W X n_clusters* three-dimensional array where *n_clusters* is the number of grape clusters observed in the image; assigning the *NumPy* array to a variable *M*, the mask for the i-th grape cluster can be found in M *[:,:, i]*. The i-th mask corresponds to the i-th line in the bounding boxes file. Likewise, the images in the original resolution (jpeg format) are located within the same directory.

Although the dataset format is perfect for training the YOLO architecture, it is not perfect for other architectures, as it is not a standard in the field; therefore, for the present study, several changes have been made to adapt the dataset to the PASCAL VOC 2012 format.

To start, three different folders are created: *Annotations*, *JPEGImages,* and *ImageSets* (with the *Main* subfolder); all images present in the *data* folder are moved to the JPEGImages folder, all files that describe the train and test sets are moved to *ImageSets/Main* (no validation set is present).

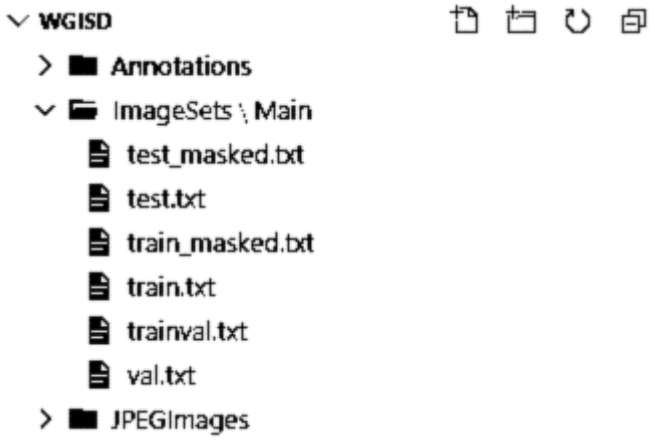


Image 3.8. *WGISD dataset in PASCAL VOC 2012 format.*

Then, similar to the update script used in the ACFR dataset, a format conversion script is used to convert all annotations in text format to PASCAL VOC 2012 *XML* format; it is visible in the **Appendix B** and which follows the following scheme:

- As each line in the *txt* file is in YOLO format, the header for the *txt* file as *CLASS, CX, CY, W, H* is defined.
- *Python* goes through the *data* folder looking for the annotations folder and the images folder.
- *Pandas* library is used to load the file with the predefined header.
- Both the image file name and the final annotation are deduced from the path of the current directory.
- Using *OpenCV* you get the width, height, and depth of the image.
- The python *XML* release would be used to build an XML tree which will then be saved.
- Each of the labels required by the format is created and filled with the variables obtained previously, calculating the final coordinates for each boundary box multiplying the width and height of the annotated boundary box, as well as the central coordinate (x, y), by the width and height of the image (denormalizing), and then, subtracting or adding half the height or width of the boundary box obtained as appropriate.
- Finally, the XML tree is saved as a file in its respective directory.

After this process, the original files are removed to avoid wasting unnecessary disk space.

Another problem with this dataset is that it lists the names of the images in the training and test files and the architectures usually receive absolute paths for ease; to solve this problem, another script (**Appendix C)** is built, which:

- Differentiate the path for the main folder and the images folder
- Read the training and test files.
- It runs line by line of the file, adding both the absolute path and the image extension.
- Save the resulting files.

As it is previously mentioned, the dataset does not contain a validation file, useful to balance the training of different neural networks, so a python script has also been built to generate a validation file from a training file following the 70% rule - 30% (within the state of the art, it is considered important to divide the training set between validation and training at a ratio of 30% - 70% respectively, to avoid overfitting). Thus, the algorithm (**Appendix D**) used consists of:

- Read the training file.
- Convert the file into lines.
- Randomly select a certain number of lines.
- Return the selected set and the resulting set of subtracting the selected lines randomly.
- Save each set of lines as separate *txt* files.

With these three processes, a data set compatible with the architecture standard is obtained, it will be used with RetinaNet, leaving the original dataset for YOLO.

Finally, **Table 3.6** presents the number of images for each type of experiment, training, and test, of each set of data previously exposed.

Table 3.6. *Datasets summary, totally they are five different datasets, for four different fruits.*

Dataset	Resolution	Training	Validation	Test	TOTAL
ACFR Almond	300 × 300	420	100	100	620
ACFR Apple	308 × 202	896	112	112	1120
ACFR Mango	500 × 500	1464	250	250	1964
MangoYOLO	612 × 512	1300	130	300	1730
WGISD	2048 × 1365	170	72	58	300

3.4. Training

Now that the resources to be used in this study are detailed described, the training process for each of the selected frameworks is explained, and it consists of provisioning the machine, provisioning the datasets, processing them if required, installing the frameworks, and finally training the neural network.

To begin with, since it was decided to work with **Google Collaboratory**, the provisioning process is easier: **Colab** offers software for machine learning on relatively capable hardware, for this, it offers on-demand instances (virtual machines), for which it is only needed to update some libraries when required; in our case, it only requires installing **Tensor RT** to improve training performance, the other libraries are updated from the base, and were listed in previous sections. A single library is necessary, ***gdown***, to acquire the previously processed data sets, since they have been hosted on **Google Drive**.

In the case of the YOLOv3 / v4 architecture, the implementation tutorial is used as a base as follows:

- The project repository is cloned.
- Enable options to compile with GPU, using CUDA and OpenCV:

```
GPU=1
CUDNN=1
CUDNN_HALF=1
OPENCV=1
AVX=0
OPENMP=0
LIBSO=0
ZED_CAMERA=0 # ZED SDK 3.0 and above
ZED_CAMERA_v2_8=0 # ZED SDK 2.X
```

Figure 3.2. *Makefile first lines: as the tutorial says, this enables Darknet to work with GPU by using CUDA.*

- Using the configuration file for yolov3 provided by the repository as a base, make the necessary changes to set the network for training one object, as it is mentioned before.

```
batch=64
subdivisions=16
...
burn_in=400
max_batches=5200

steps=3800
scales=.1
...
# For each convolutional layer
[convolutional]
size=1
stride=1
pad=1
# filters = (num/3) * (5+classes)
filters=18
...
#For each YOLO layer
[yolo]
classes=1

ignore_thresh = .9
iou_normalizer=0.3
iou_loss=giou
...
[upsample]
stride=4

[route]
layers = -1, 11
```

Figure 3.3. *YOLOv3 configuration file changes: setup for training with just one object.*

- Compile and get the *Darknet* executable.
- A dataset folder is created inside of *Darknet* folder.

- Datasets are housed in the dataset folder using the *gdown* library to acquire those hosted on Google Drive. The original WGISD dataset is cloned from its repository.
- A *names* file is created by each one of the datasets (five) with the class name (object) inside each one respectively.
- The conversion file from PASCAL VOC 2012 format to YOLO format is acquired from GoogleDrive as well and is executed to obtain the necessary files for the YOLO format (in four of the five data sets).

The logic behind this configuration is simple: in the repository, it is recommended to run 4000 to 5000 batches per object with which you want to train the neural network, in our case there are five objects, with which it would have to run 25000 batches to be able to detect five objects; this process is computationally long and could take about a week to complete; for the convenience of the training time, and because the goal is to evaluate how the architecture behaves for each object, and not for five objects, the model is trained once per object and the weights for each one are obtained, saving configuration time and of training.

- A *data* file is created for each object to classify, it contains the information about the number of classes to train (1 as it is previously mentioned), the location of the training file, as well as the validation file and the *names* file, finally, the address of the directory, where a backup copy of the weights will be kept, is also included.

```
classes= 1
train = /content/darknet/dataset/VOCDevkit/VOC2007/train.txt
valid = /content/darknet/dataset/VOCDevkit/VOC2007/val.txt
names = /content/darknet/dataset/m.names
backup = /content/darknet/backup/
```

Figure 3.4. *Data file example: MangoYOLO dataset, for instance.*

- A *"models"* folder is created to gather all resulting model weights.
- The initial *Darknet*53 architecture weights are gotten as *Darknet* requires an initial model to be trained.

- For each of the data sets, the initial time and the final execution time is captured to obtain the training time.
- For each training, the results of each batch to process are collected for further analysis.
- Finally, all the models are collected and transferred to Google Drive.

The process to train RetinaNet is similar, although it changes in its implementation and the configuration of the training execution, which is described below:

- After cloning the repository, the requirements provided by the requirements file are installed.
- Keras is updated to a recent version, required by the Tensorflow 2.1.x version.
- The framework is built.

In the execution of the training the following parameters are used: enable random transformation, batch size to eight, 500 steps per batch, number of times to ten, the annotation format (PASCAL), and the directory where the data sets are located. Otherwise, both processes are similar: the experiments are executed per object and the results of the execution are recorded in a file for later analysis, the start and end time are also collected, the resulting weights are stored in Google Drive.

Since, as Koirala et al. (Koirala A. a., 2019) say, a greater number of layers in architecture does not mean greater precision and a higher computational cost, *ResNet50* and *Darknet53* are used as the framework's backbone, so the number of layers present in these architectures is light enough to save computational cost but robust enough to achieve high precision. Likewise, since Koirala et al. (Koirala A. a., 2019) report that transfer learning does not provide greater improvement in model training, untrained models have been used (both *ResNet50* and *Darknet53*).

3.5. Testing

The evaluation process is equally simple: the trained models are used as input for executing the evaluation function offered in each framework to obtain *mean Average Precision* (mAP) and their *harmonic mean* (F1), the standard state of the art measurements, and the inference time for each classified object using the evaluation

subsets defined by each dataset (as it is described in their respective section). As with training, evaluation logs are collected for later analysis.

The value of the Intersection over Union (IoU) is set to 0.3, since, as it is verified with some previously done experiments: the value of the standard measurements decreases as the value of the intersection over union (IoU) increases (Santos T. a., 2020).

The command to run the evaluation in *Darknet* is "*map*", and before evaluating with *YOLOv3*, it is necessary to change the validation file to the training file for the value *valid* in the *data* file. The new version of *Darknet* integrates a value for the test directory. This, as previously mentioned, has implementations for standard measures and also includes inference time, offering a complete description of the evaluation results. Because virtual instances on the network were used, and therefore do not have access to other functionalities of a personal computer, this study was unable to obtain the native training graphics that *Darknet* also offers.

For its part, the *RetinaNet* implementation does not offer an implementation to obtain the *F1 Score* and the inference time, so it has been necessary to implement it manually. Before evaluating, the model was converted to an inference model (required to be used for detection, as mentioned in its respective section) and the folder where the processed images are saved is declared.

All notebooks containing the training and evaluation process are available in the same repository with the rest of the material generated in this study, and will be in the public domain once the necessary permission is obtained; with this, its revision, correction or improvement will be facilitated.

3.6. Logs

Both Darknet and Tensorflow have their registration formats, to analyze the corporation of the models during their training, a series of scripts have been made to capture the information present in the logs taken during the training and capture them in useful graphs for subsequent analysis.

The logs were captured from the shell, as these logs are printed in the terminal, just using the *> file* command to capture all the useful information provided by the framework and save it in a plain text file with its respective name, to have order in the logs and identify which file corresponds to which object and framework.

In the case of YOLO, two *python* scripts have been built: per instance and global; the first one receives as input the file name and the number of batches to be plotted, this for illustrative purposes to better understand the training behavior; the second receives as input parameter the name of the folder that contains the files, and also the number of batches to be plotted, with this file visibly comparing the training behaviors of the framework concerning each object is enabled.

Appendix F and **Appendix G** correspond to scripts built to the graph by instance and globally respectively, they are built simply in *python* using the *matplotlib* library to graph data from data arrays with the required information extracted from the logs.

For RetinaNet, the matplotlib library has also been used; the *Tensorflow* library has its record format: it reports both steps and epochs, according to the training parameters of the model, also, it not only reports the average loss, but also loss, regression loss, and classification loss; for which a script to graph (**Appendix H**) these four values by dataset: the last three are included in a graph to determine their behavior, average loss appears in the summary graph, and it is graphically reversed, in reality, the graph shows the precision growth (although it is called average loss) during training. With these four measures, a global graph comparing each dataset concerning each measure is built, to contribute a little more to the analysis of performance, taking advantage of the logs offered by *Tensorflow*.

The scripts used for these charts are available along with all the research material in the same repository.

4. RESULTS

In this section, the results of the different experiments carried out in the research are presented and organized as follows: the first three sections offer a detail of the performance results for each framework, to wit, RetinaNet, YOLOv3, and YOLOv4, consistent in a performance graph for each data set and a global comparative graph of the framework and its respective description; The final section is a summary of the experiments and includes a comparison of the different frameworks against the different data sets, training time and inference, and the weight of the final trained models.

4.1. RetinaNet

The first experiment carried out was RetinaNet due to the relative simplicity of its user interface (implemented through python) and its logic (done in Keras / TensorFlow), in addition to not requiring a specific format for the data set, which greatly facilitates its use.

The parameters used to execute the training, as mentioned before, were: batch-size is eight, steps are 500, epochs are ten, giving a total of 5000 iterations, and random-transformation, to enable random transformations for data augmentation; the experiments were carried out in the following order: ACFR (Almonds, Apples, Mangoes), MangoYOLO and WGISD. RetinaNet makes it easy to validate the training to TensorFlow: depending on the annotation format, RetinaNet passes TensorFlow the validation dataset (*val* dataset in PASCAL VOC) so that it validates the training at the end of each epoch.

In general terms, the training time for each of the datasets is acceptably short, in terms of average computational power, except, as it will be seen, for WGISD, which throughout all the experiments is the data set that training requires; equally, it is for the inference time. In terms of the weight of the generated model, once transformed into inference, which is important for the industry is also, acceptably, small. The conclusions based on these results are discussed in their respective section.

Based on TensorFlow, RetinaNet's training records offer the following information: loss, regression loss, and classification loss, and RetinaNet itself offers the mAP and F1 for each epoch; taking this information as a basis, a graph for mAP and another for all types of loss were constructed.

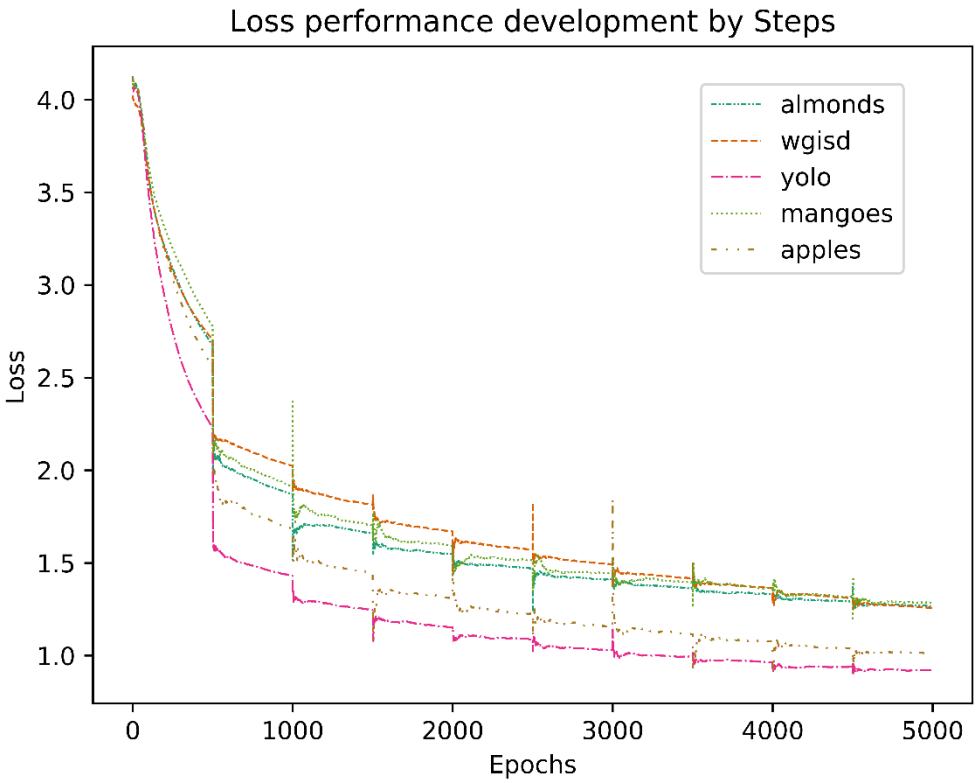


Figure 4.1. *Comparison between training losses across all datasets.*

ACFR-Almonds is the initial analyzed dataset**;** the first thing that can be seen in **Figures 4.1, 4.2, and 4.3.** is that all the losses show more or less similar behavior and start with different values. More or less until iteration 500, the first epoch, the curve of each one descends in a greater proportion, for the beginning of the second epoch there is an abrupt decrease, followed by a decrease, this minuscule time, until the following epoch; This behavior, abrupt decrease or increase followed by a stable but minuscule decrease, will appear in each of the times until the end of the training.

Looking at the graph, it can be inferred that the highest degree of adjustment of the loss in training, within the RetinaNet framework, for this first case, always occurs in the first period, after which it stabilizes and the adjustment gradually decreases, although in much lesser measure.

Likewise, in the mAP graph (**Figure 4.4**), it can be seen that the most pronounced growth occurs from the first to the second epoch (first thousand iterations), going from 0.5767 in the first epoch to 0.7224 in the second epoch; later it suffers an increase, although not very pronounced, until the seventh epoch, after which it suffers a

considerable, although not significant, decrease in the eighth epoch, to increase again and reach its maximum point in the tenth epoch.

For apples, it is seen behavior similar to the previous one: during the first epoch, the highest rate of decrease in loss is seen, followed by an abrupt growth or decrease at the beginning of the following epoch that then continues with a slight decrease, and so on.

However, there are two remarkable things here: the first, at the beginning of the third epoch, the decrease, although it is abrupt, the behavior observes during the initial iterations, at least one hundred, until it reaches stability by decreasing continuously; the second, in the sixth epoch, the abrupt growth is notably greater than the decrease or growth that occurred in the other epochs, in at least 100%, followed by a brutal decrease, which continues for at least 50 iterations until it undergoes growth, to then stabilize in decline again; these behaviors occur only in loss and regression loss.

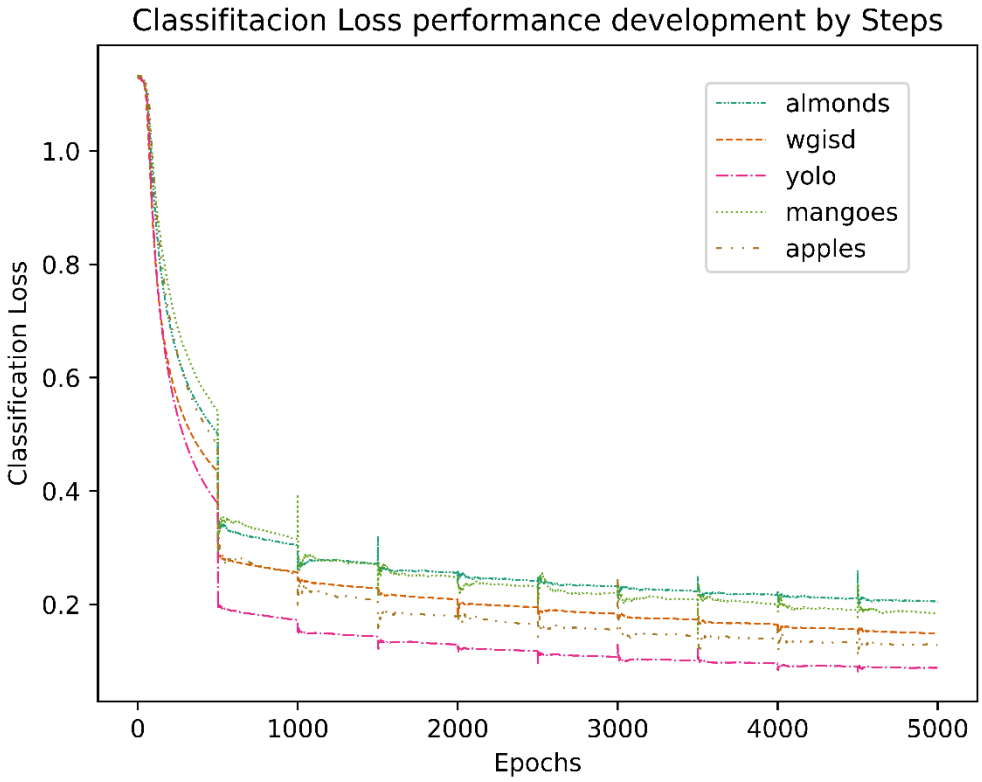


Figure 4.2. *Comparison between training classification loss across all datasets.*

As for the mAP(**Figure 4.4**), its behavior is equally similar: from the first to the second epoch its greatest growth occurs, followed by a rather constant growth, although its maximum occurs in the ninth epoch, to decrease slightly in the final epoch.

Regarding Mangos, equally, the behavior is always similar to the previous ones, however, in the second epoch an event similar to the one described above occurs: there is an abrupt increase in the loss, both in loss and regression, from at minus 300% (compared to the other changes that are much smaller, hence it is quite remarkable), to fall and then increase slightly until finally stabilizing.

Similarly, regarding the mAP, the behavior is similar, always increasing until the ninth epoch, its maximum, and then decreasing very slightly (much less significant than for apples), in the final epoch.

In MangoYOLO, it is seen a stable behavior in losses, as it is already described before, without anything worth noting. As for mAP, its behavior is similar to that of ACFR-Apples, growing continuously until the eighth epoch, decreasing in the ninth epoch, and reaching its maximum value in the tenth epoch.

Finally, regarding the WGISD dataset, the behavior in losses is the same described above, although it registers an abrupt growth of at least 100% (compared to the others) at the beginning of the fifth epoch, in the same way, that is, with the same behavior, described above (remarkable cases, for example, in apples).

Regarding mAP, a very slight decrease can be seen in the seventh epoch, and the maximum value in the final epoch, which is common in four of the five data sets used for training.

Comparing the losses of all training sessions(**Figure 4.1**), it can be seen that: WGISD has the lowest initial value (4.0227) and ACFR-Almonds has the highest initial value (4.1261); MangoYOLO is the one who presents the greatest and fastest decrease during all the iterations, also presenting the lowest loss value of all (0.9211); ACFR-Mangoes presents the highest final loss value (1.2851); finally, although ACFR-Mangoes presented less decrease during the first and last epoch (thus, there is a contrast between both sets of mango data), WGISD is the one that presents the greatest decrease throughout the remaining periods.

Regarding the classification loss (**Figure 4.2**), it is observed that: again, WGISD has the lowest initial value (1.1288) and Mango YOLO is the one with the greatest and fastest decrease during all iterations, also presenting the value of the lowest regressive loss of all (0.0878), but ACFR-Mangoes is the one who presents the highest initial loss value (1.1310) and is the one who presents the smallest loss decrease for at least the first 600 iterations, to then be surpassed (is it correct to say?) by ACFR-Almonds, whom it is maintained to the end as who presents the least loss and whose final value, the least of all, is 0.2049 (a difference of at least one-tenth, appreciable in the graph).

Regarding the regression loss (**Figure 4.3**), the following is observed: again, WGISD has the lowest initial value (2.8939) and ACFR-Almonds has the highest initial value (2.9983) and MangoYOLO is the one with the largest and fastest decrease during all iterations, also presenting the lowest regressive loss value of all (0.8333); However, here WGISD always remains the one with the lowest regressive loss during all iterations, obtaining a final value of 1.1084.

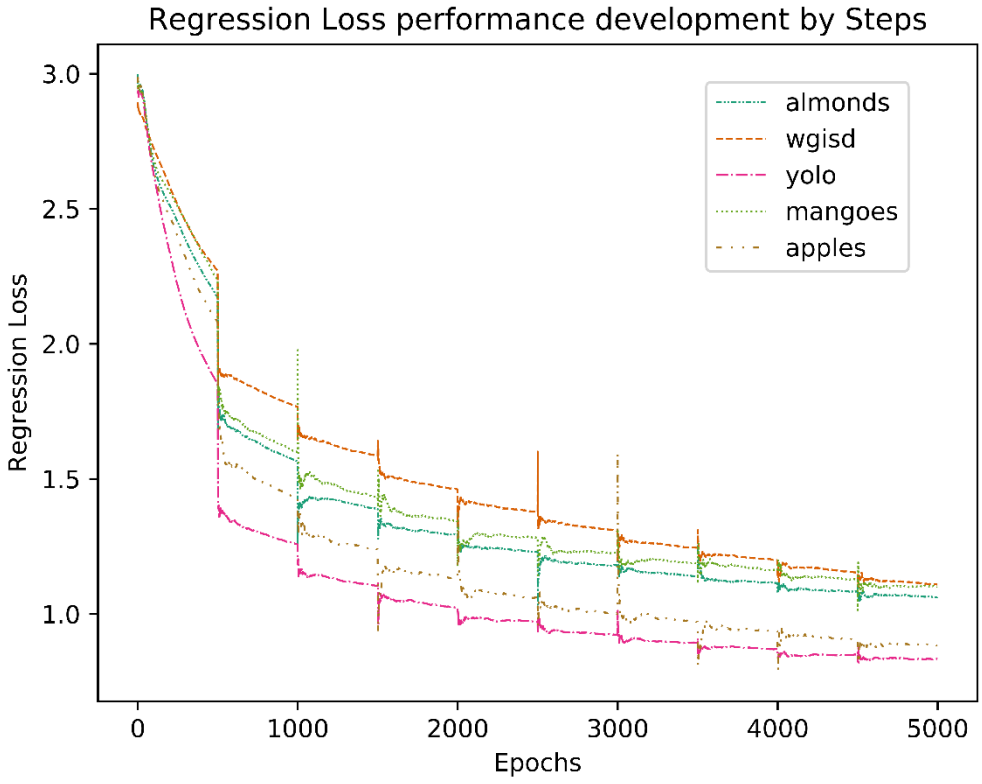


Figure 4.3. *Comparison between training regression loss across all datasets.*

The latter just mentioned above becomes significant when, when comparing the *mAP* from all the experiments(**Figure 4.4**), it is found that indeed, ACFR-Almonds is the one that presents the lowest AP gain throughout all periods, presenting the lowest mAP at the end (0.7978); however, it can be seen that, as in all types of losses, Mango YOLO is the one with the highest mAP gain throughout all periods, presenting the highest value at the end (0.9796); the other data sets always show prolonged growth at different levels, highlighting WGISD, which, from 400 iterations to 1200 iterations, more or less, presents lower mAPs than the ACFR-Almonds mAPs, demonstrating an increasing rate lower and slower than ACFR-Almonds, who gains mAP much faster in the first two epochs, but then loses momentum in subsequent epochs.

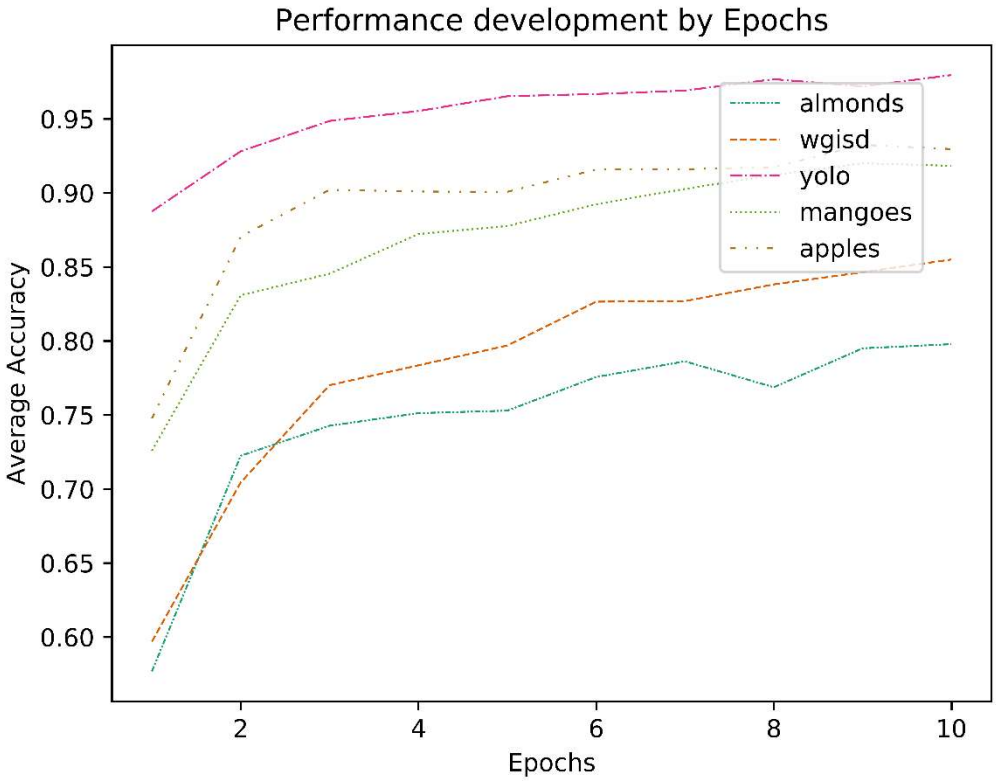


Figure 4.4. *Comparison between Average Accuracy performances across all datasets.*

4.2. YOLOv3

The next batch of experiments was carried out with YOLOv3, whose architecture is Darknet, implemented in C. Requires preprocessing, carried out with a python script provided by the same Darknet developer team.

Darknet's training record is even more verbose (extensive) than that offered by TensorFlow, offering information about the Normalizer (iou and class), the region worked, the average loss (iou and class), the class, the object, the count, total loss, class, and iou, among other things for each iteration (set 3800 in number) in each of the batches (set maximum 5200), making the record too long and dense to work (19,760,000 iterations); fortunately, Darknet only sends specific information to the command line for each processed batch, to wit, batch number, average loss, rate, seconds occupied in the batch, number of images processed and time remaining to finish the training, information that was captured for study purposes.

Based on the aforementioned, for each data set only the average loss graph is presented, and finally a comparative graph for all data sets.

The first thing that it can be noticed in the graph (**Figure 4.5**) is that most, if not the only, part of relevant information are in at most the first thousand batches, being that long before reaching a thousand batches, the average loss drops considerably until reaching its maximum (minimum value) and then remains the same for all remaining batches; because of this, a graph for the first five hundred batches is first empirically constructed and it is found that affirmatively, all relevant information is found in the first one hundred batches (380000).

The first reviewed experiment is ACFR-Mangoes(**Figure 4.6**), and it is seen the following behavior: during the first ten batches the loss remains almost stable (differences in values are noted in the records), and then steadily decreases until 40^{th} batch, where it suffers a slight increase and then decrease constantly until reaching the minimum value in 100^{th} batch (and the rest of subsequent batches).

In ACFR-Almonds, there is a similar behavior: the first ten batches show the same quasi-static behavior (the initial value is lower than that of ACFR-Mangoes), and it decreases continuously on a somewhat steep curve until it undergoes a breakpoint in the twentieth batch, then undergoes a steeper decrease, steeper curve, until it receives the breakpoint in the thirtieth batch, then it undergoes exponential growth up to 35^{th} batch or 36^{th} batch when it returns and decreases (thus building a kind of arc) and then decreases quickly in a not so steep curve until reaching its maximum decrease (minimum value) in from 100^{th} batch onwards.

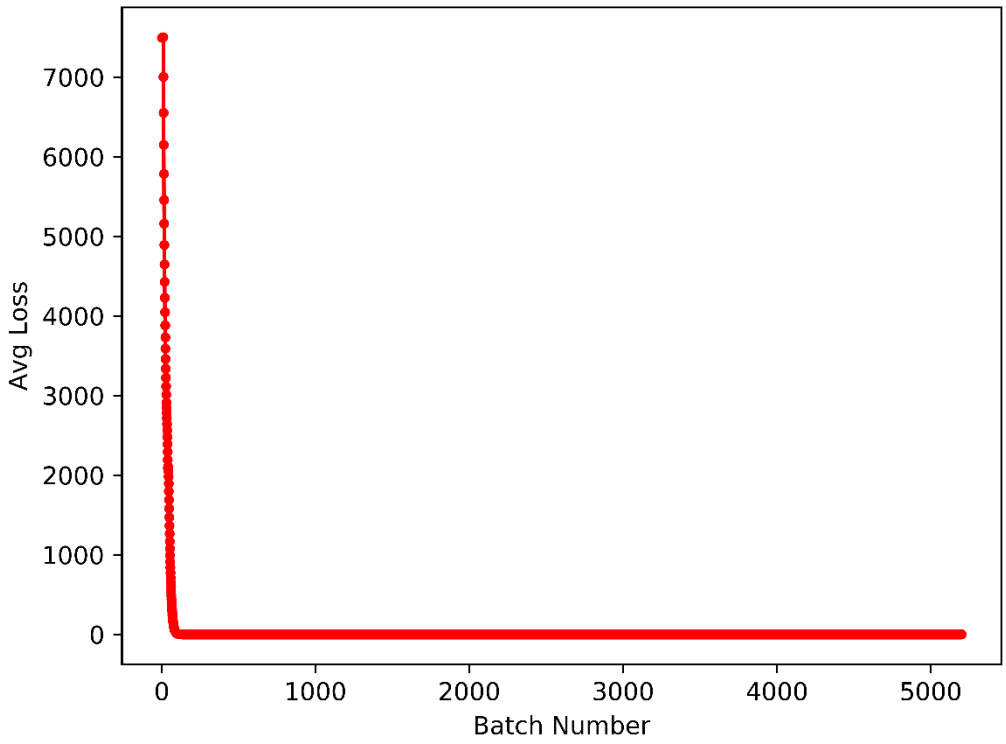


Figure 4.5. *YOLOv3 trained with ACRF Mangoes: despite the whole training takes 5000 batches, it seems the main Average Loss gets its low peak in before to the first thousand batches.*

At ACFR-Apple the same behavior is again gotten, although the average loss value is much higher than that of the previous ones: "static" up to 10th batch, a smaller almost linear decrease until 20th batch, a very pronounced exponential decrease until 30th batch, followed by a small growth in one or two lots, followed by a sharp decrease until it reaches its minimum value from 100th batch onwards.

The same for MangoYOLO, from the tenth batch to twentieth batch there is a pronounced spontaneous decrease, which is truncated until the thirtieth batch becoming rather linear; then, again, it is seen a growth/decrease forming an arc, from the thirtieth batch to 40th batch, and then again a decrease in a curve until reaching the minimum value.

Finally, WGISD presents a curve with fewer "accidents": "static", a pronounced decrease from the tenth batch to the thirtieth batch, then there is an increase and decrease in an arc up to the fortieth batch, and finally a decrease until reaching the minimum value.

Now, by comparing performance across all data sets (**Figure 4.6**) interesting data is obtained: two datasets belonging to ACFR occupy the extremes: Almonds has the

lowest initial value (6324.618652) and Apples have the highest initial value (7994.836426); all values tend to be the same or very close at the end of the training; ACFR-Apples presents the smallest decrease in average loss up to 42th batch, when ACFR-Mangoes becomes the one with the smallest decrease; on the other hand, initially, the greatest decrease is presented by MangoYOLO until the thirtieth batch, after which it is surpassed by ACFR-Mangoes until 42th batch, which is surpassed by WGISD, then surpassed by ACFR-Almond, in constant competition with MangoYOLO; the thirtieth batch is of utmost importance, as it is presented as a "turning point" for at least four data sets, where the prominent decrease stops, receives a growth and then a decrease (forming an arc) that will continue until it reaches a minimum value.

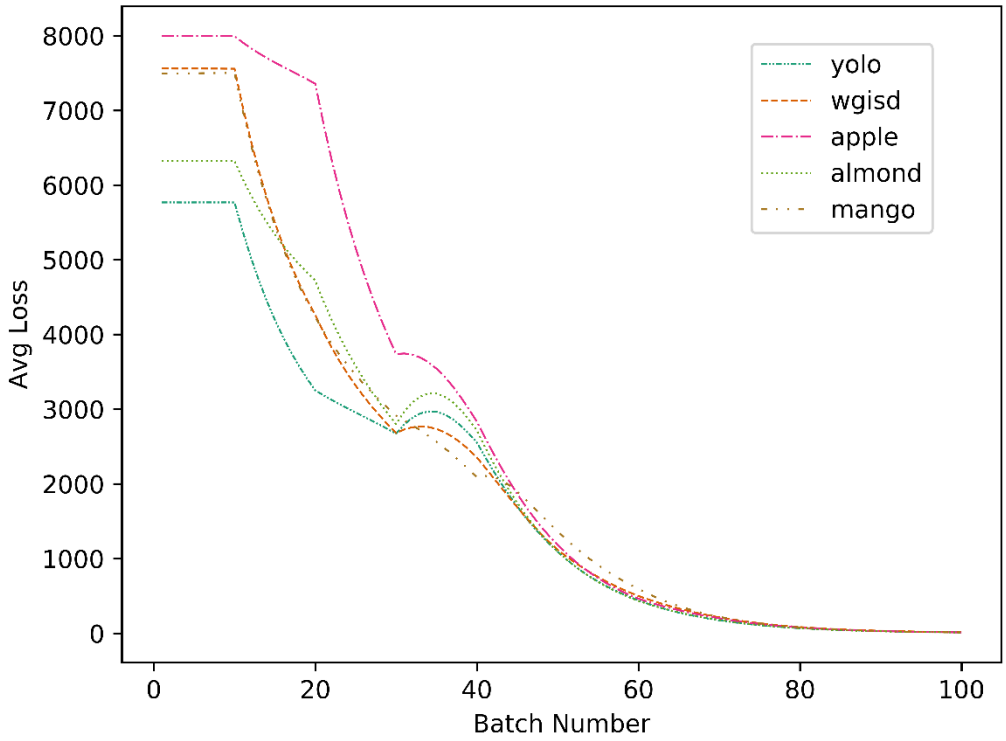


Figure 4.6. *Comparison of average loss performance across all datasets.*

It is very likely that in batches after batch 100 to batch 5200, the average loss will continue to decrease or continue to change, but as they are “insignificant” they are not visible on the graph.

4.3. YOLOv4

YOLOv4 retains the same formatting style for its records, so there is not much to mention in this section, except that this time it uses 4800 to 5400 iterations for each of the 6000 batches; it must be emphasized that keeping the parameters recommended by the framework developer was decided, and only one class (detected object) per training.

Following the previous example regarding YOLOv3, in the first graph (**Figure 4.7**) it can be seen that the most relevant performance is before the first thousand batches. So the same method was performed to these logs: first, it was graphed to the first 500 batches up, where it is realized that the relevant information is in the first 175 batches, with which the results up to this limit were plotted for all YOLOv4 experiments.

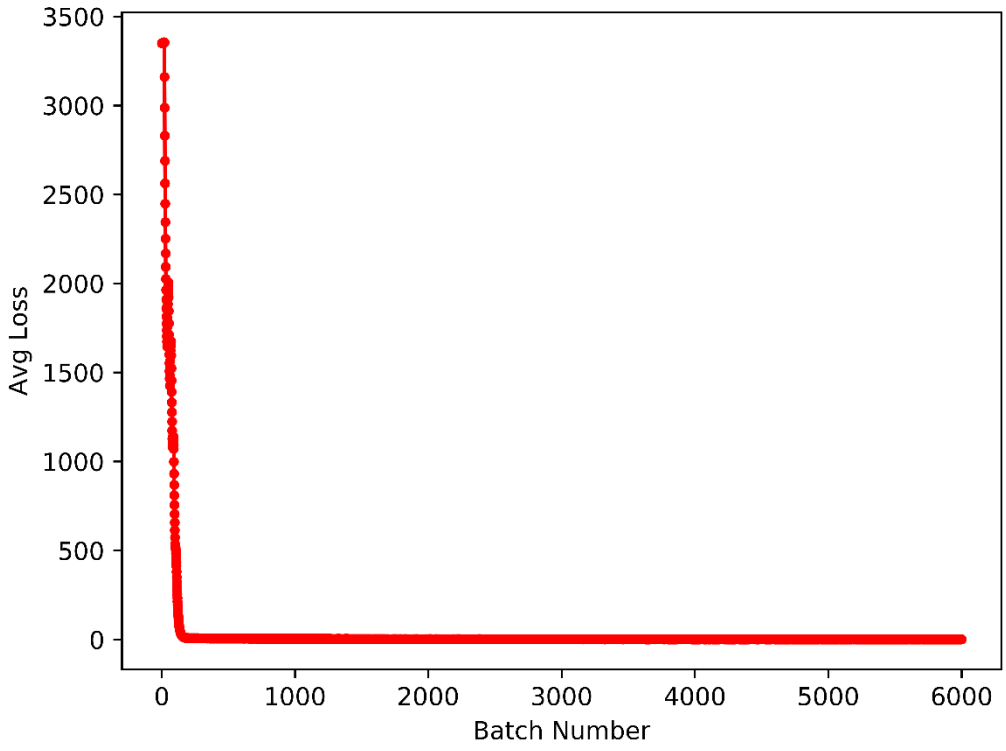


Figure 4.7. *YOLOv4 trained with WGISD: YOLOv3 same case, it takes 6000 batches to train it but the main average loss decreasing is in the first 500 batches.*

Starting by looking at the resulting graph (**Figure 4.8**) for WGISD: for the first twenty batches there is the same type of "static" behavior, then it decreases sharply until more or less fortieth batch, where it stops abruptly and starts to grow exponentially up to the fiftieth batch, where it decreases in the same way as it did before, this until sixtieth or

65th batch, where it grows again until the seventieth batch and then decreases and grows (although less pronounced), to decrease almost linearly and find another point in the 100^{th} batch, where it grows in a minuscule way and is followed by a decrease in a less pronounced way, finally, in 110^{th} batch approximately, a final change is made in the decrease, which will continue regularly until reaching its minimum value in the following lots, from the same way as in YOLOv3.

The next one is MangoYOLO, and it can be observed that the "static" behavior goes up to the tenth batch, followed by a pronounced decrease in the following ten batches; from the twentieth batch, the average loss increases a little to 31^{th} batch, where it breaks and declines sharply again during the next 10 batches and the previous behavior is repeated a third time except for batches from 51 to 63, approximately, where a curve is seen (growth followed by a decrease, both minuscule), it follows a four-cycle of pronounced growth and slight decrease, to end with a decreasing curve that can be divided into three segments according to the spacing between the values of the batches.

In ACFR-Mangoes, the "static" behavior occurs during the first ten batches; then it suffers a considerable decrease in the following ten batches (appreciable by the distance between the points), which continues less pronounced during the next eleven batches, then the already characteristic growth followed by a decrease (in the form of a sea wave) is seen, which is followed by an almost linear and segmented decrease to approximately seventieth batch; this is followed by a somewhat pronounced growth over the next eleven batches, which is then followed by an almost linear decrease that breaks into a more exponential one until it reaches its minimum value, as always.

ACFR-Apples exhibits a "static" behavior in the first ten batches, then declines sharply in the next ten batches; later three crests are seen, the last of which presents a growth segmented into two, evidencing a small decrease that breaks the previous growth, followed again by another growth, at the end of this crest a decrease occurs (the previous decrease presents visible distance between their values, while the following is not), which breaks slightly in 95^{th} batch, approximately, which continues with a slight concavity and then continuously decays to its minimum value.

Finally, in ACFR-Almonds, the "static" behavior occurs in the first ten batches; then it presents a continuous decrease, which changes in three different intervals, up to

the fiftieth batch; a series of two ridges continues, the second of which shows a rather pronounced decrease until 100th batch; it is then followed by a small arc (growth and decrease) of a maximum of ten batches, to end with a decreasing curve until it reaches its minimum value and thereafter the rest of the batches.

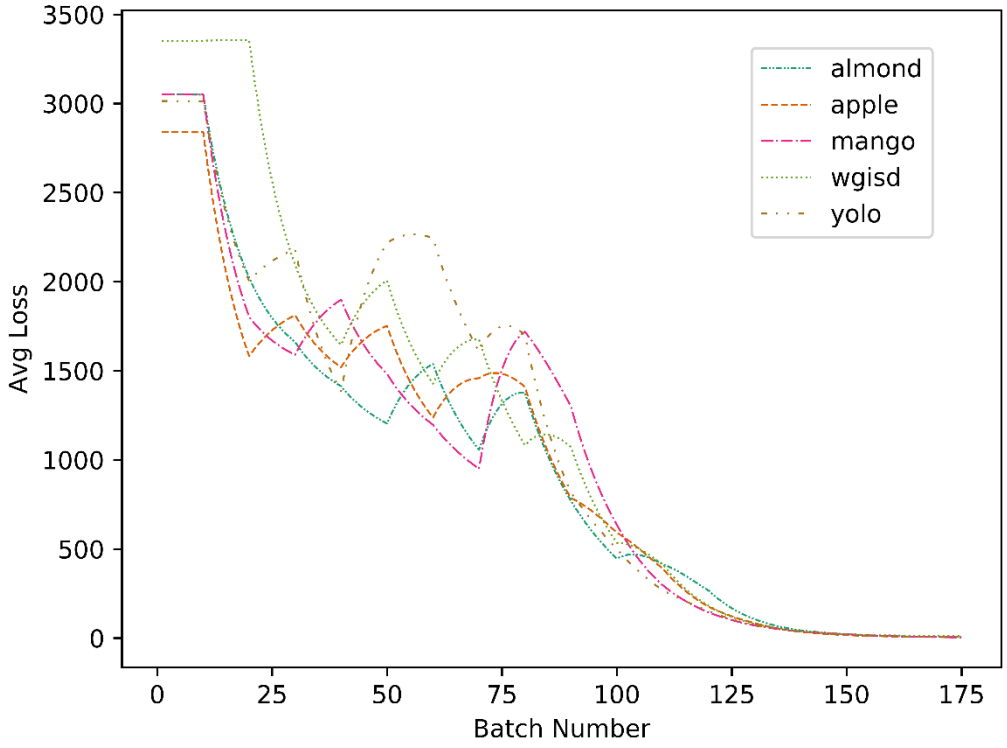


Figure 4.8. *Comparison of average loss performance across all datasets.*

When all the average-loss curves belonging to each data set are compared, **Figure 4.8,** interesting things are seen: WGISD is the only one that "takes" to start with the decrease of the average-loss, while for the others the "static" interval takes around ten batches, WGISD doubles the number of batches required before the average loss begins to decrease; during the entire training session, it is not possible to find a set that dominates one of the two extremes: at various points in the training, for example, each of the sets has been the one that has shown the greatest decrease, although it is found that there are inflection peaks after the which the average loss increases again, and at the end only decreasing curves are presented, all tending to the same limit (asymptote); likewise, although WGISD starts as the set that has the smallest decrease, after the first fourty batches each of the sets has intervals where its value is the highest of all, being so that none have to be above or below the others pre-eminently.

4.4. Summary

In this section, all collected data for the evaluation of each of the frameworks is presented: mAP and F1 for the evaluation of the performance in the detection and classification of objects, the training time and inference (in hours and seconds respectively), and the weight of the resulting trained models.

Each of these parameters responds to different needs of the problem addressed by the researcher: Which framework for which data set offers the highest precision? Which offers the best balance between precision and recall? Which model is the most conducive when the researcher lacks the time and resources to train? What when there is a limited waiting time when detecting and classifying objects? And finally, which model is more suitable to be used on mobile platforms? Knowing that these types of platforms lack the robust resources that in-house platforms possess.

4.4.1. Object detection and classification performance

Table 4.1 presents the results for the mAP and F1score metrics obtained by evaluating each of the models resulting from the training experiments; presents the value that the dataset (row) obtained with each of the three selected frameworks.

Table 4.1. *Media Average Precision and F1 Score across different datasets and frameworks.*

Dataset	**RetinaNet**		**YOLOv3**		**YOLOv4**	
	mAP	**F1**	**mAP**	**F1**	**mAp**	**F1**
MangoYOLO	0.9764	0.992	0.968022	0.92	**0.981608**	0.94
WGISD	**0.9239**	0.9979	0.898916	0.88	0.905797	0.90
ACFR – Almonds	**0.8530**	0.9662	0.659695	0.64	0.700880	0.71
ACFR – Apples	**0.9443**	0.95	0.904642	0.87	0.889885	0.85
ACFR – Mangoes	0.9283	0.9971	0.892233	0.85	**0.931554**	0.89

To begin, there is a contrast between the results obtained for MangoYOLO with the YOLOv3 framework in the paper (mAP: 0.967, F1: 0.951) (Koirala A. a., 2019) against the results of the same framework in the present study (mAP: 0.968, F1: 0.92), likewise, the results of the MangoYOLO framework (mAP: 0.986, F1: 0.967) (Koirala A. a., 2019) are still superior to the improved version of YOLOv3, YOLOv4 (mAP: 0.9816, F1: 0.94); RetinaNet is not mentioned here as its results are between YOLOv3 and YOLOv4.

Then, WGISD also presents different values regarding the YOLOv3 framework: mAP 0.566 and F1 0.718 in the paper (Santos T. a., 2020) compared to mAP 0.8989 and F1 0.88 of the present study; regarding the final results of the paper, although it is obtained in F1 less than a tenth (0.90 vs. 0.915), YOLOv4 manages to surpass Mask R-CNN in precision (mAP 0.9057 vs. 0.855); however, it is RetinaNet who manages to overcome both results obtaining the highest mAP (0.9239).

Regarding the ACFR data set, all the results obtained in the paper (Bargoti S. a., 2016) are surpassed by the frameworks used in the experiment (since Fast R-CNN is improved in Mask R-CNN and surpassed by the different frameworks developed later, as it is explained previously). It is found that Almonds has the worst performance, having for each of the three frameworks the lowest values in the table (mAp 0.8530 and F1 for RetinaNet, mAP 0.659695 and F1 0.64 for YOLOv3, and mAP 0.700880 and F1 0.71 for YOLOv4, respectively); a preponderance of RetinaNet over the other frameworks is seen. This also happens at Apple (mAP 0.9443), where it is also noticed something interesting: YOLOv3 outperforms YOLOv4 in performance (mAP 0.904642 and F1 0.87 vs. mAP 0.889885 and F1 0.85).

Finally, in Mangoes, it is YOLOv4 (mAP 0.931554 and F1 0.89) who takes the lead, followed by RetinaNet (mAP 0.9283 and F1 0.9971) and YOLOv3 (mAP 0.892233 and F1 0.85).

4.4.2. Training and inference time

Training time may be a decisive factor for many when choosing a framework: the specifications used to carry out the experiments in this study correspond to standard / average cost hardware, and although many of the largest research centers in the world indeed allow high-end or higher hardware, it is assumed that not all those interested in object detection in the field of agriculture could afford this type of hardware that is a quite significant investment from the beginning.

Table 4.2. Training *(hours) and inference (seconds) time across different datasets and frameworks.*

Data-set	RetinaNet		YOLOv3		YOLOv4	
	Training(h)	**Inference(s)**	**Training(h)**	**Inference(s)**	**Training(h)**	**Inference(s)**
MangoYOLO	**1.4187**	**0.0716**	8.5	2	13.85	3
WGISD	**5.3242**	**0.1405**	10.1246	3	18.6806	4
ACFR – Almonds	**1.2205**	**0.0931**	7.3723	2	13.15	3

ACFR – Apples	**1.7687**	**0.1022**	6.2833	3	13.26	3
ACFR – Mangoes	**1.2561**	**0.0695**	7.6446	7	13.8	7

Now, it can be seen then how RetinaNet is the one who takes the shortest possible time to train models, with an average of between an hour and hour and a half per class, being that all frameworks used the same parameters in this regard: a single class per experiment; YOLOv3 follows, with an average of 6 and a half hours to 7 and a half hours; finally, it is YOLOv4 who requires the longest training times per class: 13 hours on average.

Table 4.2 shows the training and inference time values for each of the frameworks with each of the datasets. WGISD stands out among all of them for breaking the averages and requiring, on average, 350% time for RetinaNet, 150% more time for YOLOv3, and 140% more time for YOLOv4 in terms of training time. Unfortunately, the papers consulted do not offer any information on the training time for their experiments, so there is no information to contrast with the given results.

Now, in terms of inference time, it is RetinaNet who has the best records, since none of them exceeds the second, while YOLOv3 exceeds YOLOv4 in inference time (tied it only two times). Here there is something to contrast: MangoYOLO registers 0.025 seconds (Koirala A. a., 2019) per image, being 300 the number of evaluation images, it would have approximately a total of 7.5 seconds in inference time for the evaluation set, while the in the present study it has two and three seconds for YOLOv3 and YOLOv4 respectively. It is worthy that only MangoYOLO (Koirala A. a., 2019) offers inference time records, hence, again, there is a lack of data to compare.

4.4.3. Model weight

Each of the models used for training have an initial weight; the architecture used by RetinaNet, in this case, ResNet50, has the lowest initial weight (100,533Mb) and therefore it could be deduced that the trained model will have the lowest weight as well, Darknet53, used by YOLOv3 follows (158,675Mb) and finally, the architecture used by YOLOv4, is slightly heavier than Darknet53 (166.054).

RetinaNet offers, as a result, a training model, which is not prepared for inference (regression and classification values), and whose weight is much higher than ideal;

however, since what is required is to infer (detect and classify objects), RetinaNet has a script that allows you to convert this training model to an inference model, significantly reducing its weight.

Table 4.3. *Trained model's weight (all in KB). Initial weights are ResNet-50-model.keras.h5, darknet53.conv.74 and yolov4.conv.137 respectively.*

Data-set	**RetinaNet**		**YOLOv3**	**YOLOv4**
	Training	**Inference**		
Base	100.533		158.675	166.054
MangoYOLO	427.170	**143.011**	240.469	250.016
WGISD	427.170	**143.011**	240.469	250.016
ACFR – Almonds	427.170	**143.011**	240.469	250.016
ACFR – Apples	427.170	**143.011**	240.469	250.016
ACFR – Mangoes	427.170	**143.011**	240.469	250.016

Table 4.3 presents the weights obtained for each of the trained models, including the RetinaNet inference model; RetinaNet is the one who offers the least weight per model; Although YOLOv3 and YOLOv4 offer 70% higher weights, they are not sizes that a modern mobile platform cannot handle.

It is also interesting to note that regardless of the number of images or image types that each data set provides, the deep-learning frameworks used in this study always provide a static model size (there are no changes from one value to another in each cell. of the respective column).

As for the GPU consumption in each experiment, having been done in Google Colab/Jupiter Notebooks, it was not possible to monitor the performance of the GPU throughout each experiment, so the present study cannot offer data on it.

5. DISCUSSION

A first detail that has been observed is the extrapolation of MangoYOLO and WGISD across all data sets: MangoYOLO is the one who excites a rapid increase in performance in the accuracy of the classifier while WGISD is the one that shows the slowest increase in most of the experiments.

As it is known beforehand, factors such as ambiguous conations (light intensity, saturation, etc.), generally affect the performance of classifiers when training; this is also related to the fact that each framework takes time with each of the datasets. It is seen that here also MangoYOLO is the one who needs the least training time in each framework while WGISD is the one who takes the longest time of all, sometimes even doubling the time required by MangoYOLO.

To understand this each of the datasets must be analyzed: MangoYOLO has been carefully built based on different experiments in which different types of hardware for image capture were tested, but also different atmospheric conditions, based on the light conditions of the environment (Koirala A. a., 2019), so it was concluded that daylight gives too much light intensity that could alter the perception of deep learning architecture in terms of extracting the characteristics of the object; on the contrary, the daytime lighting conditions provide better control of the light intensity, thus allowing images of objects to be obtained in better quality.

For its part, ACFR is composed of images (Bargoti S. a., 2017), small in size that facilitates rapid training, but which lacks good clarity, that is, many of the objects to be classified are blurred with the environment since the images are quite pale. and discolored, making it difficult for the neural network to extract information valuable enough to obtain good precision, hence its precision curve is normally always greater than that of MangoYOLO; Now, in the case of mangoes, although they share with MangoYOLO that they are taken at night, it can be seen that to capture the photos, not enough measures were taken to control the light intensity, making them look very dark, almost without light, and the objects to be classified are lost in the environment, precisely due to the lack of correct lighting; It is interesting to see how, although they follow a similar scheme, MangoYOLO and ACFR Mangoes follow opposite paths in their performance when training RetinaNet: MangoYOLO is who always has a better precision

gain while ACFR Mangoes has the worst performance, it can be seen here how the lack careful lighting significantly alters performance when training.

Finally, WGISD is made up of high resolution images of different grape clusters (Santos T. a., 2020), of different varieties (and therefore, different colors and shapes); most of these clusters correspond to fruits in the process of maturation or that in themselves are pale green in color, which can be easily confused with the color of the leaves that make up their bottom; therefore, this has two drawbacks: first, the images have three or four times the size of the images present in the other datasets, hence the training time required to train the neural networks is much higher than that of the other datasets, knowing that the size of the input images directly affects the number of operations required to extract the characteristics of the image, and therefore the number of calculations necessary to update the weights; On the other hand, since the number of images with objects of a color similar to other objects in the background of the images is so abundant, it is advisable to waste some time recalculating based on new examples trying to correctly differentiate, for example, a grape of a sheet.

This scheme is repeated throughout the three frameworks, not even the different operations introduced in YOLOv4 (Bochkovskiy, 2020) to mitigate the impact of overfitting on Darknet53 (Distance-IoU Loss, (Zheng, et al., 2020), etc.) alleviate the loss of precision suffered by the neural network when training with WGISD.

It can also be assumed that the distribution of the images in the training sets can affect to a lesser extent the performance of the classifier, since starting with very diverse examples would facilitate avoiding overfitting while having similar examples in the first training sessions (which are the most important as can be extracted from the comparative graphs) affects performance a little more when reaching the precision peak (for instance, most of the first images in WGISD correspond to pale-colored grapes that can easily be confused with the environment, while later you can already see varieties of different colors partially and totally).

This section also affects the architecture of the classifier: it has been seen that to train RetinaNet, by default, 5200 iterations are needed, while for YOLOv3 it is necessary for 19760000 iterations and 32400000 iterations for YOLOv4; Darknet is built on C and is optimally designed to mitigate the impact of the number of operations required per

iteration; The chosen implementation for RetinaNet, commercial, is built on Keras / TensorFlow, Python libraries for data science; All in all, RetinaNet offers better training performance, far exceeding the training speed of its architecture.

It is a surprise to see that while YOLOv4 (Bochkovskiy, 2020) boasts of being trained and used in conventional GPUs, YOLOv3 offers less training time, at the cost of less precision (in almost all cases), than YOLOv4, and this highlights especially when remembering that the experiments carried out in the present study were carried out on a Tesla GPU and whose results in terms of training time were superiorly poor concerning YOLOv3 (twice) and for RetinaNet (almost 10 times more). While it is true that most research teams in the field of data science and deep learning can easily count on more modern and much more powerful GPUs for commercial use, and to alleviate costs, it is not feasible Having high-powered GPUs but also large sizes on drones or any type of mobile device, a goal that Smart Agriculture researchers are aiming for today.

In the same way, both small and medium farmers will think twice before buying high-power GPUs since they are quite expensive to buy, so it is quite logical to bet on mid-range cards, which work both on desktops and computers. mobile devices, not to mention microcomputer-style mobile devices, which do not have the possibility of using large-caliber GPUs; Likewise, with the release of a new range of graphics cards, the cards of the previous ranges depreciate their value, making them more easily accessible and therefore more attractive to teams that are not fully dedicated to research in computer sciences.

6. CONCLUSIONS

To answer the question, which framework is better for which dataset? It can be said that if the ultimate objective is to prioritize precision above all, for handles (MangoYOLO and ACFR Mangoes) the most suitable framework is YOLOv4, however, if what you want is to have a well-trained architecture in a short time, to RetinaNet is recommended for all datasets, since it offers much shorter training and inference times, is faster, while maintaining very acceptable accuracy. Thus, RetinaNet is easy to implement, easy to train and its registration format is simple and useful (facilitating the analysis of training performance); RetinaNet also offers models with an acceptable "light" weight, which allows it to be used on mobile devices.

Likewise, it is important to emphasize the importance of registering the experimental environments, since it is imperative to repeat both experiments and results to corroborate them since the lack of information has made it impossible to contrast several of the results obtained in the present study, for which reason It is possible to conclude a lot about the size of the weights and the inference time of the models obtained in the papers used in this study.

What can be contrasted is the suitability of the method used by Koirala et. Al. (Koirala A. a., 2019), To build your dataset: in every way, it offers an ideal dataset to train in any type of framework, following well-defined parameters based on the results obtained in experimentation and which demonstrate, throughout the experiments, that the images obtained to improve the performance of the network and its precision by at least 0.01. Therefore, it is recommended to follow these guidelines in any type of research in the field of agriculture, since, as Korala et Al explain, improving the performance of a deep learning architecture does not only depend on the architecture itself but also the quality of the datasets.

Following the recommendations presented in the papers (Koirala A. a., 2019), this study includes the specifications of both the hardware and software used in this study, as well as the Jupiter Notebook files that were used to provision the experimental environment for each of the frameworks; Google Colab generates virtual instances for each notebook, so the environment for each experiment is initially the same in each experiment, therefore, by providing the notebooks, the reproducibility of the experiments is facilitated; all datasets used in this study are accessible to the public; the scripts used

to process the datasets are included in the source code of this study, allowing auditing of those present; the records extracted from the experiments, as well as the files to process them, are also included in the source code.

REFERENCES

Abdulla, W. (2017). Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. *GitHub repository*. Retrieved from https://github.com/matterport/Mask_RCNN

Alavi, N. a. (2012). Date grading using rule-based fuzzy inference system. *Journal of Agricultural Technology, 8*(4), pp. 1243 - 1254.

Bagri, N. a. (2015). A comparative study on feature extraction using texture and shape for content based image retrieval. *International Journal of Advanced Science and Technology, 80*(4), pp. 41 - 52.

Barghout, L. a. (2003, 07 11). *United States Patent No. 10/618,543.*

Bargoti, S. a. (2016). Deep Fruit Detection in Orchards.

Bargoti, S. a. (2017). Deep fruit detection in orchards. In *2017 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3626 - 3633).

Bargoti, S. a. (2017). Image Segmentation for Fruit Detection and Yield Estimation in Apple Orchards. *Journal of Field Robotics, 34*(6), pp. 1039 - 1060. doi:10.1002/rob.21699

Bhargava, A. a. (2018). Fruits and vegetables quality evaluation using computer vision: A review. *Journal of King Saud University - Computer and Information Sciences*. doi:https://doi.org/10.1016/j.jksuci.2018.06.002

Bochkovskiy, A. a.-Y.-Y. (2020). Detection, YOLOv4: Optimal Speed and Accuracy of Object. *ArXiv, abs/2004.10934*.

Brinkmann, R. (1999). *The Art and Science of Digital Compositing.* (M. Kaufmann, Ed.)

Corporation, N. (2020). *CUDA GPUs | NVIDIA Developer*. Retrieved from https://developer.nvidia.com/cuda-gpus

Crawford, K. a. (2018). Datasheets for Datasets. *ArXiv*.

Cui, H. a. (2016, 04). GeePS: scalable deep learning on distributed GPUs with a GPU-specialized parameter server. pp. 1 - 16.

Daheim, C., Poppe, K., & Schrijver, R. (2019). *Precision agriculture and the future of farming in Europe.* Scientific foresight study.

Dai, J. a. (2016). R-FCN: Object Detection via Region-based Fully Convolutional Networks. In *NIPS.*

Dalal, N. a. (2005). Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (Vol. 1, pp. 886 - 893).

Everingham, M. a. (2010, 06). The Pascal Visual Object Classes (VOC) challenge. *International Journal of Computer Vision, 88*, pp. 303 - 338. doi:10.1007/s11263-009-0275-4

Everingham, M. a. (2012). The PASCAL Visual Object Classes Challenge 2012 VOC2012 Results.

Ghabousian, A. a. (2012). Segmentation of apple color images utilizing fuzzy clustering algorithms. *Advances in Digital Multimedia, 1*(1), pp. 59 - 63.

Ghiasi, G., Lin, T.-Y., & Le, Q. V. (2018). DropBlock: A regularization method for convolutional networks. *Advances in Neural Information Processing Systems (NIPS)*, pp. 10727 - 10737.

Girshick, R. B. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 580 - 587).

Girshick, R. B. (2015). Fast R-CNN. In *IEEE International Conference on Computer Vision (ICCV)* (pp. 1440 - 1448).

He, K. a. (2016). Deep Residual Learning for Image Recognition. In IEEE, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770 - 778). Las Vegas.

He, K. a. (2017). Mask R-CNN. *CoRR, abs/1703.06870.*

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 37*(9), pp. 1904 - 1916.

Huang, G., Liu , Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2261 - 2269).

Huang, Z., Wang, J., Fu, X., Yu, T., Guo, Y., & Wang, R. (2020). DC-SPP-YOLO: Dense connection and spatial pyramid pooling based YOLO for object detection. *Information Sciences, 522*, pp. 241 - 258.

Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société vaudoise des sciences naturelles, 37*, pp. 547 – 579.

Jamil, N. a. (2014). A rule-based segmentation method for fruit images under natural illumination. In IEEE, *2014 International Conference on Computer, Control, Informatics and Its Applications (IC3INA)* (pp. 13 - 18).

Jhawar, J. (2016). Orange sorting by applying pattern recognition on colour image. *Procedia computer science, 78*(C), pp. 691 - 697.

Kamilaris, A. a.-B. (2017). A review on the practice of big data analysis in agriculture. *Computers and Electronics in Agriculture, 143*, pp. 23 - 37.

Kamilaris, A. a.-B. (2018). A review of the use of convolutional neural networks in agriculture. (C. U. Press, Ed.) *The Journal of Agricultural Science, 156*(3), pp. 312 - 322.

Kamilaris, A. a.-B. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture, 147*, 70 - 90.

Kestur, R. a. (2019). MangoNet: A deep semantic segmentation architecture for a method to detect and count mangoes in an open orchard. *Engineering Applications of Artificial Intelligence, 77*, pp. 59 - 69. doi:https://doi.org/10.1016/j.engappai.2018.09.011

Kitzes, J. a. (2008). Shrink and share: humanity's present and future Ecological Footprint. *Philosophical Transactions of the Royal Society B: Biological Sciences, 363*(1491), pp. 467-475.

Koirala, A. a. (2019). Deep learning – Method overview and review of use for fruit detection and yield estimation. *Computers and Electronics in Agriculture, 162*, pp. 219 - 234.

Koirala, A. a. (2019). Deep learning – Method overview and review of use for fruit detection and yield estimation. *Computers and Electronics in Agriculture, 162*, pp. 219 - 234. doi:https://doi.org/10.1016/j.compag.2019.04.017

Koirala, A. a. (2019, 12 01). Deep learning for real-time fruit detection and orchard fruit load estimation: benchmarking of 'MangoYOLO'. *Precision Agriculture*, pp. 1107 – 1135. doi:https://doi.org/10.1007/s11119-019-09642-0

Krizhevsky, A. a. (2012, 01). ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems, 25*.

Lee, B. R. (2015). An image segmentation approach for fruit defect detection using k-means clustering and graph-based algorithm. (Springer, Ed.) *Vietnam Journal of Computer Science, 2*(1), pp. 25 - 33.

Li, F.-F. a. (2017). *Detection and Segmentation.* (S. University, Ed.) Retrieved from CS231n: Convolutional Neural Networks for Visual Recognition: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lec-

Liakos, K. a. (2018, 08). Machine Learning in Agriculture: A Review. *Sensors, 18*, p. 2764.

Liang, Q. a. (2018). A real-time detection framework for on-tree mango based on SSD network. *International Conference on*, pp. 426 - 436. doi:https://doi.org/10.1007/978-3-319-97589-4_36

Lin, T. a. (2017). Feature Pyramid Networks for Object Detection. In IEEE, *Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 936 - 944).

Lin, T. a. (2020). Focal Loss for Dense Object Detection. In IEEE, *IEEE Transactions on Pattern Analysis and Machine Intelligence* (Vol. 42, pp. 318 - 327).

Lin, T.-Y. a. (2014). Microsoft COCO: Common Objects in Context. In *ECCV*.

Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). Path Aggregation Network for Instance Segmentation. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, W. a. (2016, 10). SDD: Single Shot MultiBox Detector. *9905*, pp. 21 - 37. doi:10.1007/978-3-319-46448-0_2

Loshchilov, I. a. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv*.

Martín Abadi, A. A. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Retrieved from https://www.tensorflow.org/

Mcbratney, A. a. (2005, 02). Future Directions of Precision Agriculture. *Precision Agriculture, 6*, pp. 7 - 23.

McBratney, A. a. (2005, 02). Future Directions of Precision Agriculture. *Precision Agriculture, 6*, pp. 7 - 23.

McNicoll, G. (2002). World Population Ageing 1950-2050. *Population and Development Review, 28*(4), p. 814+.

Mehra, T. a. (2016). Maturity and disease detection in tomato using computer vision. In IEEE, *2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)* (pp. 399 - 403).

Misra, D. (2019). Mish: A Self Regularized Non-Monotonic Neural Activation Function. *arXiv*.

Müller, R., Kornblith, S., & Hinton, G. (2019). When Does Label Smoothing Help? *CoRR, abs/1906.02629*.

Otsu, N. (1979). A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics, 9*(1), pp. 62 - 66.

Perry, M. (2015, 06). Science and Innovation Strategic Policy Plans for the 2020s (EU,AU,UK): Will They Prepare Us for the World in 2050? *Applied Economics and Finance, 2*.

Ramachandran, P., Zoph, B., & Le, Q. V. (2018). Searching for Activation Functions. *CoRR, abs/1710.05941*.

Redmon, J. (2013 - 2016). *Darknet: Open Source Neural Networks in C.* Retrieved from http://pjreddie.com/darknet/

Redmon, J. a. (2015, 06). You Only Look Once: Unified, Real-Time Object Detection.

Redmon, J. a. (2016, 12). YOLO9000: Better, Faster, Stronger.

Redmon, J. a. (2018). YOLOv3: An Incremental Improvement. *ArXiv, abs/1804.02767*.

Ren, S. a. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 39*(6), pp. 1137 - 1149.

Russakovsky, O. a.-F. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV), 115*(3), pp. 211 - 252. doi:10.1007/s11263-015-0816-y

Santos, L. a. (2020). Deep Learning Applications in Agriculture: A Short Review. In *Robot 2019: Fourth Iberian Robotics Conference* (pp. 139 - 151). Springer International Publishing.

Santos, T. a. (2020). Grape detection, segmentation, and tracking using deep neural networks and three-dimensional association. *Computers and Electronics in Agriculture, 170*, pp. 105 - 247. doi:https://doi.org/10.1016/j.compag.2020.105247

Sasaki, Y. (2007, October 26). The truth of the F-measure. *Teach Tutor Master*.

Saxena, L. P. (2014). A survey of image processing techniques for agriculture.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks, 61*, pp. 85 - 117.

Schmitz, A. a. (2015, 01). Mechanized agriculture: Machine adoption, farm size, and labor displacement. *18*, pp. 278-296.

Shafiee, M. J. (2017, 09). Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video. *Journal of Computational Vision and Imaging Systems, 3*. doi:10.15353/vsnl.v3i1.171

Shapiro, L. G. (2001). *Computer Vision.* New Jersey: Prentice-Hall.

Singh, A. a. (2015, 12). Machine Learning for High-Throughput Stress Phenotyping in Plants. *Trends in Plant Science, 21*.

Tan, M., Pang, R., & Le, Q. V. (2019). EfficientDet: Scalable and Efficient Object Detection. *ArXiv, abs/1911.09070*.

Team, T. (n.d.). *GPU support | TensorFlow*. Retrieved from https://www.tensorflow.org/install/gpu

Uijlings, J. a. (2013, 09). Selective Search for Object Recognition. *International Journal of Computer Vision, 104*, pp. 154 - 171. doi:10.1007/s11263-013-0620-5

University, S. (2020). *CS231n: Convolutional Neural Networks for Visual Recognition*. Retrieved from cs231n.stanford.edu

Walter, A. a. (2017, 06). Opinion: Smart farming is key to developing sustainable agriculture. *Proceedings of the National Academy of Sciences, 114*, pp. 6148 - 6150.

Wang, C. a. (2016, 05). DLAU: A scalable deep learning accelerator unit on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 36*.

Wang, C.-Y., Liao, H.-Y. M., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., & Yeh, I.-H. (2020). CSPNet: A new backbone that can enhance learning capability of cnn. *Proceedings of the IEEE Conference on Computer Vision*.

Wang, Z. a. (2019, 06). Mango Fruit Load Estimation Using a Video Based MangoYOLO—Kalman Filter—Hungarian Algorithm Method. *Sensors, 19*, p. 2742. doi:10.3390/s19122742

Woo, S., Park, J., Lee, J.-Y., & Kweon, I. S. (2018). CBAM: Convolutional block attention module. *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3 - 19.

Yao, Z. a. (2020). Cross-Iteration Batch Normalization. *arXiv*.

Yun, S. a. (2019, 10). CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6022 - 6031.

Yürekli, A. (2019). Lisansüstü Eğitim Enstitüsü Tez Yazım Kılavuzu. *Eskişehir Teknik Üniversitesi Lisansüstü Eğitim Enstitüsü*, 1-12.

Zhang , Z., He, T., Zhang, H., Zhang, Z., Xie, J., & Li, M. (2019, 02). Bag of Freebies for Training Object Detection Neural Networks. *arXiv*.

Zhang, B. a. (2014). Principles, developments and applications of computer vision for external quality inspection of fruits and vegetables: A review. *Food Research International, 62*, pp. 326 - 343. doi:https://doi.org/10.1016/j.foodres.2014.03.012

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., & Ren, D. (2020). Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. In *The AAAI Conference on Artificial Intelligence (AAAI).*

APPENDIX

Appendix A: acfrtovoc.py

```
import argparse
import cv2
import os
import pandas as pd
from xml.etree import ElementTree as ET

# Create arguments to pass to the script in the shell
parser = argparse.ArgumentParser()
parser.add_argument("-p","--path", help="Directory")  # Folder path
parser.add_argument("-cn", "--class_name", help="Object class") # Folder object class
parser.add_argument("-s", "--shape", default='square', help="Object class") # Folder object class
args = parser.parse_args()

print(args)

class_name = args.class_name
folder_name = os.path.basename(os.path.normpath(args.path))
folder_path = args.path

annotation_folder_path = folder_path + "/Annotations/"
image_folder_path = folder_path + "/JPEGImages/"

print(annotation_folder_path, image_folder_path)

square_fields = ['#item','x','y','dx','dy','label']
circle_fields = ['#item','c-x','c-y','radius','label']
depth = 3

# load the folder
for path, dirs, files in os.walk(annotation_folder_path):
        print("Analyzing {}".format(path))
        for each_file in files:
            # The path variable gets updated for each subfolder
            file_path = os.path.join(os.path.abspath(path), each_file)
            print(each_file)
            # load each file in the folder
            if args.shape == 'square':
                df = pd.read_csv(file_path, usecols=square_fields)
            elif args.shape == 'circle':
                df = pd.read_csv(file_path, usecols=circle_fields)

            # get the name of the image file
            fn = each_file.replace(".csv", ".png")

            # get the height and width of the image
            image_path = os.path.join(os.path.abspath(image_folder_path), fn)
            img = cv2.imread(image_path)
            height, width, channels = img.shape

            # creating the XML file
            annotation = ET.Element('annotation')
            ET.SubElement(annotation, 'folder').text = folder_name
            ET.SubElement(annotation, 'filename').text = fn
            ET.SubElement(annotation, 'path').text = image_path
            src = ET.SubElement(annotation, 'source')
            ET.SubElement(src,'database').text='Unknown'
            size = ET.SubElement(annotation, 'size')
            ET.SubElement(size, 'width').text = str(width)
            ET.SubElement(size, 'height').text = str(height)
            ET.SubElement(size, 'depth').text = str(depth)
            ET.SubElement(annotation, 'segmented').text = '0'
```

```
            for i in range(0, len(df)):
                ob = ET.SubElement(annotation, 'object')
                ET.SubElement(ob, 'name').text = class_name
                ET.SubElement(ob, 'pose').text = 'Unspecified'
                ET.SubElement(ob, 'truncated').text = '0'
                ET.SubElement(ob, 'difficult').text = '0'
                bbox = ET.SubElement(ob, 'bndbox')

                if args.shape == 'square':
                    xmin = df['x'].iloc[i]
                    ymin = df['y'].iloc[i]
                    xmax = xmin + df['dx'].iloc[i]
                    ymax = ymin + df['dy'].iloc[i]
                elif args.shape == 'circle':
                    xmin = df['c-x'].iloc[i] - df['radius'].iloc[i]
                    ymin = df['c-y'].iloc[i] - df['radius'].iloc[i]
                    xmax = df['c-x'].iloc[i] + df['radius'].iloc[i]
                    ymax = df['c-y'].iloc[i] + df['radius'].iloc[i]

                ET.SubElement(bbox, 'xmin').text = str(max(xmin,1.1))
                ET.SubElement(bbox, 'ymin').text = str(max(ymin,1.1))
                ET.SubElement(bbox, 'xmax').text = str(min(xmax,width))
                ET.SubElement(bbox, 'ymax').text = str(min(ymax,height))

            tree = ET.ElementTree(annotation)
            tree.write(os.path.join(os.path.abspath(path), each_file.replace(".csv", ".xml")),
encoding='utf8')
```

This script “updates” the ACFR dataset’s format.

Appendix B: wgisdtovoc.py

```
import cv2
import os
import pandas as pd
from xml.etree import ElementTree as ET

class_name = "uva"
folder_name = "wgisd"
folder_path ="datasets/wgisd/"
annotation_folder_path = folder_path + "Annotations/"
image_folder_path = folder_path + "JPEGImages/"

fields = ['CLASS', 'CX', 'CY', 'W', 'H']
depth = 3

# load the folder
for path, dirs, files in os.walk(annotation_folder_path):
        print("Analyzing {}".format(path))
        for each_file in files:
            # The path variable gets updated for each subfolder
            file_path = os.path.join(os.path.abspath(path), each_file)
            print(each_file)
            # load each file in the folder
            df = pd.read_csv(file_path, sep=' ', names = fields)
            # get the name of the file
            fn = each_file.replace(".txt", ".jpg")

            # get the height and width of the image
            image_path = os.path.join(os.path.abspath(image_folder_path), fn)
            img = cv2.imread(image_path)
            height, width, channels = img.shape

            # creating the XML file
            annotation = ET.Element('annotation')
            ET.SubElement(annotation, 'folder').text = folder_name
            ET.SubElement(annotation, 'filename').text = fn
            ET.SubElement(annotation, 'path').text = image_path
            src = ET.SubElement(annotation, 'source')
            ET.SubElement(src,'database').text='Unknown'
            size = ET.SubElement(annotation, 'size')
            ET.SubElement(size, 'width').text = str(width)
            ET.SubElement(size, 'height').text = str(height)
            ET.SubElement(size, 'depth').text = str(depth)
            ET.SubElement(annotation, 'segmented').text = '0'
```

```
            for i in range(0, len(df)):
                ob = ET.SubElement(annotation, 'object')
                ET.SubElement(ob, 'name').text = class_name
                ET.SubElement(ob, 'pose').text = 'Unspecified'
                ET.SubElement(ob, 'truncated').text = '0'
                ET.SubElement(ob, 'difficult').text = '0'
                bbox = ET.SubElement(ob, 'bndbox')


                bbox_width = df['W'].iloc[i].astype(float) * width
                bbox_height = df['H'].iloc[i].astype(float) * height
                center_x = df['CX'].iloc[i].astype(float) * width
                center_y = df['CY'].iloc[i].astype(float) * height

                xmin = center_x - (bbox_width / 2)
                ymin = center_y - (bbox_height / 2)
                xmax = center_x + (bbox_width / 2)
                ymax = center_y + (bbox_height / 2)

                ET.SubElement(bbox, 'xmin').text = str(max(xmin,1.1))
                ET.SubElement(bbox, 'ymin').text = str(max(ymin,1.1))
                ET.SubElement(bbox, 'xmax').text = str(min(xmax,width))
                ET.SubElement(bbox, 'ymax').text = str(min(ymax,height))

            tree = ET.ElementTree(annotation)
            tree.write(os.path.join(os.path.abspath(path), each_file.replace(".txt", ".xml")),
encoding='utf8')
```

Script for converting YOLO format to Pascal VOC 2012 (Everingham, 2012) format.

Appendix C: fix_paths.py

```
import os

folder_path = '../../datasets/wgisd_yolo'
image_folder_path = folder_path + "/data/"

sets = ['train', 'test']

for image_set in sets:
    image_ids = open(os.path.join(os.path.abspath(
        folder_path), '{}.txt'.format(image_set))).read().strip().split()
    list_file = open(os.path.join(os.path.abspath(
        folder_path), '{}.txt'.format(image_set)), 'w')
    for image_id in image_ids:
        list_file.write(os.path.join(os.path.abspath(
            image_folder_path), '{}.jpg\n'.format(image_id)))
    list_file.close()
```

Putting all image paths from training and text files in an absolute pathway.

Appendix D: make_val.py

```python
import argparse
import random

parser = argparse.ArgumentParser()
parser.add_argument("-p","--path", help="File path")  # File path
parser.add_argument("-n", "--number", type=int, default='50', help="Number of items") # Number of items
for new file
args = parser.parse_args()

def make_file(files_list, n_items):
    """ Compute the average precision, given the recall and precision curves.
    Code originally from https://github.com/rbgirshick/py-faster-rcnn.
    # Arguments
        file:    Train file or required source file.
        elements: Number of required files for the new file.
    # Returns
        val: A list with the n random files.
        train: A list without those n random files.
    """

    val = random.sample(files_list, n_items)
    train = files_list

    for item in val:
        train.remove(item)

    return train, val


if __name__ == "__main__":
    """
    Use:
        python make_val.py -p file.txt -n #
    """

    print("Analyzing {}".format(args.path))

    raw = open(args.path, "r").readlines()

    print("Creating training and validation values")
    train, val = make_file(raw, args.number)

    print("Printing training file.")
    with open("train.txt", "w") as output:
        for row in train:
            output.write(str(row))

    print("Printing validation file.")
    with open("val.txt", "w") as output:
        for row in val:
            output.write(str(row))
```

Creating a validation file from a training file.

Appendix E: voc_label.py

```python
import argparse
import os
import pickle
import xml.etree.ElementTree as ET

from os import listdir, getcwd
from os.path import join

parser = argparse.ArgumentParser()
parser.add_argument("-p", "--path", help="Directory")  # Folder path
parser.add_argument("-ie", '--image-extension',
                    help='Declares the dataset images\' extension.', default='.jpg')
args = parser.parse_args()


sets = ['train', 'val', 'test']
# No classes here: we are training one class per experiment
# classes = ["uva", "M", "almond", "apple", "mango"]

folder_path = args.path
annotation_folder_path = folder_path + "/Annotations/"
image_folder_path = folder_path + "/JPEGImages/"
labels_folder = os.path.join(os.path.abspath(folder_path), 'labels/')


def convert(size, box):
    dw = 1./size[0]
    dh = 1./size[1]
    x = (box[0] + box[1])/2.0
    y = (box[2] + box[3])/2.0
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x, y, w, h)


def convert_annotation(image_id):
    in_file = open(os.path.join(os.path.abspath(
        annotation_folder_path), '{}.xml'.format(image_id)))
    out_file = open(os.path.join(
        labels_folder, '{}.txt'.format(image_id)), 'w')
    tree = ET.parse(in_file)
    root = tree.getroot()
    size = root.find('size')
    w = int(size.find('width').text)
    h = int(size.find('height').text)

    for obj in root.iter('object'):
        difficult = obj.find('difficult').text
        cls = obj.find('name').text
        # if cls not in classes or int(difficult) == 1:
        #     continue
        # cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        b = (float(xmlbox.find('xmin').text), float(xmlbox.find('xmax').text), float(
            xmlbox.find('ymin').text), float(xmlbox.find('ymax').text))
        bb = convert((w, h), b)
        # str(cls_id)
        out_file.write(str(0) + " " + " ".join([str(a) for a in bb]) + '\n')
```

```
for image_set in sets:
    if not os.path.exists(labels_folder):
        os.makedirs(labels_folder)

    image_ids = open(os.path.join(os.path.abspath(
        folder_path), 'ImageSets/Main/{}.txt'.format(image_set))).read().strip().split()
    list_file = open(os.path.join(os.path.abspath(
        folder_path), '{}.txt'.format(image_set)), 'w')
    for image_id in image_ids:

        list_file.write(os.path.join(os.path.abspath(
            image_folder_path), '{}{}\n'.format(image_id, args.image_extension)))
        convert_annotation(image_id)
    list_file.close()
```

Appendix F: plotYOLOTrainLoss.py

```
import argparse
import sys
import matplotlib.pyplot as plt

parser = argparse.ArgumentParser()
parser.add_argument("-p", "--path", help="Log file path")  # log file path
parser.add_argument("-n", "--number", type=int,
                    help="Number of iterations to plot")
args = parser.parse_args()

path = args.path
number = args.number

lines = []
for line in open(path):
    if "avg" in line:
        lines.append(line)

iterations = []
avg_loss = []
batches = number if number else len(lines)

print('Retrieving data and plotting training loss graph...')
for i in range(batches):
    lineParts = lines[i].split(',')
    iterations.append(int(lineParts[0].split(':')[0]))
    avg_loss.append(float(lineParts[1].split()[0]))

fig = plt.figure()
for i in range(0, batches):
    plt.plot(iterations[i:i+2], avg_loss[i:i+2], 'r.-')

plt.xlabel('Batch Number')
plt.ylabel('Avg Loss')

figure_name = path.replace(".log", "_{}_plot.png".format(str(batches)))
fig.savefig(figure_name, dpi=1000)

print('Done! Plot saved as {}'.format(figure_name))
```

Appendix G: plotAllYOLOTraunLoss.py

```python
import argparse
import os
import sys
import matplotlib
import matplotlib.pyplot as plt

parser = argparse.ArgumentParser()
parser.add_argument("-p", "--path", help="Log folder path")  # log file path
parser.add_argument("-n", "--number", type=int,
                    help="Number of iterations to plot")
args = parser.parse_args()

path = args.path
number = args.number

file_names = os.listdir(path)
names = [x.split('_')[-1].replace('.log', '') for x in file_names]
traces = [path+'/{}'.format(x) for x in file_names]
print(names)

fig = plt.figure()
axes = fig.add_subplot(111)
cmap = matplotlib.cm.get_cmap('Dark2')
ls = [(0, (3, 1, 1, 1, 1, 1)), '--', '-.', ':', (0, (3, 5, 1, 5, 1, 5)), ]

for j, trace in enumerate(traces):
    print(trace)

    c = cmap(float(j)/len(traces))

    lines = []
    for line in open(trace):
        if "avg" in line:
            lines.append(line)

    iterations = []
    avg_loss = []
    batches = number if number else len(lines)

    print('Retrieving data and plotting training loss graph...')
    for i in range(batches):
        lineParts = lines[i].split(',')
        iterations.append(int(lineParts[0].split(':')[0]))
        avg_loss.append(float(lineParts[1].split()[0]))

    axes.plot(iterations, avg_loss,
              color=c, ls=ls[j], lw=.75, label=names[j])

lines = []
labels = []

for ax in fig.axes:
    axLine, axLabel = ax.get_legend_handles_labels()
    lines.extend(axLine)
    labels.extend(axLabel)

fig.legend(lines, labels,
           loc='upper right', bbox_to_anchor=(0.85, 0.85))

plt.xlabel('Batch Number')
plt.ylabel('Avg Loss')

figure_name = "all_{}_plot.png".format(str(batches))
fig.savefig(figure_name, dpi=1000)

print('Done! Plot saved as {}'.format(figure_name))
```

Appendix H: plotTFTrainLoss.py

```python
import argparse
import re
import sys
import matplotlib.pyplot as plt

# from collections import defaultdict

# python plotTFTrainLoss.py -p retinanet_mango_yolo.log -s 500


parser = argparse.ArgumentParser()
parser.add_argument("-p", "--path", help="Log file path")  # log file path
parser.add_argument("-s", "--steps", type=int,
                    help="Step's number used at training")  # step's number
args = parser.parse_args()

path = args.path
steps = args.steps

lines = []
avg_loss = []
f1_score = []
epoch_number = []
# epochs = defaultdict(dict)


print('Retrieving epochs data and plotting training loss graph...')
for line in open(path):
    if "Epoch" in line and not "Epoch 0" in line:
        epoch_number.append(int(line.split(" ")[1].split("/")[0]))
    if "mAP" in line:
        mAP = float(line.split(" ")[1])
        # epochs[epoch_number]['mAP'] = mAP
        avg_loss.append(mAP)
    if "mF1" in line:
        mF1 = float(line.split(" ")[1])
        # epochs[epoch_number]['mF1'] = mF1
        f1_score.append(mF1)
    if "/"+str(steps) in line:
        line = re.sub('\x08', '', line)
        lines.append(line.replace('\n', ''))

fig = plt.figure()
plt.plot(epoch_number, avg_loss, label="Average Loss")
plt.xlabel('Epochs')
plt.title('Performance development by Epochs')
plt.legend()

figure_name = path.replace(".log", "_{}_summary.png".format(str(steps)))
fig.savefig(figure_name, dpi=1000)

print('Done! Plot saved as ' + figure_name)

iterations = range(0, int(steps)*len(epoch_number))


loss = []
regression_loss = []
classification_loss = []

print('Retrieving steps data and plotting training loss graph...')
for i in range(len(lines)):
    lineParts = lines[i].split('-')
    loss.append(float(lineParts[2].split(': ')[1]))
    regression_loss.append(float(lineParts[3].split(': ')[1]))
    classification_loss.append(float(lineParts[4].split(': ')[1]))
```

```
fig = plt.figure()

plt.plot(iterations, loss, label="Loss")
plt.plot(iterations, regression_loss, label="Regression Loss")
plt.plot(iterations, classification_loss, label="Classification Loss")
plt.xlabel('Epochs')
plt.title('Performance development by Steps')
plt.legend()

figure_name = path.replace(".log", "_{}_steps.png".format(str(steps)))
fig.savefig(figure_name, dpi=1000)

print('Done! Plot saved as ' + figure_name)
```

CURRICULUM VITAE

PERSONAL INFORMATION

José Luis Sandoval Alaguna

Çamlıca Mahallesi Gülperi Sokak No.9, Tepebaşı, 26130 Eskişehir (Turkey)

(+90)5373749236

alagunasalahaddin@live.com

co.linkedin.com/in/salahaddinyusuf

github.com/SalahAdDin

es.stackoverflow.com/users/801/salahaddin

EDUCATION AND TRAINING

01/2004 – 12/2009 Technician-Technologist Bachelor in Systems and Computing

Escuela Tecnológica Instituto Técnico Central, Bogotá D.C. (Colombia)

Technical-technological bachelor with systems and computing specialization. Training in programming in various programming languages: C, C ++, Java, JavaScript, and VisualBasic.

01/2010 – 12/2015 Computer and Systems Engineer

Universidad Nacional de Colombia, Bogotá D.C. (Colombia)

09/2017 – 08/2020 Master in Computer Engineering

Eskişehir Teknik Üniversitesi, Eskişehir (Turkey)

Thesis: Comparative Analysis of Deep Learning Frameworks for Automated Fruit Detection in Smart Agriculture.