



Escola d'Enginyeria de Telecomunicació i
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Extending Object Classification Convolutional Neural Networks to custom logo detection

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: David Peña Moliner

ADVISOR: Francesc Tarrés Ruiz

DATE: September 7, 2020

Title: Extending Object Classification Convolutional Neural Networks to custom logo detection

Author: David Peña Moliner

Advisor: Francesc Tarrés Ruiz

Date: September 7, 2020

Abstract

The aim of this project is to automate the calculation of the total time that the logos of the sponsoring brands of moto GP appear on the screen during the races. This document explains all the steps that have been followed to train an automatic object detection model for a specific database using RetinaNet.

At the beginning, a brief explanation of the main concepts of deep learning is given and it is explained how convolutional neural networks and their kinds of layers, convolution and pooling, operate. Afterwards, it is presented a state of the art of the main classification and object detection systems, where RetinaNet has been chosen because, currently, it is one of the systems that provides better results. It should be noted that the main difference between classification and detection is that a detection system obtains the position of the object (rectangular region called bounding box) and indicates its typology and the classification system only indicates its typology.

A database of images from 6 moto GP videos had to be created and labeled using Labeling software. Labeling an image means drawing the bounding box and defining which brand the logo belongs to. The selected brands have been: Alpinestar, DHL, Repsol, GoPro, Michelin, RedBull, Monster, Tissot, Motul and BMW and a total of 16 classes have been created since one can have several forms of logos. Due to the fact that the database is not large enough to train a model from scratch, the weights of a pre-trained network have been used, this technique is known as transfer learning. In addition, to avoid overfitting, the layers of one part of the architecture called the backbone have been frozen, which, in this case, has been used with Resnet50. Later, another model has been trained applying data augmentation, to improve the results obtained from the first model trained. Data augmentation is a technique that generates new examples by performing transformations on the images in the database. With this, it has obtained an accuracy of 83.3% and a mean Average Precision (mAP) of 65.33%.

Finally, an application example called Brand Logo Monitoring in Moto GP has been developed, which, using the model trained with data augmentation,

counts automatically the time of appearance of each brand in the MotoGP Grand Prix and give a final result with the total time.

CONTENTS

INTRODUCTION.....	1
CHAPTER 1. AI, MACHINE LEARNIG AND DEEP LEARNING.....	2
1.1. Machine Learning	3
1.2. Deep Learning.....	3
1.3. How does a neural network learn?	4
1.4. CNN (Convolutional Neural Network).....	5
1.4.1. Convolution	7
1.4.2. Pooling	7
CHAPTER 2. OBJECT DETECTION AND CLASSIFICATION	9
2.1. Object classification systems	9
2.1.1. Alexnet	9
2.1.2. ZFNet.....	10
2.1.3. VGG Net	10
2.1.4. GoogleNet and Inception modules	11
2.1.5. Microsoft Resnet.....	12
2.1.6. Xception	12
2.2. Obect detection systems	13
2.2.1. Two-stage architecures	13
2.2.1.1. R-CNN.....	14
2.2.1.2. FAST R-CNN.....	14
2.2.1.3. FASTer R-CNN	15
2.2.2. One-stage architectures	15
2.2.2.1. YOLO	15
2.2.2.2. SSD.....	16
2.3. Multiscale architectures.....	17
2.3.1. FPN: Feature Pyramid Network.....	17
2.3.2. RetinaNet.....	18
2.4. Choice of RetinaNet	19

CHAPTER 3. IMPLEMENTATION OF RETINANET 21

3.1. Development environment 21

3.2. Database creation..... 22

3.2.1. FFMPEG module for frame extraction 22

3.2.2. Frames Labeling 23

3.2.3. Convert xml files to RetinaNet csv format 26

3.3. Quality Metrics 27

3.3.1. Intersection Over Union (IOU) 28

3.3.2. mean Average Precision (mAP) 28

3.3.3. Accuracy 29

3.3.4. Confusion matrix 29

3.4. Training with RetinaNet 29

3.4.1. Training applying data augmentation 31

3.4.2. Comparison of results..... 32

3.5. Predictions with RetinaNet..... 35

3.5.1. Results 35

3.5.2. Inferences 36

CHAPTER 4. APPLICATION EXAMPLE: BRAND LOGO MONITORING IN MOTO GP 38

4.1. App definition..... 38

4.2. Algorithm..... 38

4.2.1 Total time results 40

4.5. Enviromental Impact 40

CONCLUSIONS..... 41

REFERENCES..... 42

INTRODUCTION

In the world of motorcycle or car racing, sponsors want their brands to be seen as long as possible on the screen. The time of appearance has a price, so these companies are interested in getting a record of this time at the end of each race. Currently, this process is done manually, they have hired people who watch the race from start to finish and calculate the time when their logos appear on the screen.

The main objective of this project is to create an application that automates the process of counting the time when sponsor's brands appear on the screen in Moto GP. To automate this process, an artificial intelligence system has been implemented. Specifically, a deep learning object detector called Retinanet has been used as a base model for the analysis and a new model has been trained from a trained network to detect the logos. For this purpose, different techniques have been used. The memory is structured with the intention of transmitting the main concepts used in Deep learning in the first chapters, in order to increase the reader's knowledge in this area and at the end have a clear and global vision. The final chapters are more focused on the development of the system.

Going into more detail, this memory is divided in five chapters. The first, explains the concepts of AI, machine learning and deep learning, how a neural network learns and how a convolutional neural network works, where the concept of neuron, convolution operation and pooling operation are introduced. The second chapter introduces the concepts of classification and object detection, it presents the most important technologies and it explains the reasons for choosing the Retinanet architecture. The third chapter explains all the steps for implementing a model for logo detection using techniques such as transfer learning (it uses the weights of a pretrained model due to the fact that the database is not large enough to train a model from scratch), fine tuning (freeze convolutional layer in order to avoid overfitting) and data augmentation (creates new samples applying transformations over the data base's images, to improve the model's performance). And the results obtained in the training and prediction stage are presented. The fourth chapter explains the final application, called Brand Logo Monitoring in Moto GP, and how it works through the flowchart of its algorithm. Finally, the conclusions obtained after the realization of this project are argued.

CHAPTER 1. AI, MACHINE LEARNING AND DEEP LEARNING

Nowadays, the word Artificial Intelligence sounds everywhere and it's used increasingly. Some of the most common applications are: autonomous cars, voice recognition, data analysis, assistants such as the Google assistant, Amazon (Alexa) or Apple (Siri), etc. AI was born in the year 1950 when some pioneers of computer science asked themselves if machines could think. The main reason why there hasn't been this boom until now is because the technology of the time was not within reach of what AI needed. Discussions are also held about machine learning and deep learning, but most people who are not in this area probably don't know the difference between them, so it is briefly explained in the following points.

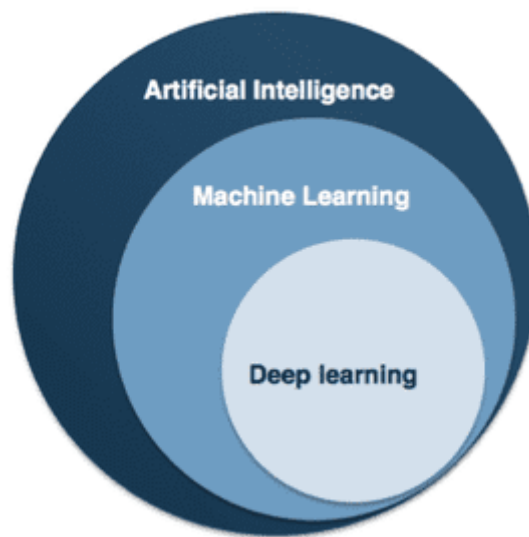


Fig. 1.1 Artificial Intelligence, machine learning and deep learning

As shown in **Fig. 1.1**, deep learning is a specific part of Machine Learning and machine learning is part of AI. So, when we talk about AI, we are referring in a global way to all the mechanisms that are used, like machine learning, deep learning or others systems like logical inference, etc.

1.1. Machine Learning

The concept of machine learning dramatically changes the way we think about classical programming when making a program. In the classic, someone provides the data and defines the rules of the program to obtain an answer. In machine learning, on the other hand, someone give the data with the answers and the machine creates its own rules. These rules can then be applied to new data to generate new answers. All that is needed to perform machine learning is:

- Input data points: In the case of object classification, it would be images.
- Example of the expected output: If they were logo images, the expected solution would be the name of the brand that corresponds to each image.
- A measure to know if the algorithm is working well: A metric is applied to compare the solution predicted with the real solution. The measurement is used as a feedback signal to adjust the how the algorithm works. This adjustment is called learning.

1.2. Deep Learning

From machine learning was born Deep learning, which is a new vision of learning from data that emphasizes the learning of successive layers with a hierarchy: this means that the first layers can detect lines, curves and specialize until they reach deeper layers that recognize complex shapes such as a face or the silhouette of an animal. The depth of a model is defined by the number of layers it uses. Most of the models used in deep learning are neural networks.

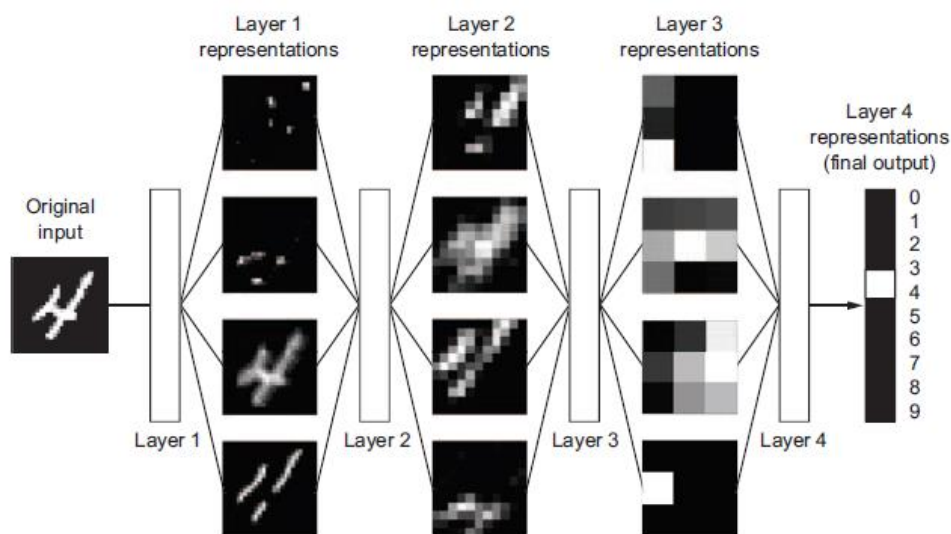


Fig. 1.2. Deep representations learned by a digit-classification model [1]

In the **Fig. 1.2** is showed a representation of the architecture that could have a neural network for the classification of handwritten digits, this example would be the Hello World of classic computing. And it shows the different representations of the digit 4 which is obtained by layers until the final solution is achieved.

1.3. How does a neural network learn?

It has been explained how these networks are formed and how they represent the information in each layer consecutively until they reach the last layer which has the best solution. But where is the learning here? How does it really learn? This section will answer these questions. The elements that compose a neural network to perform its operation and learning are the following:

- Layers: which perform the data transformations indicated by their weights.
- Input data: for this project, the images with the corresponding labels in the format adapted to the training model.
- Loss function: defines the value that is used as feedback for learning.
- Optimizer: defines the way in which it is to be learned.

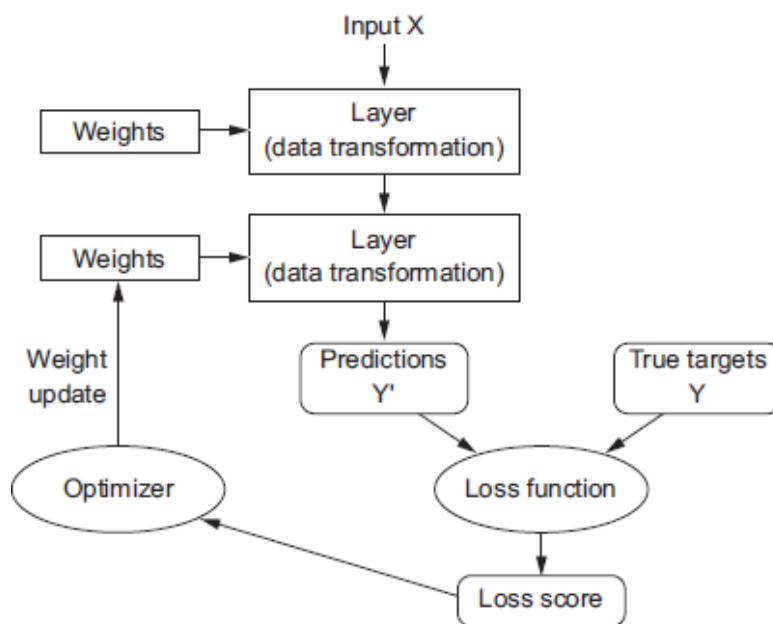


Fig. 1.3 Flow chart of a neural network [1]

The **Fig. 1.3** shows the complete flow chart of a neural network. As you can see, the input data pass through the layers of the Deep learning model used (in this project the model applied has been Resnet50). Each layer will implement the transformation of the data parameterized by its weights. In the output of the network, predictions are made and compared with the real data, this is the objective of the Loss function. From this comparison a result called loss score is obtained which is passed through the optimizer which updates the value of the weights of all the layers with the objective of decreasing the loss score. Therefore, the process of learning is the process of adapting the weight of the network. It can be concluded that the knowledge is on the weights.

There are many types of neural networks: Feedforward Neural Network, Radial Basis Function Neural Network, Multilayer Perceptron, Modular Neural Network, Convolutional Neural Network, etc. Each uses different principles to determine its own rules. For this project it has used the CNN that is explained in the next point.

1.4. CNN (Convolutional Neural Network)

When the images are analyzed, one of the most successful solutions is the CNN (Convolutional neural network) and the main difference from densely connected neural networks is that the dense layer learns global patterns in its global input space, while the convolutional layers learn local patterns in small two-dimensional windows.

Its architecture is divided into two parts, the convolutional and the densely connected. The part of the convolutional layers works with tensors, called features map. A color image would be a 3-dimensional tensor since it has three channels: red, green and blue (RGB) and it would be represented as follows: (height, width, channels). In order to explain how this type of network works, a simple example will be presented using a 28x28 pixel grayscale image as the input data. This is equal to a tensor with the shape (28,28,1), with this information it is possible to obtain the number of neurons. An artificial neuron (**Fig. 1.4**) is like a device that, from a set of connections, receives the inputs from which it perceives the information and it generates a single output.

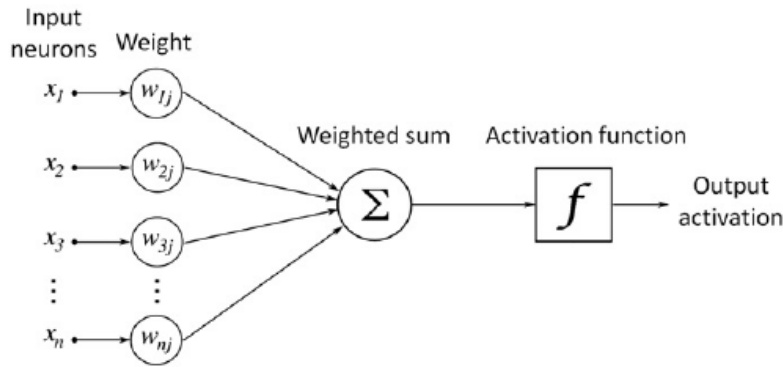


Fig. 1.4 Artificial neuron scheme

Each input set is characterized by a weight, the output will be the value obtained from the sum of the inputs considered individually (once weighted by their respective weights) and taking the result as input for the neuron activation function. The activation function is usually a deterministic function and, almost always, it is continuous and increasing. In some cases, it is a threshold value (until it exceeds the threshold the neuron is not activated).

In this case, the number of neurons obtained from the tensor would be $28 \times 28 \times 1 = 784$. If it were a color image, the tensor would have the following form $(28, 28, 3)$ and the number of neurons would be $28 \times 28 \times 3 = 2352$. Note that an image is an array where the colors of the pixels are represented with values from 0 to 255, and before feeding this network these values are normalized in a manner that the range of colors is between 0 and 1.

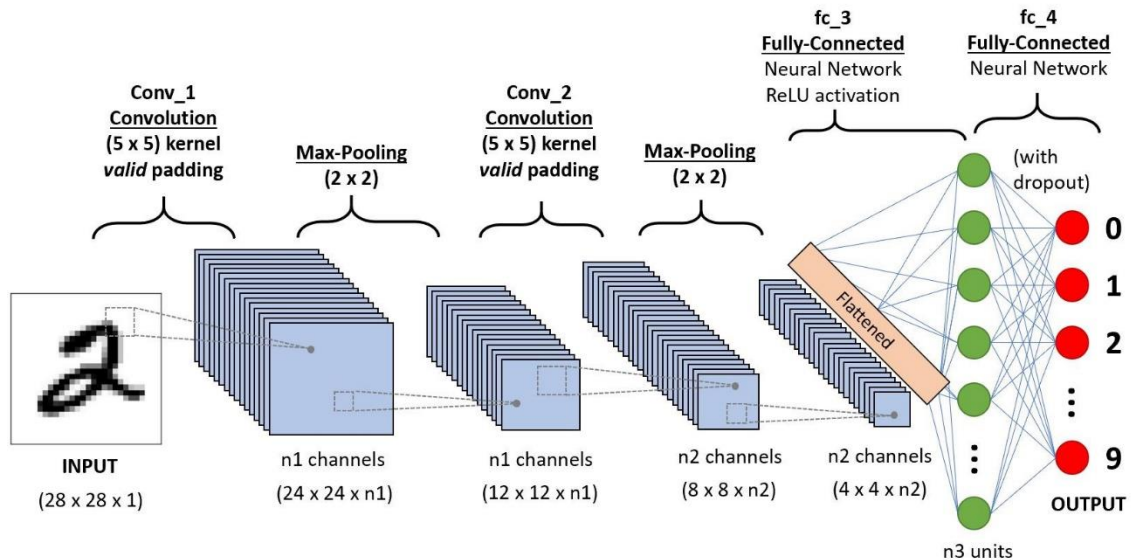


Fig. 1.5 Example of a 4 layers convolutional neural network

The previous image (**Fig. 1.5**) is an example of the architecture that has a CNN where you can see that the convolutional part is formed by 4 layers, normally they

have many more. There are two types of layers, depending on the type of the operation they perform, convolution and pooling. The last layer of the convolutional part is connected to the densely connected part and it has to provide the set of inputs with their respective weights to the neurons.

1.4.1. Convolution

Following the example in the previous image, it can be seen that the first convolutional layer uses a 5×5 window or kernel, that is, each neuron in the hidden layer will connect to a small 5×5 region of the 28×28 input layer.

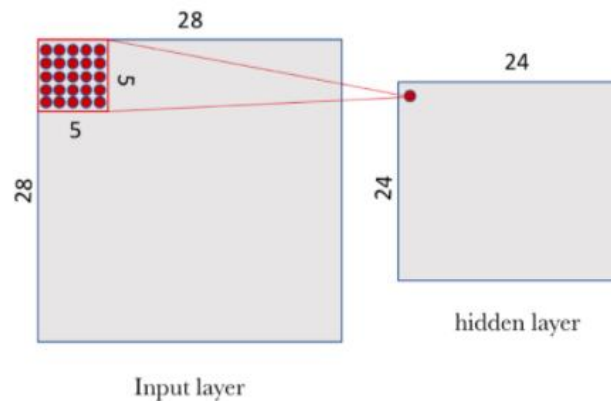


Fig. 1.6 Example of convolution 5×5 [2]

This window starts at the top corner of the input layer and moves from left to right and from top to bottom until it runs through the entire space. For each position in the window there is a neuron in the hidden layer that processes this information. The default window makes a forward movement of 1 neuron, but it can be defined with the parameter called stride (large stride values decrease the size of the information that will be passed to the next layer). In this case (**Fig. 1.6**), it will get a layer of 24×24 neurons since the stride is equal to 1 and the window will only be able to move 23 neurons down and to the right.

1.4.2. Pooling

Pooling layers are usually applied after the convolutional layers and they are used to simplify the information obtained from the convolutions by generating a condensed version of them. There are different versions to condense the information, like max-pooling which consist on selecting the maximum value of $N \times N$ window. Other alternative could be average pooling.

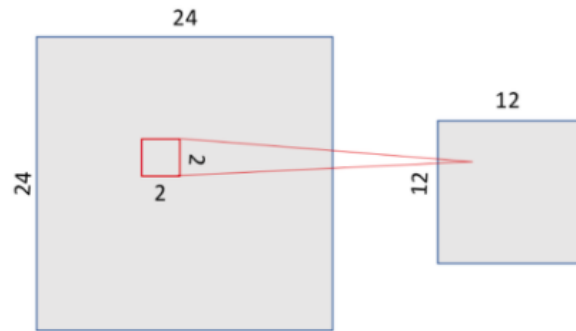


Fig. 1.7 Example of max-pooling 2x2 [2]

In the example of **Fig. 1.7** is used a max-pooling layer with 2x2 window, which will keep the maximum value of this region of 4 neurons in the 24x24 space. As a result, the space will be reduced to half (12x12 neurons). Note that with the pooling transformation the spatial relationship is maintained

CHAPTER 2. OBJECT DETECTION AND CLASSIFICATION

This chapter presents a summary of the state of the art in object detection and classification systems. The objective is to present an overview of the different existing object detection and classification networks, reasoning the choice of the Retinanet network for this project.

2.1. Object classification systems

An automatic classification system indicates the object that appears in the image. For example, in this project we have a system of 10 objects (the logos of the selected brands), the classifier, which has been trained to detect the logos, calculates the probability that each of these objects has of being in the image and normally selects the one with the highest probability (top-1), although in some cases the five best results are taken (top-5). The most popular systems for classifying objects will be explained below. It will compare their architecture, their performance and the number of parameters they have.

2.1.1. AlexNet

Developed in 2012, AlexNet [3] was the first analysis and classification system based on Deep learning. This neural network was trained with the imagenet database (ImageNet is a large database of over 14 million images. It was designed by academics intended for computer vision research). It is a neural network of 11 layers of depth and it achieved to reduce the top-5 classification error from 25% to 16%.

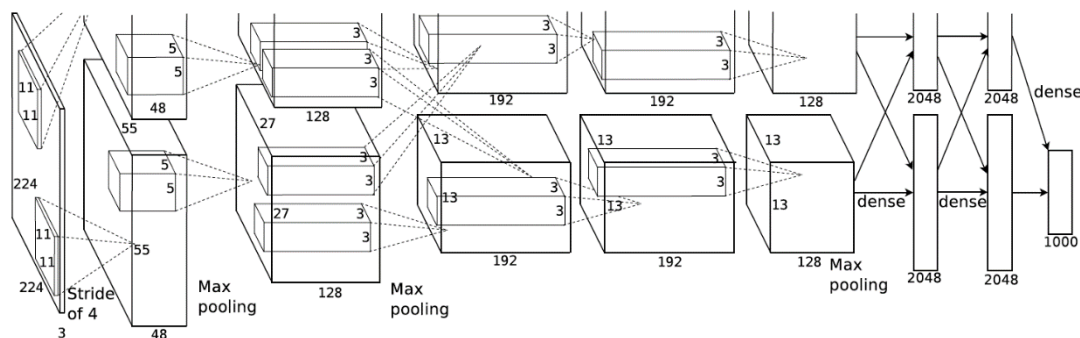


Fig. 2.1 AlexNet architecture

As shown in the **Fig. 2.1**, the first layer uses 11x11 pixel convolution elements, calculating a total of 48 outputs for each input window. This means that the number of parameters needed to learn are $11 \times 11 \times 3 \times 48 = 17.424$. And the memory to store the results of this layer would be $2 \times 55 \times 55 \times 48 = 17.424$. Therefore, the structure is complex both in terms of memory requirements and the number of parameters to be learned.

2.1.2. ZFNet

The ZFNet [4] convolution network was the winner of the Imagenet competition in 2013. It achieved a top-5 rating error rate of 11.2%. The architecture is very similar to AlexNet but brings different ideas in the interpretation of the functionalities in each layer and an improvement of the result.

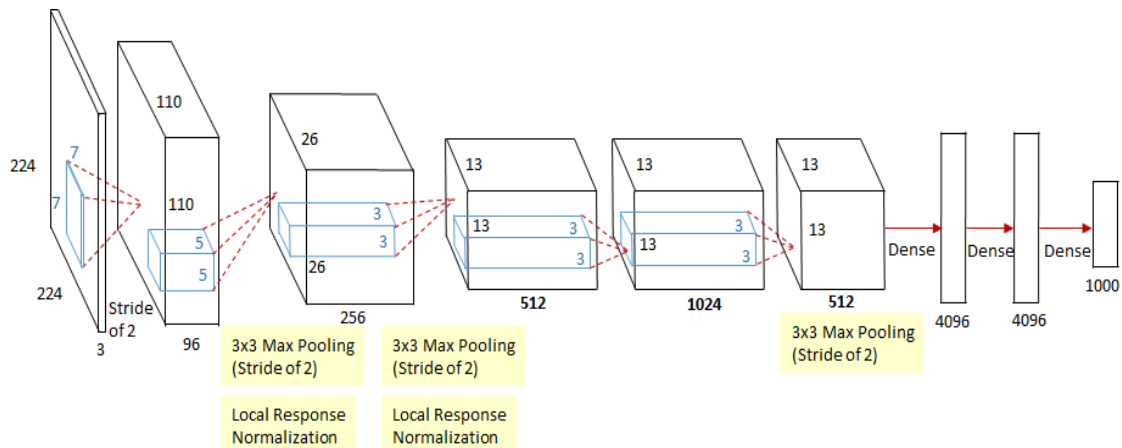


Fig. 2.2 ZFNet architecture

As you can see in the **Fig. 2.2**, the main difference with AlexNet is that the first layer has a smaller convolution structure, 7x7. Therefore, training is simplified and lower level features can be captured in the first layer.

2.1.3. VGG Net

The main feature of this network is that it uses 16 convolutional layers based on 3x3 modules. In the **Fig. 2.3** you can clearly see that, in the first layer, each of the output characteristics depends on a 3x3 region of the original image, in the second convolutional layer it depends on 5x5 of the original image and in the third one it depends on 7x7, etc. In summary, VGG Net [5] shows a simple structure where the lower layers can extract global features and the higher layers can process the global pixels of the image.

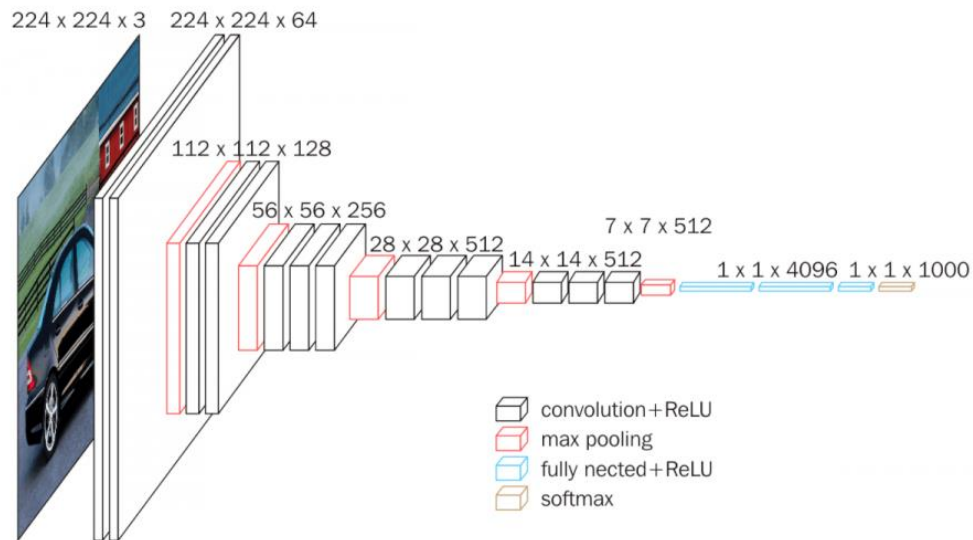


Fig. 2.3 VGG Net architecture

2.1.4. GoogleNet and Inception modules

GoogleNet [6] was the winner of Imagenet in 2014, it introduced the concept of inception modules which simplifies the design of neural network layers allowing the use of simple structures, concatenated in parallel or in series. The inception modules combine convolution cores of different sizes (Fig. 2.4). This was intended to solve the problem of the optimal size of the convolution layers and to take advantage of all the benefits offered by the different cores.

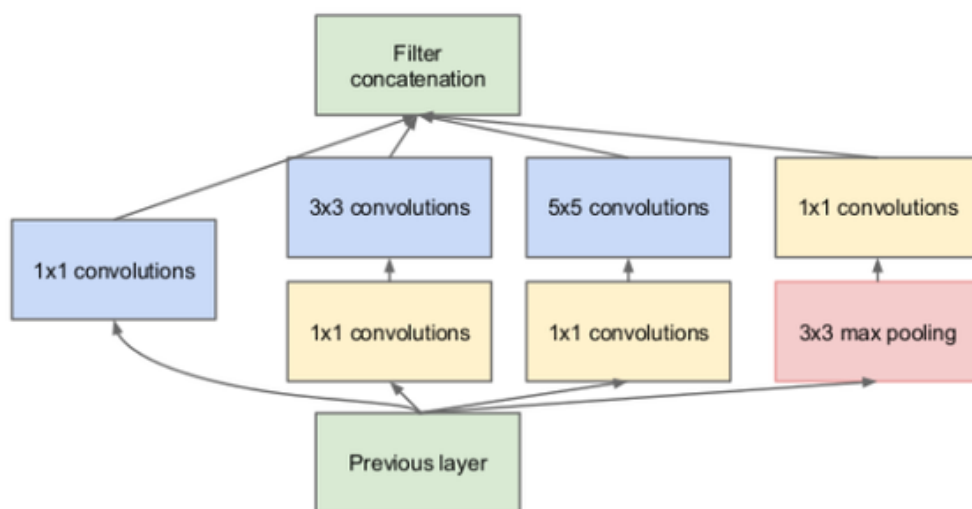


Fig. 2.4 Example of net with inceptions modules

The GoogleNet network has more than 100 layers, therefore it produces better results, but it has a big disadvantage. Since it has so many layers, the input signals from the last layers, which come from the first layers where the weights have not been trained, are not correct. For this reason, GoogleNet has a couple of outputs in the central part of the structure, which introduce the system error closer to the first process layers, in order to allow these layers to adapt their weights taking into account closer errors and facilitate the global learning of the network.

2.1.5. Microsoft Resnet

Microsoft won the Image challenge with the Resnet 152-layer proposal [7], obtaining an error rate of 3.6%. This network has some modules with forward feedback called residual blocks. The idea was to improve the error feedback from the output to the input layers.

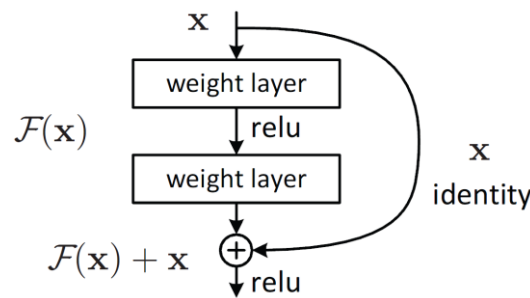


Fig. 2.4 Scheme of residual block

As shown in the **Fig. 2.5**, having a replica of the input itself at the output of the network, the learning algorithm should only learn the differences between the output and the input, which facilitates and accelerates the training significantly. The advantage of inception is that it simplifies the architecture of the model since it repeats the same topology throughout the network.

2.1.6. Xception

This model was implemented by Chollet in 2017 [8], it is very similar to inception but with computational improvements by using an enhanced version of the inceptions modules called eXtreme inception, which uses the parameters of the most efficient model. It introduces the concept of Depthwise Convolution (**Fig. 2.6**), where a 3-dimensional convolution core is decomposed into two convolutions, a 2x2 spatial one which is applied to all channels equally and a 1x1 deep one which is applied to the dimension of the layered channels. Thanks to this, the number of parameters is significantly reduced with respect to a real 3D

convolution, obtaining results in image classification better to VGG, Resnet and Inception V3.

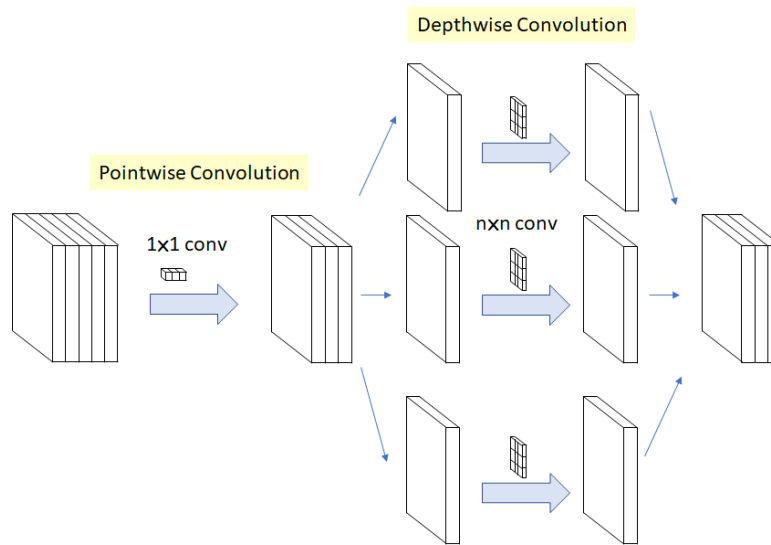


Fig. 2.6 Depthwise convolution - Xception

2.2. Object detection systems

Compared to a classification system, a detection system is used to identify the positions of multiple objects in an image by means of the coordinates of a rectangle called the "Bounding box" and also, it identifies the typology of the detected objects. Therefore, object detection is more complex problem than classification because in the latter case, only the class of the object has to be identified and not its position. In this section, the main architectures are reviewed, following a historical thread of those that have had the greatest technological impact.

2.2.1. Two-stage architectures

Two-stage architectures are the first to be used for object detection. They are systems of two-stage because they use an independent algorithm that does not use Deep learning to propose the regions (bounding boxes) on the original image and then, they use the images obtained from the regions as input data of the neural network to classify the objects. The most popular are R-CNN [9], FAST R-CNN [10] and FASTER R-CNN [11].

2.2.1.1. R-CNN

The R-CNN algorithm (**Fig. 2.7**) in the first stage, analyzes the original image and divides it up into regions (bounding boxes). These boxes are defined using a method known as Selective Search and are generally about 2000 per image. Afterwards, a wrapping is done for each of these regions, which consists of deforming the image until it gets a size of 227x227 pixels. Finally, this is the image which is introduced into the neural network. In addition, the network includes a layer based on the linear regression model to improve the proposal of location of the regions defined by the elective search.

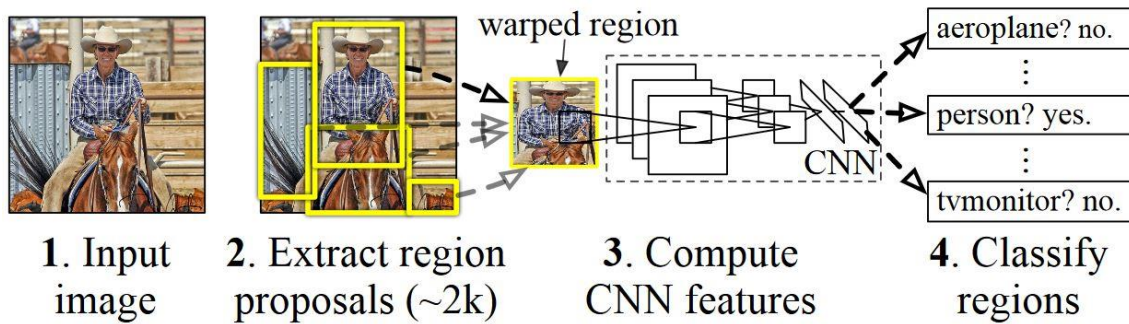


Fig. 2.7 R-CNN steps

2.2.1.2. FAST R-CNN

As its name suggests, FAST R-CNN (**Fig. 2.8**) is a modification of the previous algorithm, which improves its speed and accuracy. The speed improvement is based on processing the entire image, instead of each region, by a VGG network trained as a classifier using the Imagenet database. And the accuracy improvement is based on performing class estimation and bounding box regression coordinates using neural networks.

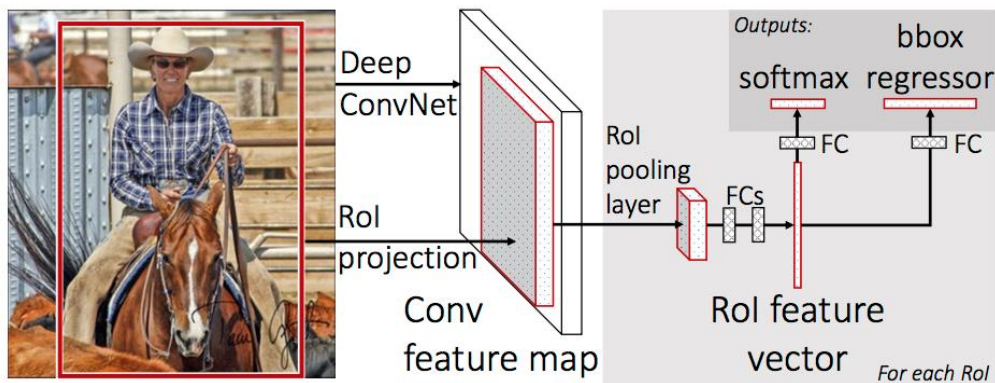


Fig. 2.8 Scheme of FAST R-CNN

2.2.1.3. *FASTer R-CNN*

The FASTer R-CNN algorithm [11] (**Fig. 2.9**) introduces a significant innovation by proposing the use of a neural network that directly performs the estimation of regions without the use of external algorithms, thus taking the first step to single-stage architectures. The basic idea is to implement the original image in a convolutional network, which will analyze it and it will obtain a set of characteristics. These features will be used as input to another neural network called RPN (Region Proposal Network) which replaces external region selection algorithms such as Selective Search. The RPN evaluates each anchor box candidate (first approach to the bounding boxes) using a confidence value and corrections to the anchor box candidate to match the object's actual bounding box.

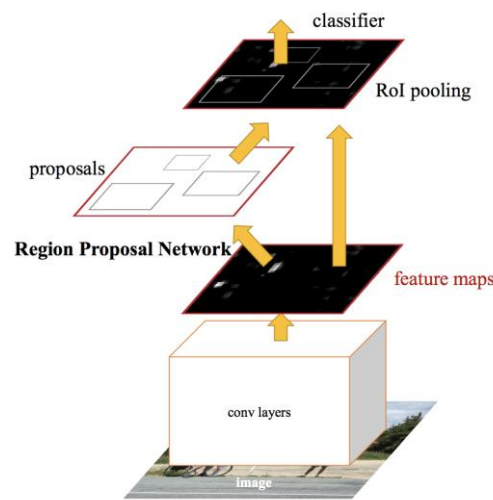


Fig. 2.9 Scheme FASTer R-CNN

2.2.2. One-stage architectures

One-stage architectures were developed to improve performance and speed up the process of detecting multiple objects with lower computational cost than two-stage architectures. These systems known as One-shot perform the detection of the different regions of the original image based on the idea of the FASTer R-CNN RPN, but they also add a new parallel network to obtain the classification of objects. The most used are YOLO [12] (You Only Look Once) and SSD [13] (Single Shot Multibox Detector).

2.2.2.1. YOLO

The basic idea of YOLO (**Fig. 2.10**) divides the original image into a grid of 7x7 cells and for each cell it estimates 2 bounding boxes and the category of the

object. So, the maximum number of bounding boxes that you can find is $7 \times 7 \times 2 = 98$. An object only will be assigned to a certain cell when its center is inside the cell.

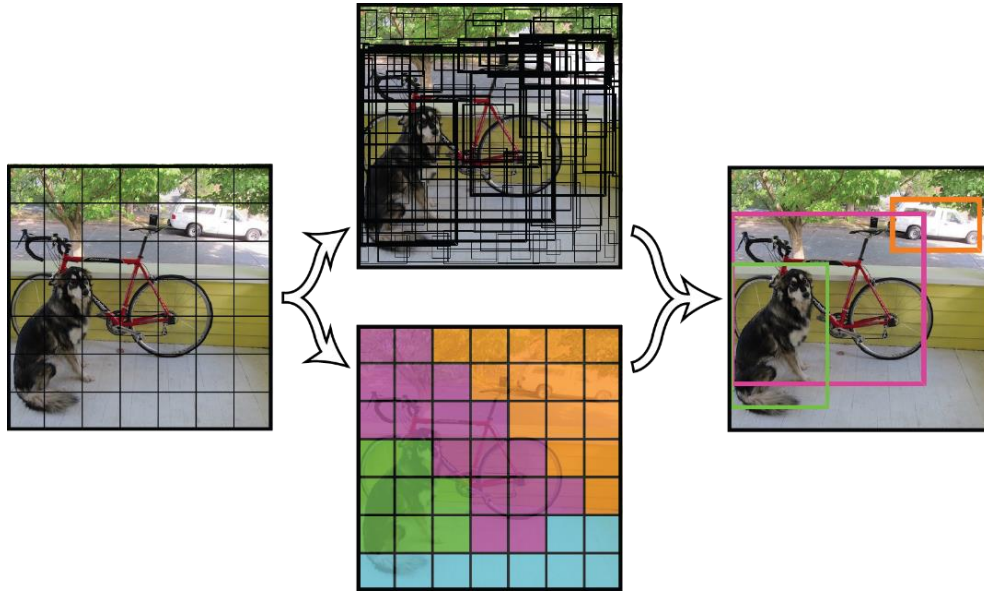


Fig. 2.10 Example of YOLO

Its architecture is based on the GoogleNet network formed by 24 convolutional layers plus 2 fully connected layers. The results in terms of detecting the correct class are close to those obtained with FAST R-CNN, although the accuracy in location is much lower.

2.2.2.2. SSD

The SSD architecture [13] (**Fig. 2.11**) aims to improve the results obtained with YOLO without losing quality. SSD defines default bounding boxes at different scales using the information provided by different intermediate layers and performs class and offsets prediction, using small (3x3) convolutional filters that are applied directly on the layer feature maps of a base convolutional network. The feature maps obtained in the last layers have lower resolution so they are responsible for detecting larger objects and the feature maps obtained in the first layers are responsible for detecting smaller objects.

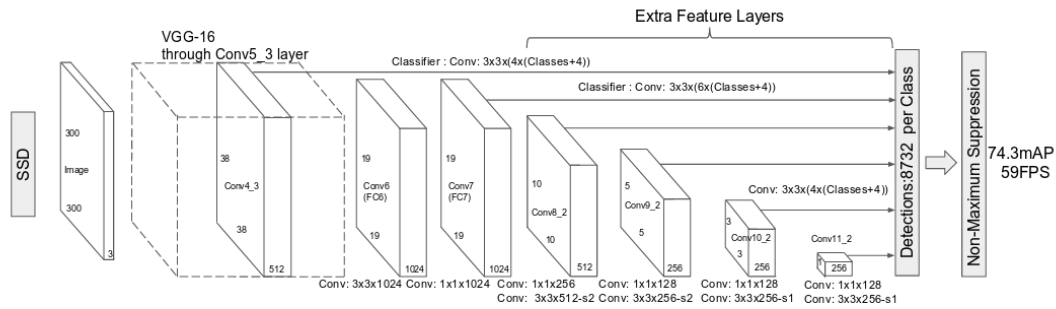


Fig. 2.11 SSD architecture

2.3. Multiscale architectures and Retinanet

2.3.1. FPN: Feature Pyramid Network

In networks such as VGG or Resnet it has been seen that the most superficial layers are in charge of calculating low level characteristics, such as contours, lines, etc. While the deepest layers, with lower resolution contain higher level semantic elements that allow the identification of objects. However, there is a correlation between semantic level and resolution whereby an increase in the previous one decreases the second one. The FPN architecture [14] (**Fig. 2.12**) aims at solving this problem by introducing to the backbone of a Resnet a second top-down structure where the high-level semantic features are propagated to the lower layers. The output of the RPN will be the features P2, P3, P4 and P5 that will represent the original image at different scales.

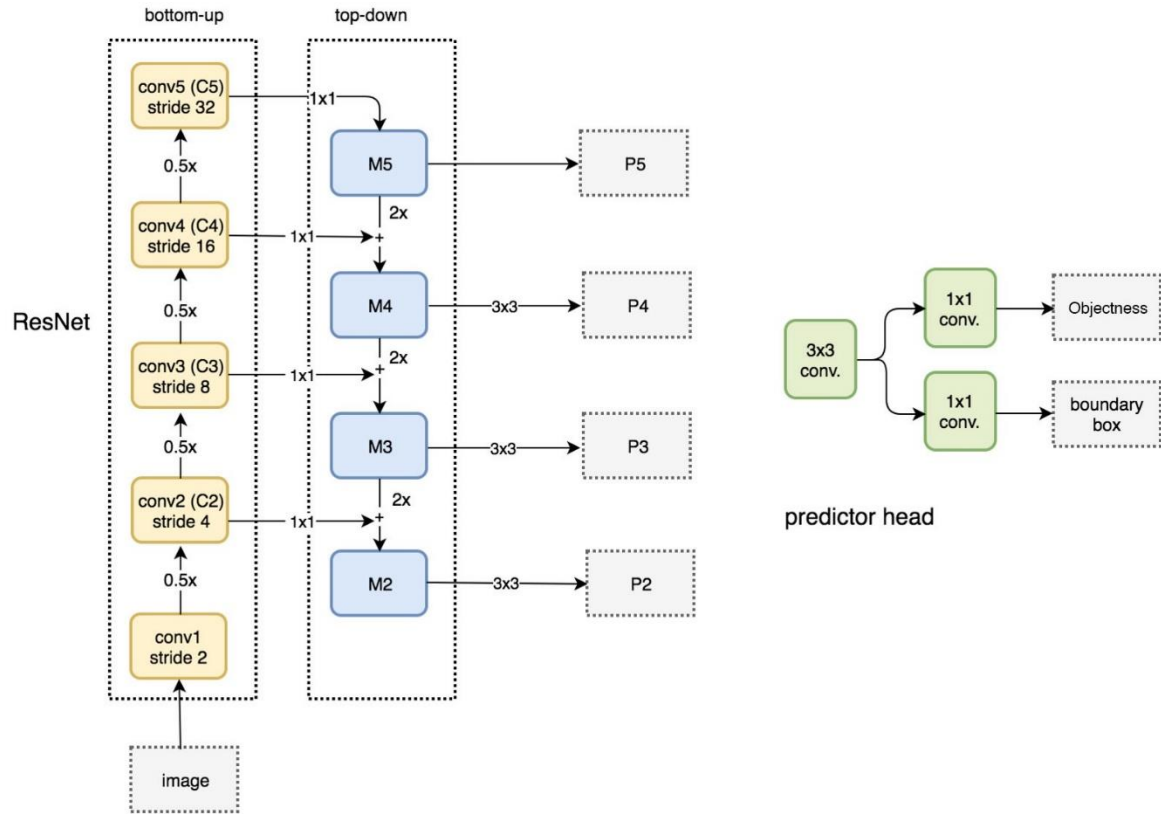


Fig. 2.12 Feature pyramid network architecture [14]

In order to calculate the characteristics at M5 level, a 1x1 convolution on C5 is applied and a total of 32 channels are calculated (M5 and C5 have the same resolution). To calculate M4, M5 is interpolated by a factor of 2, using the nearest neighbor strategy. On the other hand, the characteristics coming from the C4 layer of the Resnet are convolved with a 1x1 core with 256 output channels and these results are added, element by element, with the interpolation of M5. This process is repeated for each of the layers. Finally, the outputs P2, P3, P4 and P5 are obtained by applying 3x3 convolutions in order to smooth the aliasing effect due to the interpolation carried out.

2.3.2. RetinaNet

The RetinaNet model is a one-stage model [15], based on the FPN network, which introduces as a main innovation the loss function called focal loss. The basic idea is to use a pyramidal, multi-resolution, single-stage architecture, but with a loss function that allows for accuracies comparable to two-stage systems. Using the FPN architecture and the focal loss, better results are obtained than the two-stage methods, at all scales. The new loss function consists of:

$$L_{cls} = - \sum_{i=1}^K (y_i \log(p_i) (1 - p_i)^\gamma \alpha_i + (1 - y_i) \log(1 - p_i) p_i^\gamma (1 - \alpha_i)) \quad (2.1)$$

- K represents the number of classes, y_i takes the unit value when the object corresponds with the class and p_i represents the probability of object i.
- The parameter γ is denominated focal loss, it is used to give less importance to the easiest classes to be classified as it is in the case of the background, in other words, the most frequent ones. Therefore, the value $(1 - p_i)^\gamma$ will be a very small number for all the classes that have a very high probability.
- Alpha factor (α_i) is an alternative way of balancing the data. A small Alpha value is assigned to the classes that appear most frequently.

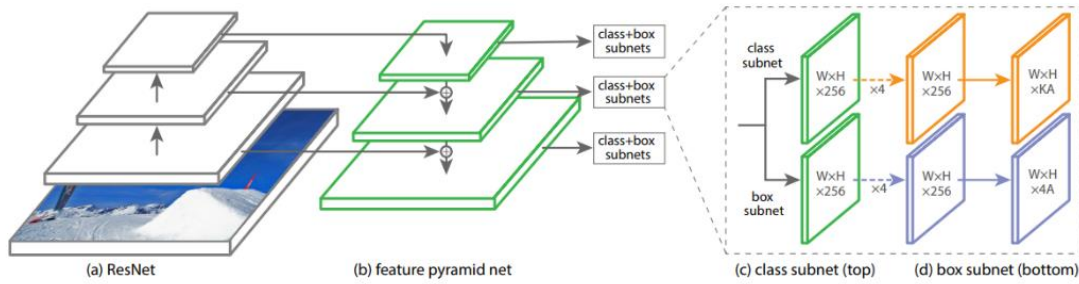


Fig. 2.13 Retinanet architecture [15]

The architecture of retinanet (**Fig. 2.13**) is based on two subnetworks which take the characteristics of the FPN levels. One subnet determines the class associated with each anchor box and the other performs the regression to adjust the position and size of the bounding boxes.

2.4. Choice of RetinaNet

RetinaNet system has been chosen because, currently, it is one of the systems that provides better results and has become a reference in object detection systems. Compared to previous networks, it offers a better accuracy by processing time thanks to the implementation of the focal loss that, solves the problem of class imbalance in the classification.

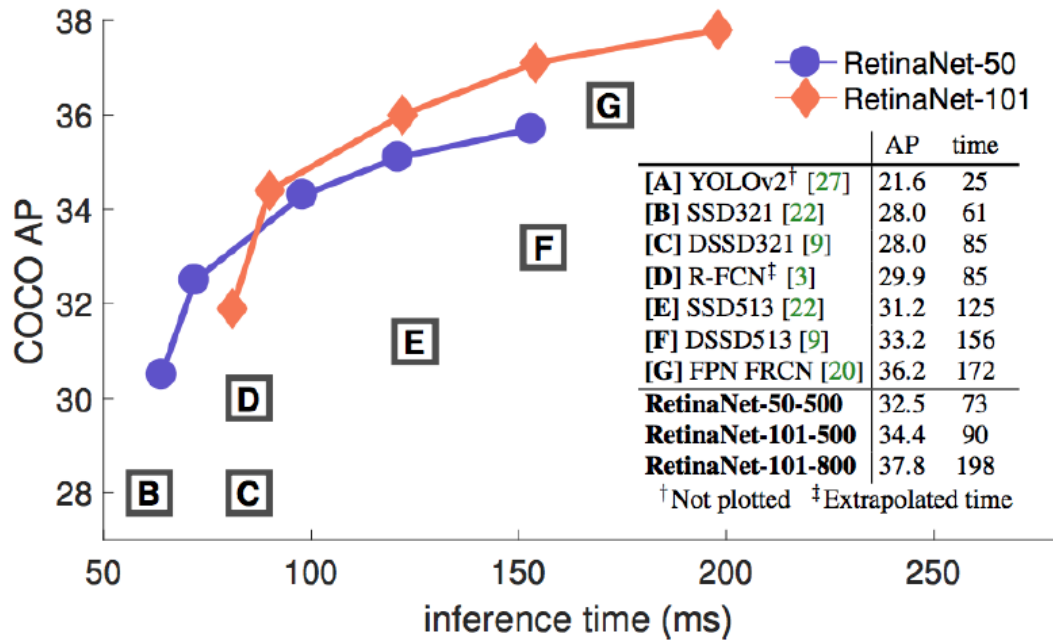


Fig. 2.14 Results of different models of average precision vs inference time [16]

The image above shows the average precision, in scale of COCO AP, vs the inference time of the different detectors and you can see that, for the models of Retinanet-50(backbone Resnet50) and Retinanet-101(backbone Resnet101), the accuracy is higher in a low inference time compared to the other single stage detectors, even higher than the two stage FRCN. It should be noted that these results have been made on a specific case and depending on the input data may benefit one type of network more than others. However, Retinanet shows a remarkable improvement in the field of real-time object detection and it is currently offering the best results. For these reasons, Retinanet has been chosen for this project.

CHAPTER 3. IMPLEMENTATION OF RETINANET FOR LOGO DETECTION

In the previous chapter, all the automatic classification and detection systems have been reviewed. In this chapter, it is going to be explained all the steps that have been followed to adapt the RetinaNet model to the detection and classification of logos in moto GP. The most important points have been the creation and preparation of the database, the training of our model using the script `train.py` of RetinaNet and finally the conversion of the trained model into an inferential one to make the predictions.

3.1. Development environment

When it has to work with object detection and it uses models like the ones explained above, it uses development frameworks that facilitate the implementation of the convolutional layers and they allow a simple configuration of the parameters of the training algorithms and the selection of training hyperparameters. These frameworks also allow the use of the GPU in training to accelerate the process. In this project it has used Keras, which is built on the basis of TensorFlow and implements modules of it (**Fig. 3.1**).

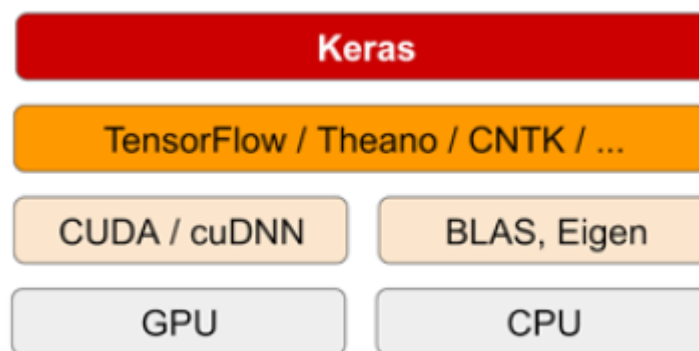


Fig. 3.1 Keras, Tensorflow and others frameworks [1]

The platform that has been chosen to work has been Google Colab, this platform gives the possibility to work using Google's servers for free, which facilitates the use of their GPUs. Currently, it has 4 GPUs: K-80, Tesla p-100, tesla P4 and tesla T4. The most common is that it assigns you the K-80 which is the worst. The best one is t4 and in order to be assigned it, the execution environment has to be reset

several times (rarely it is assigned to the first one if it is not a premium account). Another limitation of this service is that it can only be used for a limited time per day. The maximum is 12 hours, so you have to be careful when it is training the model because it could lose all the progress. To avoid this, it has been defined that, in each period, store the trained model.

3.2. Database creation

The database is essential for a neural network in order to learn, and thousands and thousands of examples are needed for this. The more examples a database has, the better the network will learn the generalities of the problem to be solved. For the creation of the data set, six videos of the 2019 Moto GP Grand Prix in HD (1080x720p) have been chosen and the logos of the brands: Alpinestar, DHL, Repsol, GoPro, Michelin, RedBull, Monster, Tissot, Motul and BMW have been selected. In this case, the database is made up of images of motorcycle GPs with the labelled logos. To do this, a module has been created that extracts a frame every two seconds from the video and, once the frames of all the videos have been extracted, the script `labelimg.py` has been used to label the images (Clarify that labeling an image means drawing the bounding boxes of all the objects that appear in an image and indicate which object it is). Each video has a duration of one hour, so it has been obtained about 10800 frames, which have been labeled one by one. On average, 5 logos can appear in an image, so creating this type of database is very entertaining and requires much time.

3.2.1. FFMPEG module for frame extraction

To extract a frame every two seconds it has been created the script `Extrac_frames.py` in Python. The code is showed in **Fig. 3.2**:

```

8 import cv2, shutil
9
10 film=4
11 input_loc = '/Users/david/Desktop/TFM/MOTO_GP_2019_VIDEOS/MotoGP_' + str(film) + '.mkv'
12 cap = cv2.VideoCapture(input_loc)
13 img_index = 0
14 print ("Converting video..\n")
15 # Start converting the video
16 while cap.isOpened():
17     # Extract the frame
18     cap.set(cv2.CAP_PROP_POS_MSEC,(img_index*2000))
19     ret,frame = cap.read()
20
21     if ret == False:
22         break
23
24     cv2.imwrite('frame' + str(img_index) + '.png', frame)
25     shutil.move('frame' + str(img_index) + '.png', 'MOTO_GP_2019_FRAMES/MotoGP_' + str(film))
26     img_index += 1
27
28 cap.release()

```

Fig. 3.2 Python code of frame extraction

The cv2 library has been used: to read the video, extract the frames every two seconds, read them and save them in jpg format. Finally, the shutil module has been used to move the frames to the corresponding folders of each video with the 'move' method.

3.2.2. Frames labeling

The tool labellmg.py has been used to label the frames extracted. This tool allows to annotate the delimiting boxes of the objects (**Fig. 3.3**) in order to train the automatic learning model.

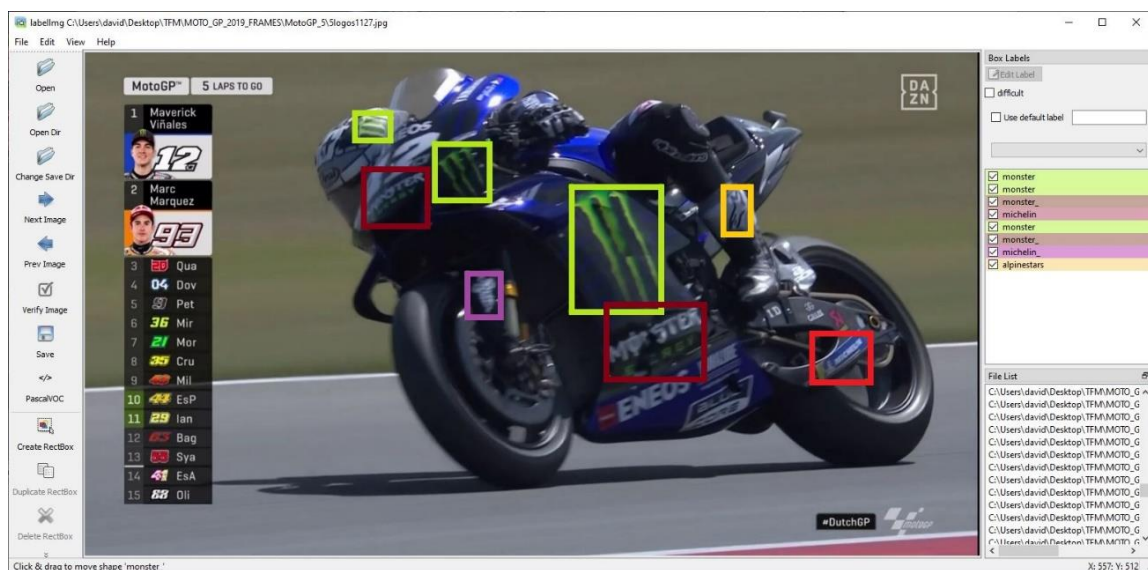



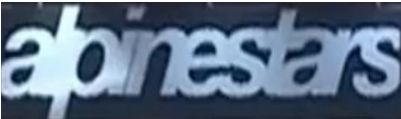







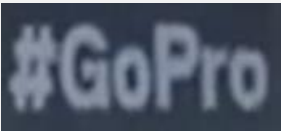
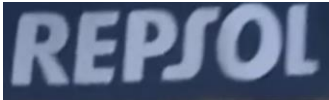

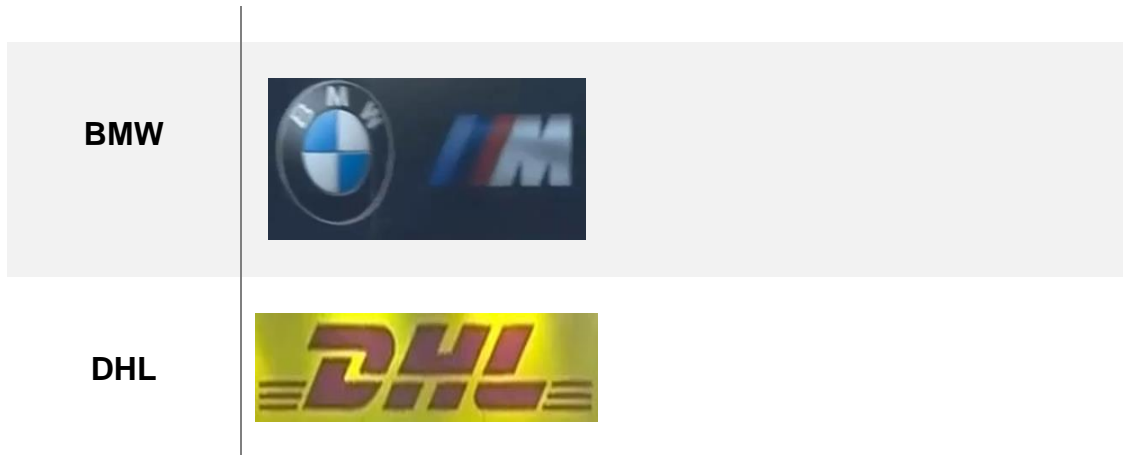


Fig. 3.3 Labeling frames with Labelimg

16 classes have been defined for the 10 brands, since the same brand, such as Monster or Alpinestar, can appear with different logos. Therefore, it has been decided to create a class for each type of logo to avoid confusion in the classification. The classes that have been defined are shown in the **Table 3.1**:

Table 3.1 Classes defined with his respective brands

BRANDS	CLASSES	
MONSTER		
ALPINESTAR		
REDBULL		
MICHELIN		
TISSOT		
MOTUL		
GOPRO		
REPSOL		



The data of the labeled image is saved in PascalVoc format, which is an xml file with all the information of the labeled objects, as you can see in the image:

```
<annotation>
  <folder>MotoGP_5</folder>
  <filename>5logos609.jpg</filename>
  <path>C:\Users\david\Desktop\TFM\MOTO_GP_2019_FRAMES\MotoGP_5\5logos609.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>1920</width>
    <height>1080</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>repsol</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>904</xmin>
      <ymin>372</ymin>
      <xmax>1180</xmax>
      <ymax>511</ymax>
    </bndbox>
  </object>
</annotation>
```

Fig. 3.4 Example of xml file PascalVoc format

An xml file is generated for each image. Through these files, the network is not ready to take the data, so a further step has to be taken. The xml files have to be parsed to get the object name and the bounding box coordinates: xmin, ymin, xmax and ymax. And with this information two csv files have to be generated.

3.2.2.1. Convert xml files to Retinanet csv format

In order to use the Retinanet network, two csv files have to be created: annotations and classes.csv. The first one will be used for model training and predictions. It has the following format:

path/to/image.jpg,x1,y1,x2,y2,class_name

Where x1, y1 will be the points on the left of the box and x2, y2 the points on the right. Each object is to be saved as a new csv line. The following code (**Fig. 3.5**) extracts the information from the xml and stores it in annotations.csv.

```
[ ] 1 ANNOTATIONS_FILE = 'annotations.csv'
    2 CLASSES_FILE = 'classes.csv'
    3 DATASET_DIR = 'dataset'
    4 annotations = []
    5 classes = set([])
    6 name = None
    7 file_name = None
    8
    9 for xml_file in os.listdir(DATASET_DIR):
   10     if xml_file.endswith('.xml'):
   11         root = ET.parse(os.path.join(DATASET_DIR, xml_file)).getroot()
   12         for elem in root:
   13             if elem.tag == 'filename':
   14                 file_name = os.path.join(DATASET_DIR, elem.text)
   15             if elem.tag == 'object':
   16                 coords = []
   17                 for subelem in elem:
   18                     if subelem.tag == 'name':
   19                         name = subelem.text
   20                     if subelem.tag == 'bndbox':
   21                         for subsubelem in subelem:
   22                             coords.append(subsubelem.text)
   23                 item = [file_name] + coords + [name]
   24                 annotations.append(item)
   25                 classes.add(name)
   26
   27
   28 with open(ANNOTATIONS_FILE, 'w') as f:
   29     writer = csv.writer(f, lineterminator='\n')
   30     writer.writerows(annotations)
```

Fig. 3.5 Python code that convert xml file to Retinanet csv format

The **Fig. 3.6** is an example of how the data extracted from the xml has been saved within annotations.csv:

```
dataset/0logos306.jpg,1634,253,1903,332,monster_  
dataset/0logos306.jpg,276,247,431,335,monster_  
dataset/0logos434.jpg,707,454,741,518,monster_  
dataset/0logos881.jpg,1038,568,1091,675,michelin_  
dataset/0logos881.jpg,999,578,1055,685,alpinestars_  
dataset/0logos881.jpg,967,575,1004,696,alpinestars_  
dataset/0logos881.jpg,1253,542,1293,594,monster_  
dataset/0logos911.jpg,1142,771,1262,811,michelin_
```

Fig. 3.6 Example of data in annotations file

After creating the annotation.csv file, the following code, showed in **Fig. 3.7** has been used to divide it into two csv files, one for training called train_annotations.csv which is 80% of the total and the other for predictions which is the remaining 20%. This division has been done by saving the samples in a random and non-sequential way, because in this manner more variety of samples is obtained in both parts.

```
1 import pandas as pd  
2 from sklearn.model_selection import train_test_split  
3  
4 annotations = pd.read_csv("annotations.csv")  
5 train, test = train_test_split(annotations, test_size = 0.2, shuffle = False)  
6 print('The numbers of annotations for train: ', len(train))  
7 print('The numbers of annotations for test: ', len(test))  
8 train.to_csv(path_or_buf='train_annotations.csv', index=False)  
9 test.to_csv(path_or_buf='test_annotations.csv', index=False)
```

Fig. 3.7 Data division into 80% of training set and 20% of validation set

El archivo de clases.csv se ha generado a mano y contiene las 16 clases enumeradas de 0 a 15 correspondientes a las marcas escogidas. Una vez generados estos dos archivos, el siguiente paso es la configuración de Retinanet para entrenar el modelo.

3.3. Quality Metrics

Before explaining the training and prediction phases, it is important to know what kind of metrics have been used in this project to define the quality of the model.

3.3.1. Intersection Over Union (IOU)

It is a measure that defines the overlap between two regions. It is calculated as the area of overlap between the total area of the sum of the two regions (Fig. 3.8).


$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Fig. 3.8 Representation of IOU equation

When the predictions are made, the real bounding box, called groundtruth, is compared with the one detected by the model and the IOU is performed. Usually it is considered a good detection when the IOU is greater to 0.5.

3.3.2. mean Average precision (mAP)

The mAP measurement is the average of the average accuracy (AP). The AP is a measure that calculates the area formed by the precision vs recall curve.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

TP = True positive
 TN = True negative
 FP = False positive
 FN = False negative

Fig. 3.9 Precision and recall equations

TP is when the detection has been correct (IOU greater than 0.5); TN is when the detection is not correct, that is, when there is no real box and the detection system does not detect anything in that area; FP is an incorrect

detection (IOU less than 0.5); FN is when it does not predict a box and in the groundtruth there is one.

As it can be observed in **Fig. 3.9**, accuracy represents the ratio of correctly detected boxes among all detected boxes and recall represents the number of correctly detected boxes among all groundtruth boxes. The ideal of these values is that they will come closest to the unit.

3.3.3. Accuracy

Accuracy is a parameter used to measure the quality of a multi-class classification system. It is defined as the ratio of the number of correctly classified cases to the total number of cases.

3.3.4. Confusion matrix

The confusion matrix is an $N \times N$ matrix where N is the number of classes. It represents how the model classifies all the predictions, both those it classifies well and those it confuses. An ideal confusion matrix would be all zeros except the diagonal, which are the well classified cases. On the other hand, if numbers appear outside of the diagonal it is that it is confusing the class. Knowing this, another way to calculate accuracy would be the ratio of the sum of the diagonal to the sum of all the numbers in the matrix.

3.4. Training with Retinanet

With the database already created in the right format, the next point is to train the model. Because a database has not been created large enough to train a neural network from scratch, it has been decided to use the technique known as transfer learning. This technique, showed in **Fig. 3.10**, consists of using the weights of a pre-trained model through other databases, freezing the weights of the convolutions layers and eliminating the weights of the last layers. Only the weights of the convolutional base are used, since this part of the network is more generic than the densely connected one (the last layers). Therefore, the weights can be used in other networks. In contrast, the features in the last layers only contain information about the probability that a class has to be in the image.

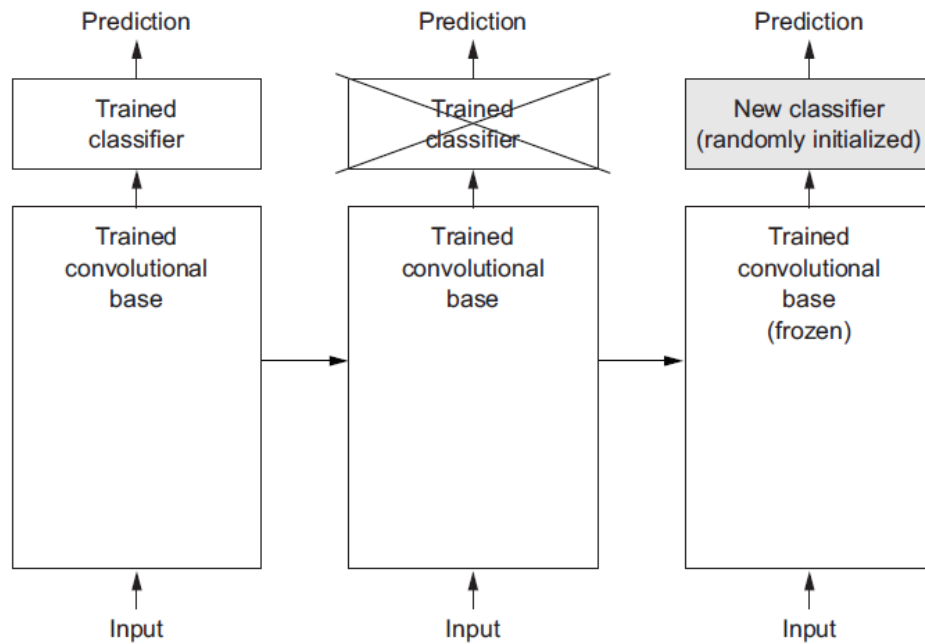


Fig. 3.10 Transfer learning technique

For the training, the script `train.py` has been used which is in the directory: `keras_retinanet/bin/train.py`. This script allows you to configure the training by passing different parameters as arguments. The following parameters have been defined:

- **Epochs:** Define the times that it is desired to use the same input data to train the neural network.
- **Batch-size:** This is the number of samples used to calculate an update of the weights.
- **Steps:** Number of steps per epoch.
- **Freeze-backbone:** It freezes the layer weights of the backbone model, i.e. it impedes its learning. Useful when it uses a small dataset, to avoid overfitting.
- **Val-annotations:** Path to `test_annotations.csv` file, shows you in each epoch the accuracy and the number of instances of each object.
- **Weights:** Initialize the model with weights from a file.
- **Tensorboard-dir:** Log directory for Tensorboard output.
- **Compute-val-loss:** Compute validation loss during training.

```
1 !keras_retinanet/bin/train.py --epochs 25
2                               --steps 500
3                               --batch-size 8
4                               --weights {'resnet50_coco_best_v2.1.0.h5'}
5                               --freeze-backbone
6                               --tensorboard-dir tensor/tensor
7                               --compute-val-loss
8                               --val-annotations test_annotations.csv
9                               csv train_annotations.csv classes.csv
```

Fig. 3.11 Training configuration

As shown in the **Fig. 3.11**, 20 epochs of 500 steps have been performed with a batch-size of 8 (the GPU has a limited memory so with a higher value the Google Colab server can collapse). The Resnet50 architecture has been used as a backbone since the training is faster than the Resnet101 and the Resnet152, and the results obtained were very similar. On this architecture the weights of the pretrained model `resnet50_coco_best_v2.1.0` have been loaded and the layers of the backbone have been frozen, so the layers that have been trained have been those of the FPN network and the two subnets, this is called finetuning. With this configuration, the network has been trained and the data file has been generated, which has been called `resnet50_logos_20.h5`.

3.4.1. Training with data augmentation

In order to improve the results, it has been decided to use data augmentation with the same configuration explained above. This technique is useful when you have a small database. Data augmentation generates new examples, performing transformations on the images in the database.

```
# create random transform generator for augmenting training data
if args.random_transform:
    transform_generator = random_transform_generator(
        min_rotation=-0.1,
        max_rotation=0.1,
        min_scaling=(0.9, 0.9),
        max_scaling=(1.1, 1.1),
    )
    visual_effect_generator = random_visual_effect_generator(
        contrast_range=(0.9, 1.1),
        brightness_range=(-.1, .1),
        hue_range=(-0.05, 0.05),
        saturation_range=(0.95, 1.05)
    )
```

Fig. 3.12 Code of data augmentation with the transformations used

The image above shows the transformations that have been made to generate more data. Simple transformations have been used since many of the images obtained from the detected logos are already distorted enough. The transformations that have been used have been:

- **Min_scaling, max_scaling:** Variation in image size. It has been defined that the increase and decrease of the original image is about 10%.
- **Min_rotation, max_rotation:** Rotate the original image by 10%.
- **Contrast_range:** Variation of the contrast of the original image between 0.9 and 1.1.
- **Brightness_range:** variation in brightness between -0.1 and 0.1.
- **Hue_range:** Hue variation between -0.05 and 0.05.
- **Saturation_range:** Saturation variation between 0.95 and 1.05.

Inference time has increased but results have improved significantly. The following is a comparison between the two trained models, with and without data augmentation.

3.4.2. Comparison of results

The following are the results obtained during the training. First, for the case without data augmentation and later, for the case with data augmentation. It has used the metrics mAP and classification loss by epochs. The expected result is that the model which has been trained with a greater number of data will obtain better results.

Whithout data aumentation:

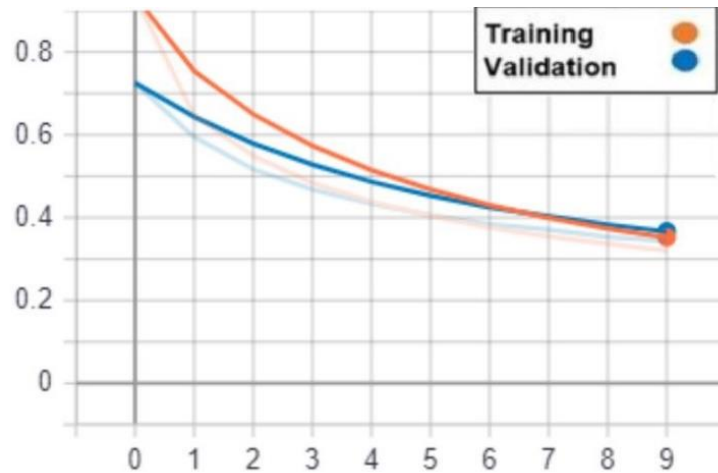


Fig. 3.13 Classification losses by epochs of the model without data augmentation

The loss by classification defines the percentage of misclassified logos over the total. The graph above shows the evolution of these losses during the training epochs. The orange line indicates the losses on the training data and the blue line indicates the losses on the validation data. It can be seen that from period 6 onwards the decrease in validation losses slows down compared to training losses. This is due to the fact that the weights are updated exclusively for the training data, so the check with the validation data serves to obtain more realistic results. The value of the validation data is the one that is relevant and has to be taken into account. In this case, a score loss of 0.386 has been achieved.

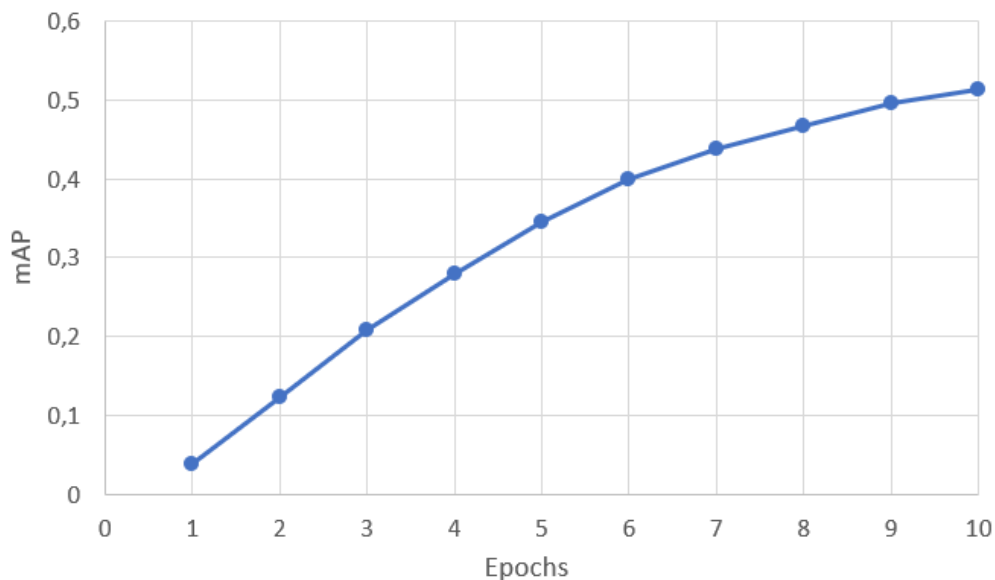


Fig. 3.14 mAP by training epochs of the model without data augmentation

The graph above (**Fig. 3.14**) shows the mAP values obtained in each epoch. The mAP has been improving in all the periods until reaching a value of 0.52. Taking into account that there is a total of 17 classes and that a random function would

give a value of 0.058, the model has made a prediction almost 10 times better than a random system.

With data aummentation:

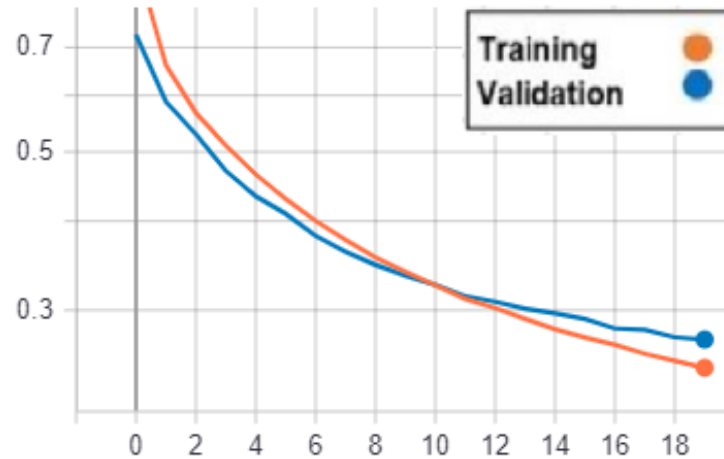


Fig. 3.15 Classification losses by epochs of the model with data augmentation

For the case with data augmentation, it can be seen in **Fig. 3.15** that from the 11th epoch onwards the decrease in validation losses is slowed down compared to training losses. In this case, a score loss of 0.273 has been achieved which, compared to the previous case, which is 0.386, is better.

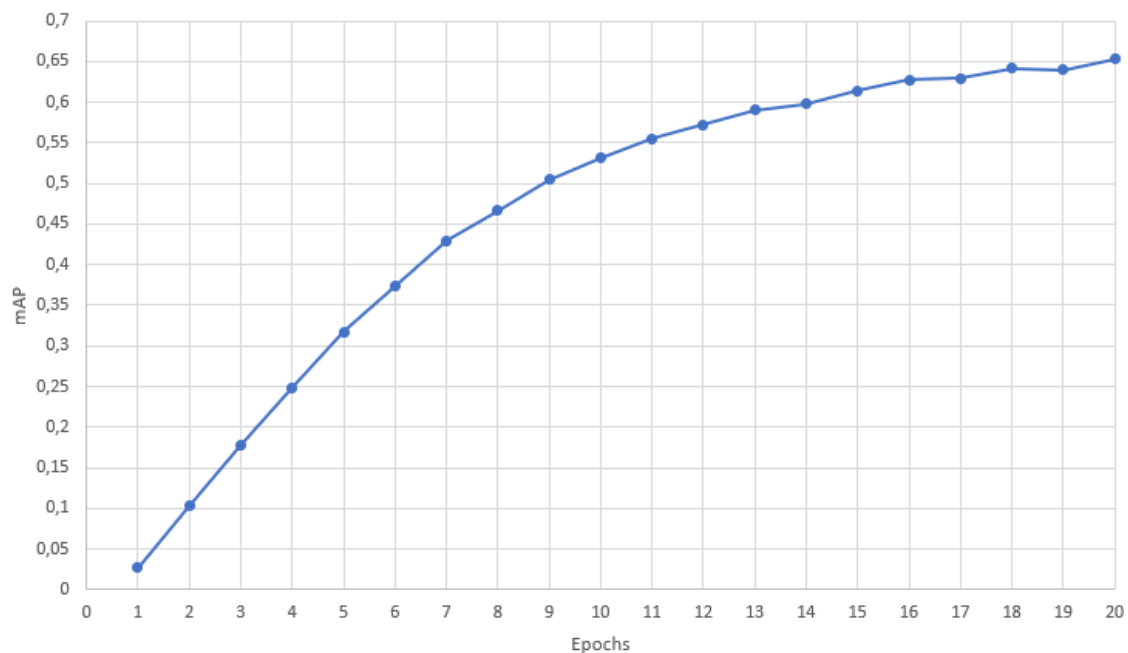


Fig. 3.16 mAp by training epochs of the model with data augmentation

And in the graph of the mAP (**Fig. 3.16**) it can be seen that the most optimum value has been 0.6533 and it has obtained in the last period (it is true that from period 18 to 19 there has been a slight decrease, this would be indicative of overfitting, but as it has happened in 1 period and it was the penultimate one, it has not been considered). This value is closer to unity than the value obtained without data augmentation, so this model is more accurate. With this, it can be concluded that the model that works better is the one trained with data augmentation.

3.5. Predictions with Retinanet

At the end of the training, the trained model is saved in a data file, so that it can be loaded and used in the predictions. To make the predictions the model has to be converted into an inference model. This model will predict, over the unlabeled images, the detection and classification of the logos and will compare them with the real examples (groundtruth) saved in `test_annotations.csv`. In order that the predicted images will be the same as those in `test_annotations.csv`, a program has been made to access to the file and obtain the paths where the images are located and it analyses them.

Retinanet assigns a score, ranging from 0 to 1, to each detected object. When making the predictions, it has to define the score value that is going to be used as the threshold for logo detection. Only those that exceed the score will be detected as valid boxes. The higher this score is, the more both bounding boxes will coincide, the classification will be better and the accuracy will be better, but, the number of detections will be lower. The threshold has been changed until obtaining the optimum value (the one that allows to detect an amount as close as possible to the number of real objects). The threshold selected has been 0.89, with this value it detects 105%, which means 5% more than those labelled. This 5% can be background or well detected logos that have not been labeled due to the error factor in human detection.

3.5.1. Results

The results to evaluate the quality of the system in the prediction, have been obtained from the accuracy and matrix confusion metrics. For the calculation of the accuracy, the detections on the background have been taken into account, that is, those regions that the model has predicted when in the groundtruth there is no box. For this case, the background class has been created. It should be

noted that when the database was created, it is possible that logos were left unlabeled. Therefore, the prediction may have detected this logo and, as it is not labeled, it is classified as background. For this reason, it has been decided to give less importance to the classification as background. Specifically, a value of 0.2 has been defined for the background class and 1 for the others. With this, an accuracy of 83.03% has been obtained.

CONFUSION MATRIX:

[1528	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	790]
[0	753	0	0	0	0	0	0	0	0	0	0	0	0	0	0	825]
[0	0	445	0	0	4	0	0	0	0	0	0	0	0	0	0	825]
[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	252]
[0	0	0	0	283	0	0	0	0	0	0	0	0	0	0	0	222]
[0	0	0	0	0	330	0	0	0	0	0	0	0	0	0	0	772]
[0	0	7	0	0	0	549	0	0	0	0	0	0	0	0	0	717]
[0	0	0	0	0	0	0	117	0	0	0	0	0	0	0	0	238]
[0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	135]
[0	0	0	0	0	0	0	0	0	670	0	0	0	0	0	0	147]
[0	0	0	0	0	0	0	0	0	0	487	0	0	0	0	0	320]
[0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	0	205]
[0	0	0	0	0	0	0	0	0	0	0	0	21	0	0	0	150]
[0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	91]
[0	0	0	0	0	0	0	0	0	0	0	0	0	0	518	0	81]
[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	113	187]
[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]

Fig. 3.17 Confusion matrix

The confusion matrix (**Fig. 3.17**) shows that the classification is quite good, since in this case only 7 times class 2 (Redbull) has been confused with class 6 (Repsol) and 4 times class 5 (Michelin) with class 2 (Redbull) has been confused. All other errors shown in the last column are detections that have been classified as background and, as mentioned above, it weighs 0.2 with respect to the other classes.

3.5.2. Inferences

In this section are exposed some examples of the detections when running the model as inference (**Fig. 3.17**, **Fig. 3.18**). These inferences have been obtained with a threshold of 0.55, where the objects detected with a score lower than this threshold are not valid and, therefore, are not shown.

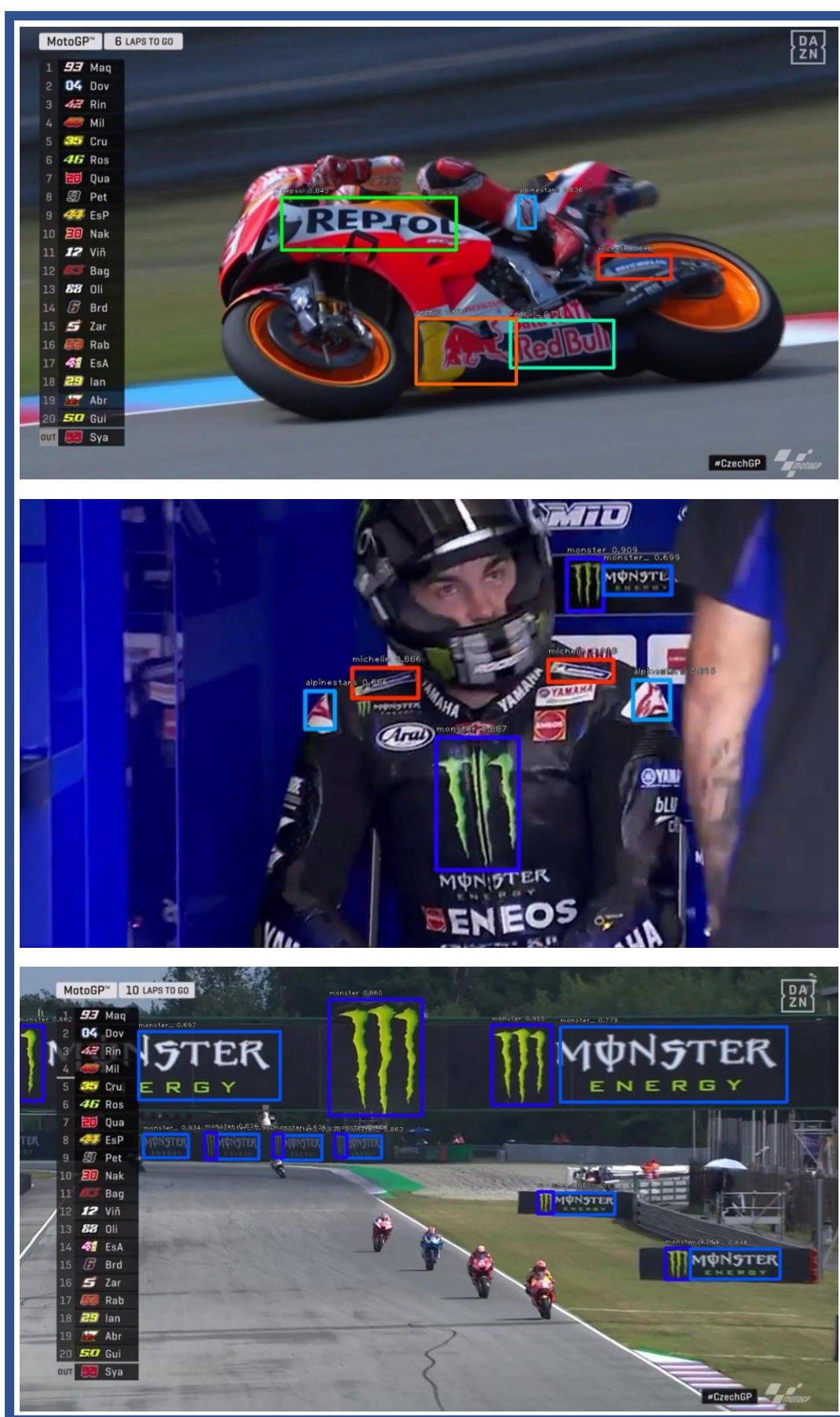


Fig. 3.17 Examples of inferences

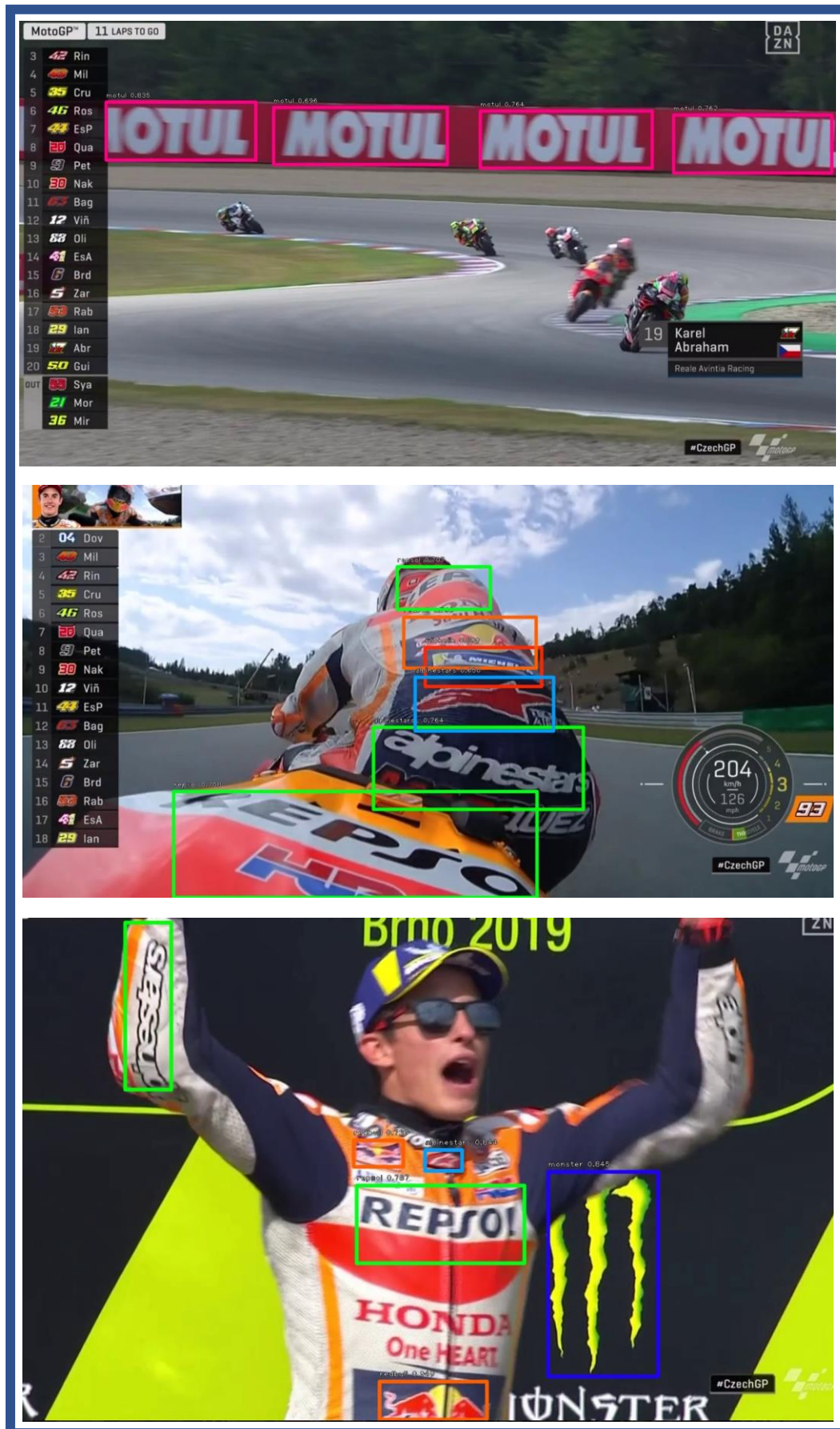


Fig. 3.18 Examples of inferences

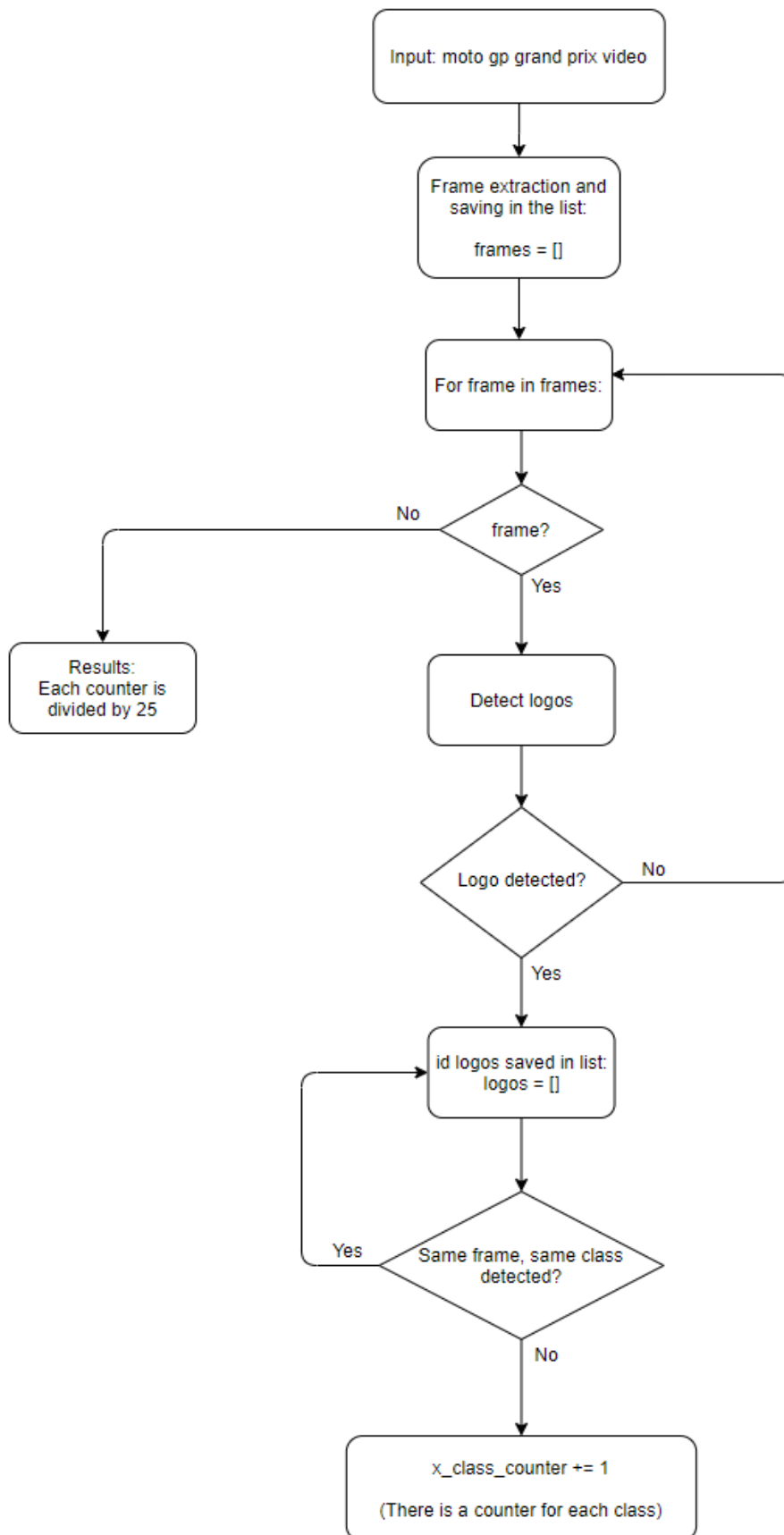
CHAPTER 4. APPLICATION EXAMPLE: BRAND LOGO MONITORING IN MOTO GP

4.1. App definition

The function of the app is to count automatically the time of appearance of each brand in the MotoGP Grand Prix and give a final result with the total time. The model that has been trained with data augmentation will be the one to perform this detection automatically and by applying the algorithm, which is explained in the following point, it will obtain the times of each brand.

4.2. Algorithm

The program algorithm is schematically represented in the following flowchart (**Fig. 4.1**), which is intended to represent the operation of the app in a simple way. From the video, all its frames will be obtained and saved in a list. Once extracted, the detection of logos will be done for each frame in the list. The logos detected in an image will be saved in another list, each logo is equivalent to an integer defined in `classes.csv`, and if the number is repeated within this list the counter will not count it (it will only count 1 logo per class in the same image). Finally, when all the frames have been analyzed, each counter will be divided by 25 (the videos are 25 fps) to obtain the total number of seconds and will become to the next format HH:MM:SS. The times will be represented on screen in a ranking, where the first will be the brand with the highest time on screen during the race.

**Fig. 4.1** Algorithm of final app

4.2.1. Total times results

In this section, it has analyzed the Spanish motorcycling Grand Prix and as it is shown in **Fig. 4.2.** the marks that appear above 25 minutes are Redbull, followed by Alpinestars, Monster and Michelin. Repsol is above 20 minutes and below 10 minutes are: Motul, Tissot, DHL, GoPro and Motul.

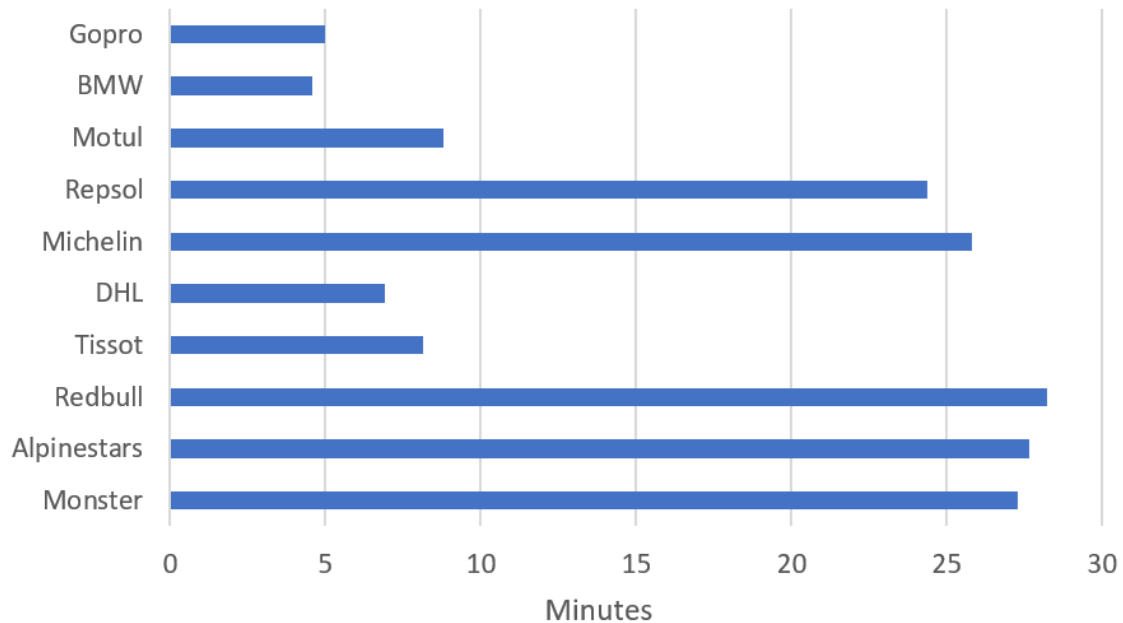


Fig. 4.2. Total times calculated by the experimental app

4.3. Environmental Impact

The developed software can be executed perfectly in any computer with a GPU, it is advisable to use an Nvidia for compatibility with the frameworks used. The power consumption of the current graphic cards has increased a lot. Superior models demand between 110 and 270 watts from the power supply.

CONCLUSIONS

Society is in a time where the amount of data stored is immense thanks to current technology, this is ideal for the development of artificial intelligence systems since they need to work with immense data banks. In this case, there was no specific database of brand logos, so it has been necessary to create one from scratch. This is a simple task but one that must be dedicated a great deal of time because the number of samples required must be quite large for a network in order to learn well how to generalize the information and obtain good results. Then, thanks to the transfer learning techniques, it has been possible to train a model from the weights of another pre-trained model, which has facilitated the learning of the RetinaNet network for logo detection. In addition, to avoid overfitting, the backbone has been frozen (Resnet50). After this, another model has been trained applying data augmentation to improve the results, which has improved in mean average precision with respect to the previous one, obtaining a value of 0.6533. Finally, this model has been chosen and used for the final application, which calculates the time of appearance of the marks in the MotoGP races. This application automates the process of counting the time of the marks and therefore, the acquisition of this information will be faster.

REFERENCES

- [1] Chollet, F., *Deep Learning with python*, Manning, United States (2017).
- [2] Torres, J., “Redes neuronales convolucionales”, *Deep Learning – Introducción practica con Keras (Primera parte)*, www.torres.ai.
- [3] I. S. Alex Krizhevsky Geoffrey E Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Neural Inf. Process. Syst. Found. 2012 Conf.*, pp. 1–9, 2012.
- [4] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks arXiv:1311.2901v3 [cs.CV] 28 Nov 2013,” *Comput. Vision–ECCV 2014*, vol. 8689, pp. 818–833, 2014.
- [5] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *Int. Conf. Learn. Represent.*, pp. 1–14, 2015.
- [6] C. Szegedy, W. Liu, Y. Jia, and P. Sermanet, “Going deeper with convolutions,” *arXiv Prepr. arXiv 1409.4842*, pp. 1–9, 2014.
- [7] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” 2016.
- [8] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 1800–1807, 2017
- [9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 580–587, 2014.
- [10] R. Girshick, “Fast R-CNN,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2016, vol. 11-18-Dece, pp. 1440–1448.
- [11] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *Nips*, pp. 1–10, 2015.
- [12] D. Impiombato et al., “You Only Look Once: Unified, Real-Time Object Detection,” *Nucl. Instruments Methods Phys. Res. Sect. A Accel. Spectrometers, Detect. Assoc. Equip.*, vol. 794, 2015.
- [13] W. Liu et al., “SSD : Single Shot MultiBox Detector,” pp. 1–15.
- [14] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature Pyramid Networks for Object Detection,” *1612.03144V1*, p. `` , 2016.

- [15] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-Octob, pp. 2999–3007, 2017.
- [16] Jonathan Hui, "Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)", 2018
- [17] L. Zhang, L. Lin, X. Liang, and K. He, "Is Faster R-CNN Doing Well for Pedestrian Detection?," *Eccv 2016*, pp. 1–15, 2016.