

E.T.S. de Ingeniería Industrial,  
Informática y de Telecomunicación

# Real-time vehicle detection and tracking in video surveillance cameras



Grado en Ingeniería Informática

Trabajo Fin de Grado

Author: Urtzi Sotés Senosiain

Director: Mikel Galar Idoate

Pamplona-Iruña, 05/03/2020

## Abstract

This End of Degree Work is directed to the field of Artificial Intelligence, expressly it is focused on the study of the Deep Learning techniques. Particularly, convolutional neural networks are studied with the aim of developing a vehicle detection model that is able to recognize them in images and videos. YOLOV3 is the vehicle detection model that has been studied, trained and optimized all through this work, in order to develop a model capable of detecting cars, trucks and motors in real time. Finally, SORT tracking algorithm has been studied and joint with the YOLOV3 detection model into a final model which will be responsible of detecting and tracking vehicles all through different videos in real time.

## Key Words

- Machine learning
- Deep learning
- Convolutional neural network
- Real-time object detection
- Real-time object tracking
- Multiple object tracking
- Traffic estimation

## Index

1. INTRODUCTION .....	4
1.1 Motivation and objectives.....	5
2. PRELIMINARIES.....	6
2.1 Introducing Machine Learning .....	6
2.2 Introducing Deep Learning techniques .....	8
2.3 Convolutional Neural Networks (CNN) .....	10
2.4 Object-detection .....	16
3. OBJECT-DETECTION WITH YOLO .....	18
3.1 What is YOLO?.....	18
3.1.1 YOLO architecture .....	22
3.1.2 Loss function and training .....	23
3.2 YOLOV2.....	27
3.3 YOLOV3.....	31
4. TRACKING WITH SORT.....	35
4.1 Estimation model .....	35
4.2 Data association .....	36
4.3 How the Hungarian algorithm and the Kalman filter work.....	36
4.4 Creation and deletion of track identities .....	37
5. YOLO AND SORT FOR OBJECT-DETECTION AND TRACKING IN REAL TIME .....	38
5.1 Dataset .....	38
5.2 Train, evaluate and test YOLOV3 to detect vehicles .....	40
5.2.1 Tools and frameworks:.....	40
5.2.2 Training and evaluation:.....	41
5.3 Join YOLOV3 and SORT to track vehicles.....	54
5.3.1 Tests on the detection and tracking model .....	55
6. CONCLUSION AND FUTURE LINES .....	61
Figure references .....	62
References.....	65

# 1. INTRODUCTION

The world is progressing fast enough on technology in the last years and computer science is not an area that falls behind. Computer science can be divided in many areas of study, but we could resume them in three big fields. Software development, information technology and security. We will be focusing on information technology such as artificial intelligence, machine learning and deep learning.

In computer science, Artificial Intelligence (AI) is the intelligence carried out by machines. Its goal is to build smart machines capable of performing tasks that normally are made by humans. AI is just software; it is an application written to do a task. For example, on Facebook with suggestions of names of people to tag on pictures, for fraud detection on credit cards or for self-driving cars that can manage their shelves to drive through the city.

Artificial intelligence can be divided into two categories:

1. Weak AI: Sometimes referred to as “Narrow AI”, is the type of AI that only learns to do one thing, It is going to be better than a human doing just a single thing such as predicting the temperature in your house to be warm but if you ask to detect a cat in a picture it will be terrible.
2. Strong AI: Also known as General artificial intelligence (AGI) is created to learn to do anything. It is supposed to recreate the intelligence of a human, what does this mean? By this intelligence It could be possible to have a machine doing the same things as humans like thinking or being able to take decisions. Summarizing, it is an intelligence that can do all the tasks possible of doing by a human. That is why it has not been invented yet.

Today there is weak AI everywhere, Siri, facial recognition, Instagram filters, Google adds recommendations or Amazon recommendations and more examples that are taken for granted. Siri is a software capable of recognizing the human voice and is great at doing very few things. Therefore, it is not a strong AI, it is composed by a few Weak AI. This type of intelligence is better than a human at doing a task. For example, focusing on working out if a credit card is good or a fraud, the software will be more efficient than a human just because it has processed a wide dataset of fraudulent or good cards. Nevertheless, the human will guide by its experience or knowledge.

Here is where Machine learning (ML) takes part, it is the ability of a machine to learn and predict by itself. It learns patterns from the data to make the prediction more accurate. These algorithms can make relevant conclusions from the dataset by creating a mathematical function that best fits the data. The main objective of an apprentice (learner) is to develop the ability to generalize and associate. When we translate this into a machine or computer, it means that they should be able to perform accurately. One of the machine learning fields that still has a long way to go is deep learning.

Deep learning (DL) is a subset of machine learning that is formed of layers recreating the human brain. It had a breakthrough many years ago, but it came to a standstill for some years and in the last few years has had a rebound. In terms of deep learning, the structure is called artificial neural network. DL plays with parameters, it trains a network with these parameters to learn on its own. By the way, after the network has been trained it will be able to recognize patterns in the input data to make the detection accurate. The network is formed by layers where each layer is formed by a group of neurons that carry information all through the network with the goal of giving an output. Both areas ML and DL will be further explored later, on Chapter 2.

In deep learning there is a very large and important study area that has gained a lot of strength in recent years, object detection in real time. It is mainly used in video vigilance cameras of highways or for autonomous cars cameras. It consists in the capacity of a machine to detect objects in a set of images in a very small processing time. The network's capacity of generalization must be very good because skipping the detection of an object in an image is a sign of loss of accuracy. This final degree project will be focusing on detecting vehicles (cars, trucks, motorcycles) in videos and then tracking those vehicles to be able to count them. Video processing must be done in record time to be in real-time.

## 1.1 Motivation and objectives

The motivation for this work has been the interest of learning something new that has not been taught during the career. The challenge of learning something with a level of complexity by their own. This work will be focusing on real time vehicle detection and tracking. The detection will be done with a convolutional neural network and the tracking with an algorithm of tracking that compares detections of the previous frame and detections of the current frame. The goal of this work is the development of an application that can be used in Pamplona to improve the traffic management.

The main objective of this end-of-degree work is to learn how to detect vehicles in images. Especially, cars, motorcycles and trucks, once we have learned to detect the idea is to move it on to videos and to be able to reduce the detection time until we are simulating the detections in real time. Once the detection has been implemented and developed correctly, the final objective would be to delve into vehicle tracking to make the model count how many vehicles visualizes in the video. To achieve our goal, we must first set the particular goals that will lead and guide us to the ultimate objective:

- The first step we must focus on is to learn about the deep learning techniques used to detect objects in images. The most powerful techniques are YOLO (you only look once) and mask RCNN (recurrent convolutional neural network). For this work, we will be using YOLO as it is the best deep learning technique to detect vehicles in images and is the fastest one. Fast processing is essential to accomplish real-time detection.
- For this work, it is very important to train the network and to do so we will need a wide and compact dataset of vehicles. This dataset must be composed of photos of vehicles from different angles, points of view, type of light, photo quality. It is also very important

to have photos in which no vehicle appears. These images must be accompanied by their respective tag files where they tell us the class of objects that appear and the position they have in the image.

- Once we have learnt to do the detection, we will be focusing on the tracking techniques. SORT is a tracking technique or method that will be used to count the vehicles all through the video in a precise way. Then the detection and the tracking will be joined to see how good the model is.

The use of a GPU is very important for this work. Google Collab provides a free Tesla K80 GPU that it is going to be used in each step of this project. In the next chapter ML's different type of learnings will be presented and we will see how different type of techniques work to improve predictive models. Also, we will see how a neural network works and we will go further on convolutional neural networks (CNN).

## 2. PRELIMINARIES

In this chapter, we review the necessary preliminaries to understand the rest of the work. In the Section 2.1, we will talk about different Machine Learning techniques to face valued predictions or classification problems. In the Section 2.2, we will talk about Deep Learning techniques such as neural networks and its uses for real life problems. In the Section 2.3, we will delve into convolutional neural networks explaining step-by-step their structure, how they work, how and where the predictions are made, type of layers they are built of, methods to optimize the neural network, etc. Finally, in the Section 2.4, we will talk about object-detection problems and how neural networks can face these types of problems.

### 2.1 Introducing Machine Learning

Machine learning is a subset inside artificial intelligence, and it serves to create systems that can learn by their own. It is a technology that makes it possible to automate a series of operations in order to reduce the need of human intervention. This is a great advantage because it controls a huge amount of information in a much more effective way than humans do. What is called learning is the ability of the system to identify many complex patterns determined by a large number of parameters. Machine learning techniques predict an output from a given input by applying the patterns they have learnt in the training before. Machine learning techniques can be divided into three different type of learnings, **supervised, unsupervised and reinforcement learning**.

**According to supervised learning**, there are two type of methods, regression and classification. Regression based algorithms estimate continuous values; the output is a real value. The objective is to create a model that represents the dataset. The model will be used to predict the output for new values or examples. In practice, the objective of this type of learning is to create a mathematical function that from an input it obtains an output. Linear regression is maybe the best-known method to predict the behaviour of the data to analyse and logistic regression is a variant that does the same but integrating categorical variables. Nevertheless, classification

methods estimate categorical values, the output is a class value. This type of algorithms are categorical methods that receiving an input of data can classify into a category. These classes can be qualitative or quantitative values.

For example, a car shop wants to predict the price of a second-hand car depending on two variables: Km done and antiquity. The variables will be represented as  $X = \{\text{Km done, antiquity}\}$ . The model will be trained until it reaches a convergence point, in that point the values of the parameters will be optimal. Those parameters are essential for the model function because now giving an input (second-hand car km done and antiquity in years) to the function it will return the predicted output that will be the price of the car. This is a very simple but representative example of a linear regression application.

**In unsupervised learning** the data has no associated labels, but the goal is to organise the data into clusters or groups and find the most important characteristics inside a dataset. In Cluster analysis the data is grouped according to similarities or distances between them. Usually in this type of algorithms data is classified into a group, this groups examples share between that they are similar (distance between them is very small) and they belong to the same cluster. Examples in the same cluster are close but far from different clusters examples. Unsupervised learning can also be used to remove unnecessary or duplicated variables from the dataset to create a smaller subset of the original data. This technique is called Dimension Reduction, it evaluates each variables importance inside the dataset, it assigns a value from 0 to 1 depending on its importance and orders them in a list. After this process, the variables are ordered by their importance in a list and depending on the problem, the variables needed are chosen to be part of the new subset.

**In reinforcement learning** the process is very easy, the machine learns from its experience. There is no initial set of data to learn from, either because they do not exist or because it is difficult to collect them. Its way of learning is a mechanism of trial and error, if the model fails it learns from that failure but in the opposite case it learns from the trial. Imagine a model capable of learning patterns without analysing a dataset? That is what reinforcement learning is searching for.

How can we know that our predictive model generalizes well for new examples? By applying cross validation and holdout techniques. With these techniques, it is possible to balance the accuracy on the training and test/validation set where the model's capacity of generalization is increased and can-do better predictions. The most used technique is the k-fold cross validation that consists of dividing the dataset in k subsets and iterating k times, train the model with k-1 subsets of training and validating it with the kth subset. At the end, the average of the accuracies of all the trainings is obtained.

It should be pointed out that, Machine learning applications are essential in areas that manage a big amount of data and their objective is to extract conclusions and learnings from the data. We could say that Machine learning shaves lives in hospitals because it can detect cancer much faster than a doctor looking at an X-Ray or it can be an essential tool in the fraud detection on bank transactions. Some other applications are, the spam detector of the email, the marketing

suggestions on amazon and the autonomous car. In the next point we will be focusing on deep learning techniques and their impact on detecting objects.

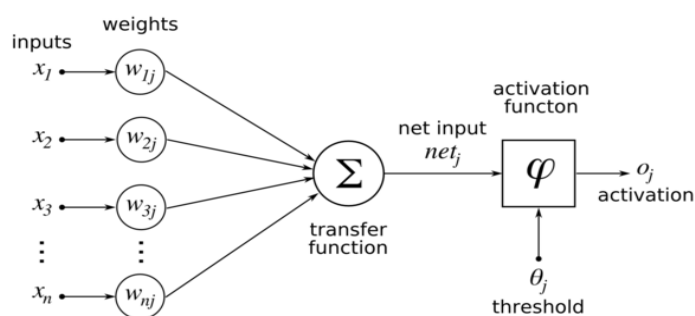
## 2.2 Introducing Deep Learning techniques

Deep neural networks also known as hierarchical learning are made up of interconnected units like neurons that getting external inputs can carry information of each neuron all through the process to predict an output. DL is characterized by the use of networks that learn and store information in several layers. In this way, the neural network will be able to learn complex patterns in big amounts of data. In the image processing field, supervised learning is the most used method where the network is trained by using images along with their corresponding labelled files.

Concerning to the amount of data, neural networks performance is higher than older learning algorithms when the amount of data increases. This means that with a big dataset, deep learning techniques work better than older algorithms like linear regression or Naïve Bayes classification. The main advantage of DL over ML methods is that DL methods can extract characteristics and patterns from input data in order to make an effective prediction.

To know how a simple neural network works, we must first know how a neuron works. A neuron is the basic processing unit that we are going to find inside a neural network, like a biological neuron. Each neuron has input values by which it receives external information, as it can be seen in the arrows entering the red neurons in Figure 1. With these input values the neuron will perform internal calculations and give an output value. In simple words, a neuron is like a mathematical function that from a few values gives us an output value. But what are the internal calculations that the neurons do to give the output? It will be explained by an example of a neuron with its parameters: (each parameter can be seen in Figure 1 graphically what it represents inside the neuron)

- $x_i$  is the information of the  $i$ -th neuron of the previous layer.
- $w_i$  is the weight of the  $i$ -th input value and the neuron.
- $\theta$  is the threshold value.
- $\Sigma$  is the weighted sum.  $\Sigma = w_1 x_1 + w_2 x_2 + \dots + w_n x_n - \theta$
- $\varphi$  is the activation function, where  $\varphi(\Sigma) = \text{output of the neuron}$ . The most used activation functions are the Sigmoid, Tanh and ReLU.

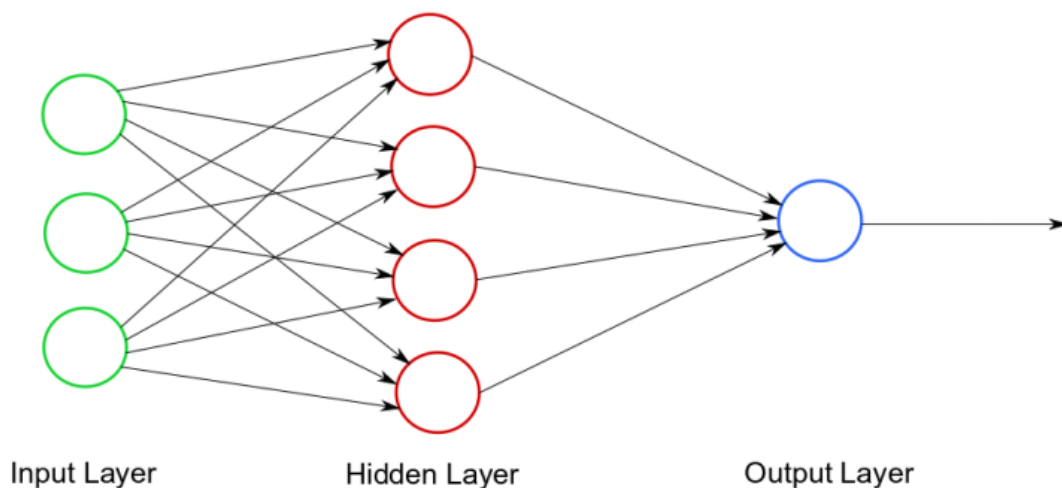


**Figure 1:** Functionality of a single neuron.



By grouping many neurons, we can build layers and by combining layers in a complex way we can build a neural network of any kind. A simple example of a neural network is shown in Figure 2 where each neuron is interconnected to numerous other neurons in which they share information. There are three main layers, the input layer that takes care of learning the easiest features and then this information is passed to the hidden layer. The hidden layer can be made up by one or a million layers and in this case, it receives that information somewhat simpler and composes it to more complex characteristics. The last layer is called the output layer, which transforms the information passed by the hidden layer to give an output.

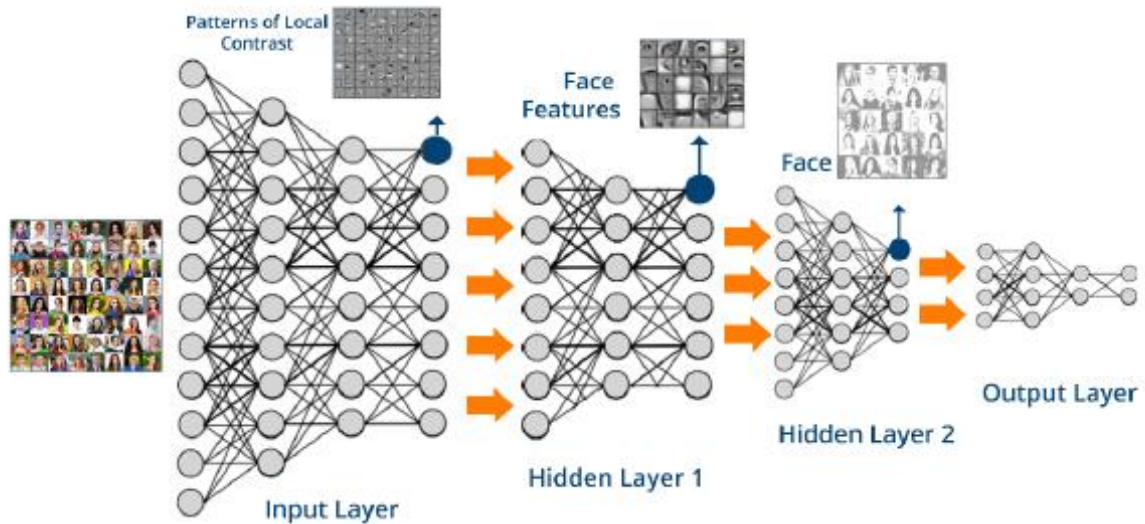
In a neural network we have two basic actions, forward propagation and backward propagation. The forward propagation consists on giving an example to the network and it is spread all through the network to give an output. Nevertheless, the backward propagation is made after the forward propagation and it propagates the error from the end to the start of the network using the derivative of the cost function. Both actions are essential in the training process, with the backpropagation we update the weights of each layer of the network and with each improvement the network is supposed to be able to predict better than in the previous step.



*Figure 2: Example of the architecture of a normal neural network with its three main layers.*

This is a very simple representation of a neural network, but nowadays to make better models we need to develop multi-layer networks with multiple neurons in each layer. As an example of these types of architectures, we will focus on a network used for face recognition that is shown in Figure 3. In the input layer, low-level or simple features such as lines, curves and edges with different orientations are learned. The input layer is composed of a simple neural network that has its input values, its hidden layers that recognize not very complex patterns and the output layer with a set of neurons that store the low-level features information. Moreover, the information that has been learnt through the input layer of the face recognition network (the output) is going to be the input value for the hidden layer 1. The hidden layer 1 is also made up of a simple neural network, mid-level face features (nose, eyes, mouth, etc) are learned from combining and processing the already learned information in the input layer. In the hidden layer

2, high-level abstraction features are learned from mid-level features like how a human face looks like. Finally, in the output layer, a result is given depending on the type of the problem: recognize a specific person or match the race of the person according to his face.

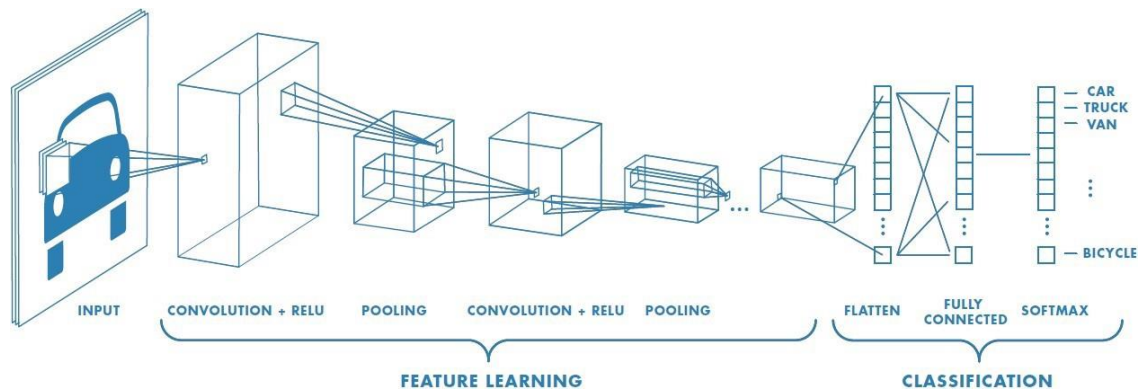


[Figure 3](#): Example of a face recognition neural network with two hidden layers.

### 2.3 Convolutional Neural Networks (CNN)

Convolutional neural networks are very powerful for everything that has to do with image analysis, because they can detect simple features like for example edges, lines, etc and compose in more complex characteristics until they detect what they are looking for. CNN techniques go another step further than the neural networks we have previously developed. Its level of complexity is much higher as the level of abstraction increases in each layer. CNN techniques nowadays are also used for text processing such as internet searches, but its main use is on image processing in both photos and videos.

The **CNN architecture** is composed of three basic operations: *convolution*, *ReLU* (Rectified Linear Unit) and *pooling*. These three operations are applied in all the layers and are complemented by a fully connected multi-layer perceptron at the end of the network, this last layer of the CNN network handles the classification. As it can be seen in Figure 4, in a CNN the feature learning is done in the layers where the convolution, ReLU and pooling operations are applied. In the first layers low-complexity patterns are learned that they are going to be very useful on future layers to learn more complex patterns until the last layer of the future learning process is reached. Once this layer is reached, the network has learned how the object looks like and the perceptron (output layer) will make the prediction.

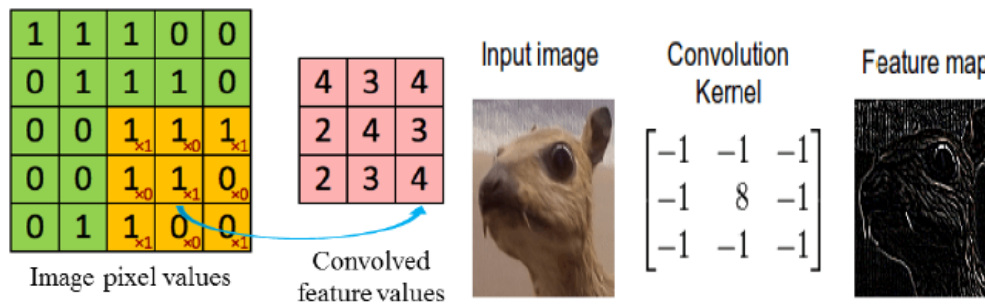


**Figure 4:** Graphical example of a CNN with its three main operations (Convolution, pooling, ReLU) with the final perceptron classifier.

The *convolution* is a linear operation that is like the application of image processing filters that extract features in the different layers of the network. These filters are matrices of dimension for example 3x3 or 5x5 and are called kernels. This technique is implemented by sliding the filter all through the image, generally with the shift of a pixel. It starts on the upper left corner and it scrolls from left to right until it reaches the lower right corner, and the computation is made by the sum of the element by element multiplication between the weights of the filter (e.g. 3x3 matrix) and the image (the pixel that is being analysed and the 3x3-1 most nearby pixels window). This operation is done for each pixel of the image and the output is called the feature map or the activation map. The features learned in these layers are normally lines, curves, corners, edges, etc. Figure 5 shows an example of a convolution operation over a 5x5 image. In addition, in Figure 5 we can appreciate how with the kernel the animal silhouette can be learned and used on future layers with higher complexity

*ReLU* is a non-linear pixel-by-pixel operation that replaces negative pixels found in the feature map with zeros. Its objective is to introduce non-linearity in the data because most of the worlds data is nonlinear. *ReLU* is the activation function used after the convolution is done to an image and is the one usually best performs, but another activation functions can be used such as sigmoid or tanh. Figure 6 shows an example of the *ReLU* operation.

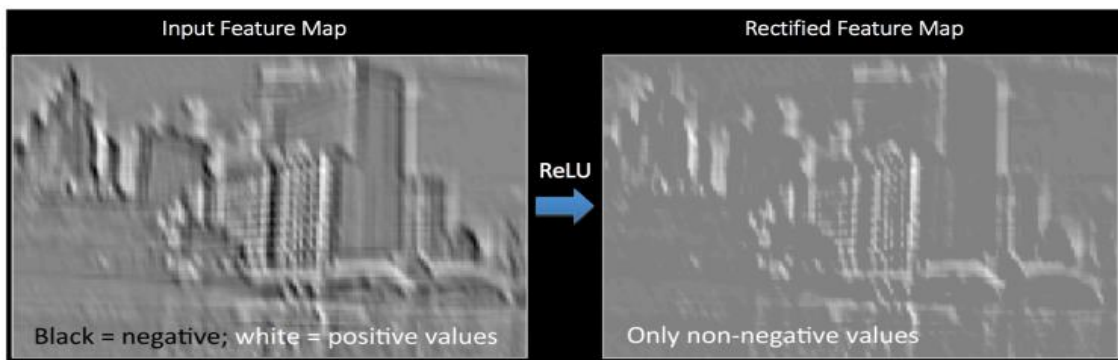
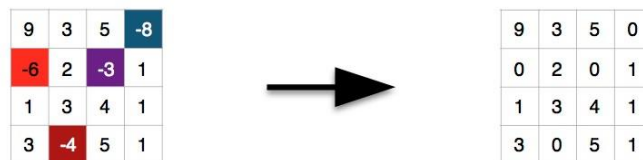
*Pooling* decreases the number of parameters by remaining with the most common features obtained in the *convolution* and *ReLU* steps. The goal of this operation is to make features to scale by making a reduction of the dimensionality of the feature map. The way to reduce parameters is done by using different mathematical operators such as the average, maximum or sum of a fixed region of the feature map. In practice, the operator that best works is the maximum operator and is the most popular one. In Figure 7 it is shown an example of the maximum and average pooling operator.



**Figure 5:** Convolution operation in an image of dimension 5x5 with a kernel of dimension 3x3. Application of a linear convolution kernel to an animal and the output.

The animal image of Figure 5 is a multiple channel image because it is a coloured image (e.g. RGB). This image is represented in a three-dimensional matrix where each dimension is the intensity of a particular colour (red, blue, green). In the case of this type of images, the kernel has the same depth of the original matrix. When a convolution is applied to an image, there are two important concepts to understand: **stride** and **padding**.

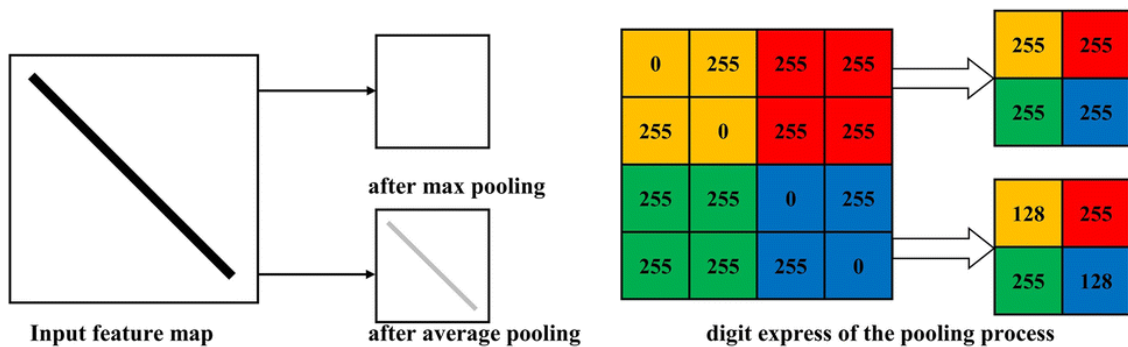
**Filter 1 Feature Map**



**Figure 6:** Example of a ReLU operation replacing negative pixels by zeros. Graphically it is seen the change of negative pixels by zeros and how the picture changes.

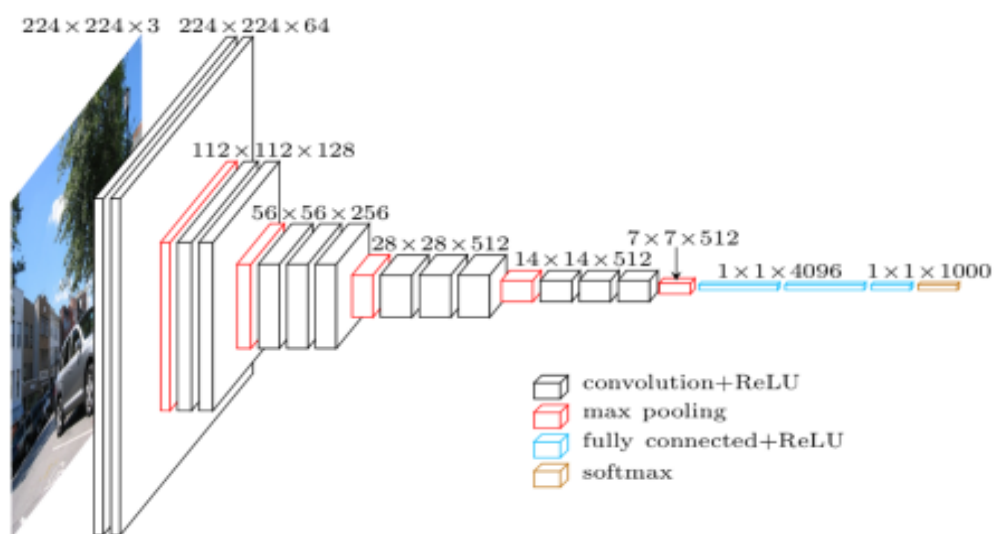
**Stride** is the pixel offset we do every time we move to the right with the kernel. If the stride = 2, this means that if we start with the first pixel in the top left corner of the matrix, after applying the filter to that pixel we must move two pixels to the right and apply the filter again. This will be done successively until the end is reached. There are two different types of padding; same padding, where we augment the number of rows and columns of the original image in order to get the same dimension in the feature map as the original image; and the valid padding, which performs the same operation without adding rows and columns and gets a feature map with

same dimension of the kernel. These two techniques are used in layers where low-level patterns are learned.



**Figure 7:** Example of the pooling operator. The original image is reduced by two in each axis after pooling it. Max pooling and average pooling are shown graphically.

Once we have learned features with the convolutional and pooling layers it is time to classify the object is being analysed. With all the layers above, the model has learned patterns enough that it is able to make a classification. Here is where the output layer takes part, the fully connected multilayer is a traditional Multi-Layer Perceptron that takes as input value the reduced image, converts it to a column and passes through the layer to get an output. This output layer uses the SoftMax activation function which gives a probability between 0 and 1 to each possible output class (the sum of all possible class probability must be 1). For example, if we want to detect a car in an image and we use a CNN trained to detect cars, trucks and motorcycles, the output layer will give to each class a probability value depending on what it is detecting on the input image and the highest probability will be the class predicted. An example of a CNN with all the operations and layers shown in this Chapter are shown in Figure 8, including the classification layer.



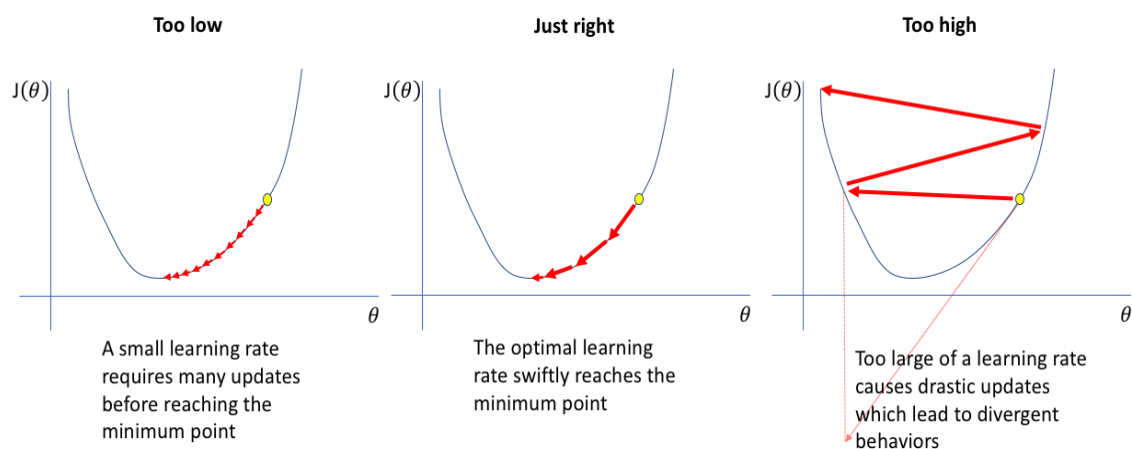
**Figure 8:** Example of a CNN with the convolution + ReLU layers (black cubes), pooling layers (red cubes) and the classification final layer (blue cubes) showed graphically.

Regarding the optimization of a CNN, the objective of optimizing a CNN is to adjust the hyperparameters to a specific dataset in order to minimize the cost function. Normally, feature maps are learned by the back-propagation technique with the Gradient Descent (GD) that adjusts the weights and filters used in the different layers and this leads to a minimization of the total error. Some optimization algorithms are Stochastic Gradient Descent (SGD), Adam, Adaptive Gradient (AdaGrad), etc. but the most stable algorithm is the SGD with momentum. The objective of this algorithm is to modify the parameters in order to reach a convergence point.

This algorithm is based on two hyperparameters, the learning rate and momentum that are included in the weights update function. The learning rate ( $\alpha$ ) of an optimization algorithm is how fast can the algorithm learn from the data in the training process. It controls the weight updating function with its hyperparameter in order to minimize the total error. As it has been mentioned above, the objective is to converge and with the learning rate hyperparameter the convergence point can be reached quickly or slowly. If the learning rate has been set too low at the start of the training, it will converge slowly to a point and the training process will be very expensive as it will need many iterations to get the global minimum. Also, depending on what point the algorithm starts iterating on the cost function, we may not get any convergence point.

Alternatively, if the learning rate has been set too high, the algorithm will update the weights too quick and therefore it may happen that the local minimum is skipped and therefore it will enter a loop that will diverge instead of converging. In order to deal with both problems, it is advisable to establish an intermediate learning rate called step decay to ensure that our algorithm is computationally fast and able to converge in the training. In Figure 9 is shown how different type of learning rates behave with respect to the cost function.

The momentum ( $\mu$ ) is a constant that is applied in the weight updating function that will be added to the new weight. The momentum hyperparameter is used to avoid getting stuck in a local minimum of the cost function without reaching the global minimum. The objective is to get the lowest value for the cost, and with the momentum we will avoid getting stuck in a very low value and let us reach the global minimum.



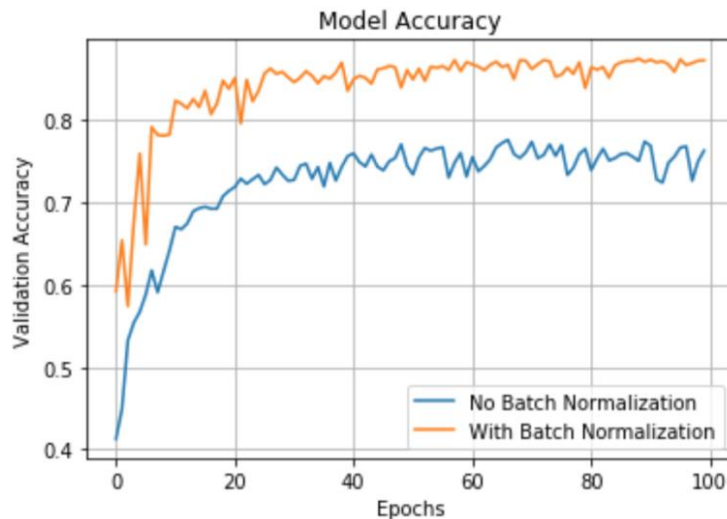
**Figure 9:** Example of the cost function behaviour with respect to different learning rates. When it is too low the training will be very slow, if it is too high the training will diverge.

Continuing with the theme of optimizing a CNN within the SGD, having mentioned above the two previous hyperparameters, there are two other parameters that are not as important as learning rate and momentum in the training but should be mentioned: batch size and number of epochs. Both are integer values and are very easy to confuse and are closely related. An epoch (cycle) is the number of times forward propagation and backpropagation algorithms are executed in which all training data passes through the neural network. The batch size is the number of data each iteration of an epoch has, so the batch is like a small part of an epoch. For the computer is very difficult to process a whole epoch at once because maybe we cannot store all the dataset at once, so it is divided in batches and each batch is processed in each iteration until one epoch is reached. For example, if we have 200 images in our dataset and we want to do 10 epochs we will process the 200 images 10 times, it is very important to pass the images more than one time to the network in order to improve the learning. If we set a batch size of 100, we will have two iterations for each epoch because  $100 * 2 = 200$  that is the whole dataset.

To improve the optimization of a CNN there are other methods such as dropout or batch normalization that are different, but they lean at each other. Dropout is a method that randomly disables several neurons from a neural network. At each iteration of the neural network, dropout will disable different neurons, the deactivated neurons are not considered for forward propagation or backward propagation which forces nearby neurons not to depend so much on the deactivated neurons. This method helps to prevent overfitting because nearby neurons learn patterns that may be influenced by their neighbours, with dropout this dependency between neurons is lower because they learn to work on their own. It is possible to establish a different dropout in each layer of the network, depending on what type of patterns each layer learns.

Batch normalization focuses on the set of data defined by the user's batch size and normalizes each data batch. It is very necessary to normalize data to avoid big differences between them such as in a colour image that can have values from 0 to 255. When standardizing is done, data distances go from 0 to 1 and this helps to neural networks work better and have less problems. But when the standardization is only done in the input layer, only the input benefits from this and therefore when the data passes through other hidden layers this normalization is lost. The batch normalization method normalizes the data before they go through the activation function in each layer of the network, in this way the data will be always normalized through the training process. This method normalizes the output of a layer by subtracting the batch mean and dividing it by the batch standard deviation. It also resolves the problem of covariate shift, for example if we have a dataset of white and black cars and we divide the dataset into batches it is very important to have in each batch images of black and white cars. With the normalization of the data the examples will be closer between them and it will not affect that much to the training process.

Figure 10 shows an example where the accuracy is better using batch normalization than without using it, the dataset is processed in 100 epochs and it can be seen that as the number of epochs increase the accuracy improves. Therefore, CNN use batch normalization specially after a convolution, ReLU or fully connected layer.



*Figure 10: Accuracy on validation set of a dataset trained using batch normalization and without using it with respect to the number of epochs.*

If the dataset is small data augmentation layers can be used to increase the volume of the training set. This technique applies transformations to the images such as translations, rotations, reflections or deformations. Specially, they are used when the training needs a bigger amount of data and it is not possible to obtain more images or maybe manual annotations are not available, this technique is essential for small datasets because it reduces overfitting. The problem of the shortage of images for improving the model will not be a problem on this work.

CNN are the most popular techniques inside DL when image processing is needed, this work is based on image and video analysis and processing. We will focus on improving a CNN with a dataset big enough to do a good training and then we will test the weights obtained to see how good the detection is.

## 2.4 Object-detection

According all the techniques mentioned so far, ML algorithms or models could not face the object-detection problem because it is impossible to make a mathematical function that can predict an object in an image. For example, if we want to develop a model that detects apples in an image, a regression or a classification model has the added cost of thinking about an appropriate characteristics extraction model and therefore we should focus on deep learning technics. Regarding DL, neural networks are a powerful tool that can process a big amount of data increasingly fast ensuring to develop good predictive models.

CNN is the best tool regarding object-detection, nowadays it is the best way for developing models that can make face-recognition or person detection on video vigilance applications. An object has a specific position in the image, it is in a particular height and width and the model



should be able of detecting the object and surround it with a rectangle as it can be seen in Figure 11. This rectangle is called a bounding box (BB) and the model must know how to return this box automatically every time it has detected an object. To obtain a model that does so, some essential steps must be taken.

First, a dataset of images with the objects we want to detect with their labelled files must be obtained, the CNN in the training will learn to detect the object and ubicate it in the image. The labelled files are plain text files containing for each row the class of the object that appears and the coordinates of the bounding box that surrounds the object. If the objects are very strange and we are not able to find images with their labelled files, we will have to note them manually because is essential for the network to know where the object is in the image.



*Figure 11: Example of a model detecting objects and surrounding them with a bounding box with the percentage of assurance of the class detected.*

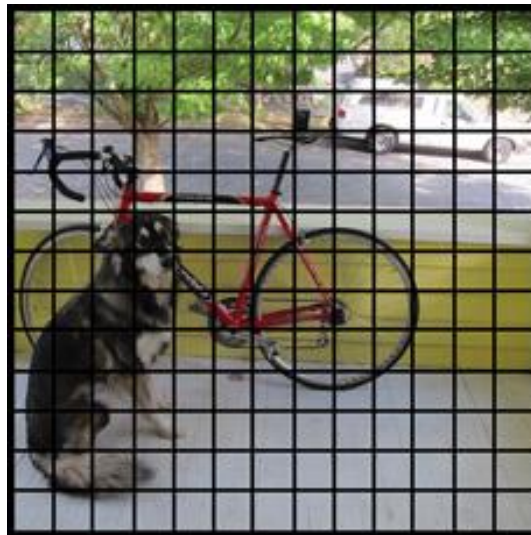
Once the dataset has been obtained, we will train the network and test it with the test set to see if the model is good at detecting the objects we want. Regarding this work, our objective is to develop a model that predicts cars, trucks and motorcycles in images and videos. We will have to find a wide dataset with images containing these objects together with their labelled files containing the position inside the image (bounding box) and the objects class because we want to detect more than one object. Apart from this, the dataset must contain images not containing cars, trucks and motorcycles because it can happen in a video frame that does not appear any object and the model must be capable of not detecting anything.

For the training we have to divide the dataset into three subsets, train, validation and test set. The model will be trained with the train set and the weights will be updated during the training process depending on the cost function, the model will be evaluated with the validation set. After the training and the evaluation of the model, we will use the test set to see if we have a good generalization model, if it has learned how a car, truck and motorcycle looks like and it is able to locate the objects correctly in new images. In Chapter 3, YoloV3 CNN model will be explained and reasoned why it has been selected for this work.

## 3. OBJECT-DETECTION WITH YOLO

### 3.1 What is YOLO?

YOLO (You Only Look Once) is a state-of-art algorithm devoted to object detection, as the name implies it can predict objects just by looking once to the image in a clever way. YOLO makes the prediction by classifying the object and locating it in the image. It uses deep learning and CNN techniques to detect objects, and distinguishes itself from its competitors because, as its name indicates, it requires to see the image only once, allowing it to be the fastest of all although it sacrifices a little accuracy. This speed allows users to easily detect objects in real time in videos (up to 30 FPS). In other words, YOLO takes as input value an image and passes through the neural network that looks like a CNN and returns a vector of bounding boxes and class prediction. To understand better how YOLO makes the prediction we will explain it with an example of an image as it can be seen in Figure 12.



*Figure 12: Input image divided in 13x13 grid cells.*

To perform the detection YOLO first divides the image into an  $S \times S$  grid of cells, in Figure 12 we can see that the image is divided into a  $13 \times 13$  grid of cells. In each of the cells it predicts  $N$  possible bounding boxes and calculates the level of certainty (probability) of each of them (shown in Figure 13), that is,  $S \times S \times N$  different bounding boxes are calculated in the image, the vast majority of them with a very low level of certainty. The score of each BB does not classify what type of object is or it does not know what object is fitting the bounding box, it just gives a confidence value or a probability value of how good the BB is surrounding an object in each cell. In case of Figure 12 for each grid cells 5 bounding boxes are predicted and are graphically shown in Figure 13 where the higher the probability is the fatter the box is. Each cell makes  $N$  bounding box predictions and  $M$  class probabilities.



*Figure 13: Example of the bounding boxes predicted in each cell; the high confidence boxes are drawn fatter.*

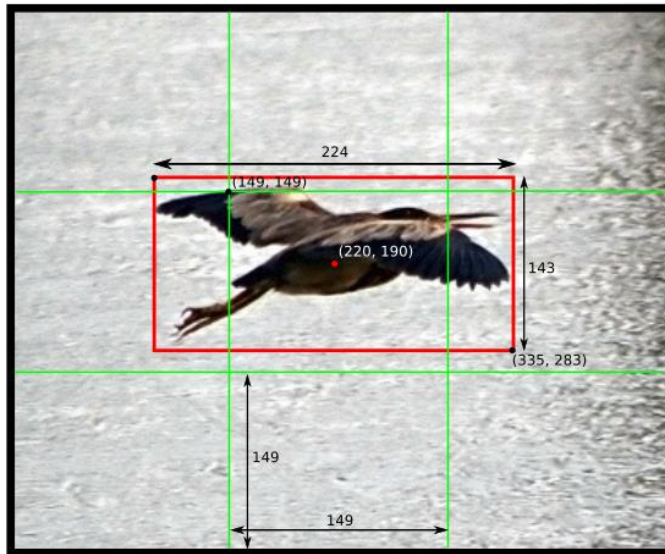
The bounding box prediction is formed by 5 components:  $x$ ,  $y$ , width, height, confidence score.

- The  $(x, y)$  components are like a mathematical graph coordinates that refer to the centre of the box, relative to the grid cell location. If the centre of the bounding box does not fall inside the grid cell this means that the grid cell is not responsible of the object that has predicted the bounding box. This happens because each object that appears in the image is related to a single grid cell that is responsible for predicting the object.  $(x, y)$  are normalized between 0 and 1.
- (Width, height) represent the dimension of the box that contains an object, they are also normalized to values from 0 to 1 and they are fundamental to locate the object in the image.
- Confidence score is a real value that represents the assurance the algorithm has that the box contains an object of any class. The method how the confidence is calculated will be explained later.

Maybe it is tricky to understand how the normalization of  $(x, y, \text{width}, \text{height})$  it is done. In Figure 14 we can see how this process is done. First, the image contains a bird flying above something that seems to be a river. YOLO divides the image into a  $3 \times 3$  grid cells and predicts for each cell the  $N$  bounding boxes with its probability of containing an object. In our image, we only focus on the bounding box that frames the bird, how we get to this result it will be explained in the following paragraphs but now it is not necessary to know.

If we enumerate each cell from top to bottom the cell in the middle has to be number 5, so the bounding box containing the bird has its centre inside this cell and as it has been mentioned above, it is responsible of predicting the object and the prediction has been done correctly. Looking at the normalization, each grid cell has a width and height of 149 and these measures will be used for normalizing the two first components of the bounding box.  $X$  will be subtracted and divided by the width of the fifth grid cell and  $y$  the same as  $x$  but changing the width by the height of the fifth grid cell, nevertheless, the width of the object will be divided by the width of the image and the height same.

(0, 0)



(447, 447)

$$\begin{aligned}x &= (220-149) / 149 = 0.48 \\y &= (190-149) / 149 = 0.28 \\w &= 224 / 448 = 0.50 \\h &= 143 / 448 = 0.32\end{aligned}$$

**Figure 14:** Example of a 448 x 448 image divided in grid cells of  $S = 3$  where the birds bounding box components have been normalized.

Going back to Figure 13, we have  $N$  bounding boxes with their 5 components and each bounding box has a probability of containing one object of each class. There are in total  $S \times S \times N \times 5$  outputs regarding the bounding box predictions, for example, in Figure 13 the input image has been divided into a  $13 \times 13$  grid cells and in each cell 5 bounding boxes are predicted, so  $13 \times 13 \times 5 = 845$  bounding boxes will have the YOLO algorithm to process where  $845 \times 5 = 4225$  components.

In addition, as it has been mentioned before, the grid cells also calculate  $M$  (number of classes) class probabilities. The conditional class probability “ $P(\text{Class}(l) \mid \text{Object})$ ” is a probability that reflects the confidence that an object contained by a bounding box belongs to a specified class. For example, if we are working in a problem of 3 classes YOLO will give for each cell three different probabilities, probability that the object detected belongs to class 1, probability that the object detected belongs to class 2 and probability that the object detected belongs to class 3. YOLO’s prediction can be expressed as a vector regarding our example of *Figure 12 and 13* ( $S, N \times 5 + M$ ) = (13, 13, 5 x 5 + 3) = (13, 13, 28).

Finally, for each bounding box the **class confidence score** is computed with the following formula (also shown in Figure 15): **class confidence score** = *box confidence score* x *conditional class probability of the cell the BB belongs to*. With this value we obtain the probability that an object contained by a bounding box is well located inside the BB and what is its probability to belong to one of the problems classes. In this step there are still all the bounding boxes, the ones containing objects and the ones that do not contain any object because they do not affect to the loss function.

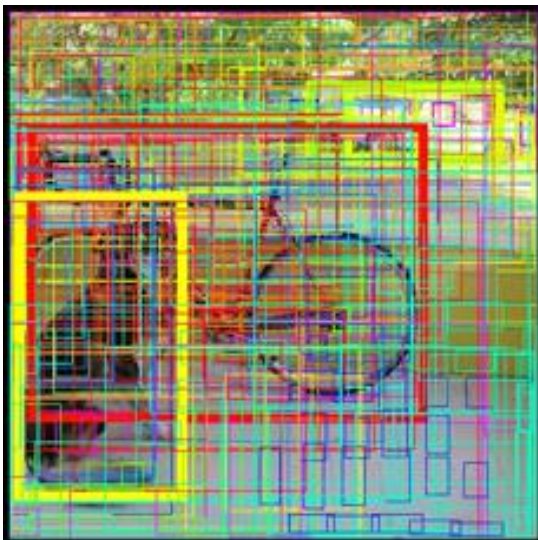
$$\begin{aligned}
\text{box confidence score} &\equiv P_r(\text{object}) \cdot \text{IoU} \\
\text{conditional class probability} &\equiv P_r(\text{class}_i | \text{object}) \\
\text{class confidence score} &\equiv P_r(\text{class}_i) \cdot \text{IoU} \\
&= \text{box confidence score} \times \text{conditional class probability}
\end{aligned}$$

where

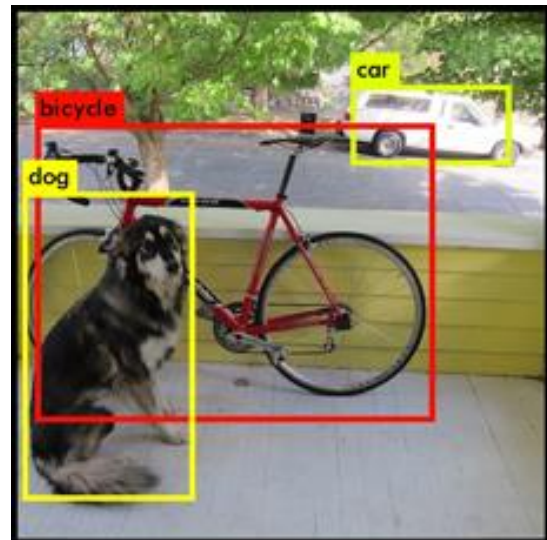
$P_r(\text{object})$  is the probability the box contains an object.  
 $\text{IoU}$  is the IoU (intersection over union) between the predicted box and the ground truth.  
 $P_r(\text{class}_i | \text{object})$  is the probability the object belongs to  $\text{class}_i$  given an object is presence.  
 $P_r(\text{class}_i)$  is the probability the object belongs to  $\text{class}_i$

**Figure 15:** Explanation of the different components for the computation of the class confidence score.

Once the class confidence score has been computed YOLO obtains an image like in Figure 16, the bounding boxes have a percentage of certainty they are containing a class. For example, in Figure 16 we can see that the red BB is 80% sure that contains a bicycle and that is why the red bounding box is fatter than the rest red BBs. When we have computed the number of bounding boxes through the image, we have obtained 845 bounding boxes in total, but it is time to take out most of the boxes. The user will set a confidence interval and those boxes that have lower confidence values will be deleted ending up with the final BB that contain the objects. The remaining boxes are put on a non-max suppression which removes possible objects that were detected in duplicate and thus leaving only the most accurate one. As we can see in Figure 17 from the 845 bounding boxes we had in previous steps, only three have been kept because they are the ones that best represent the detection of the dog, bicycle and car.



**Figure 16:** Computation of class confidence.



**Figure 17:** Final result of the detection.

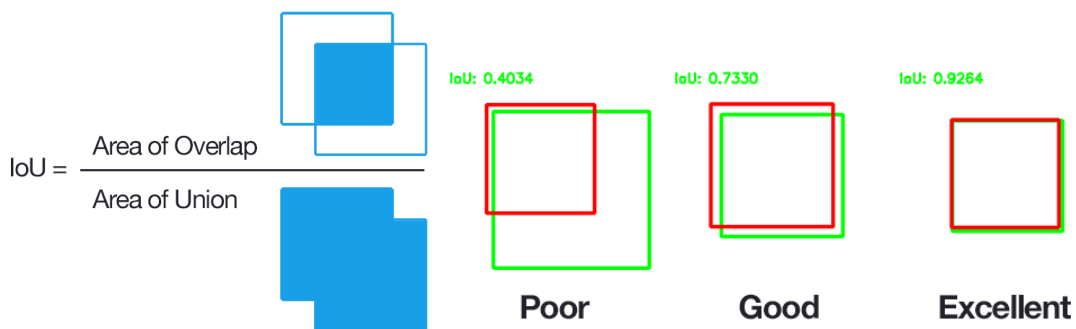
Figure 16 shows that the box confidence score is computed by multiplying P (object) by Intersection over Union (IoU). What does Intersection over union concept mean for object detection? As it has been mentioned above it will be briefly explained now.

It is an evaluation metric used to evaluate the performance of the CNN detectors such as R-CNN, Faster R-CNN, YOLO, etc. This evaluation metric is used in the training process and can be used to evaluate any algorithm that returns bounding boxes as output. In case of YOLO, it is an algorithm that returns the most “rated” bounding box and the class of the object contained in the input image. To apply this metric is necessary to have the following characteristics:

- Ground-truth bounding boxes, this are the labelled bounding boxes of the training and validating set.
- Predicted bounding boxes from the model.

During the training process, the IoU metric is applied by dividing the area of overlap between the predicted bounding box and the ground-truth bounding box and the area of union that is the area formed by the predicted bounding box plus the ground-truth bounding box (Figure 18 shows graphically how are the areas of the numerator and denominator). In the training, the class confidence score is computed multiplying P (class(i)) by the Intersection over union of the two BB. However, if we are predicting a new image, this image has not an associated labelled file and so the formula of the class confidence will be the second one in Figure 15.

*Figure 19: Example of different IoU ratios depending on different BB.*

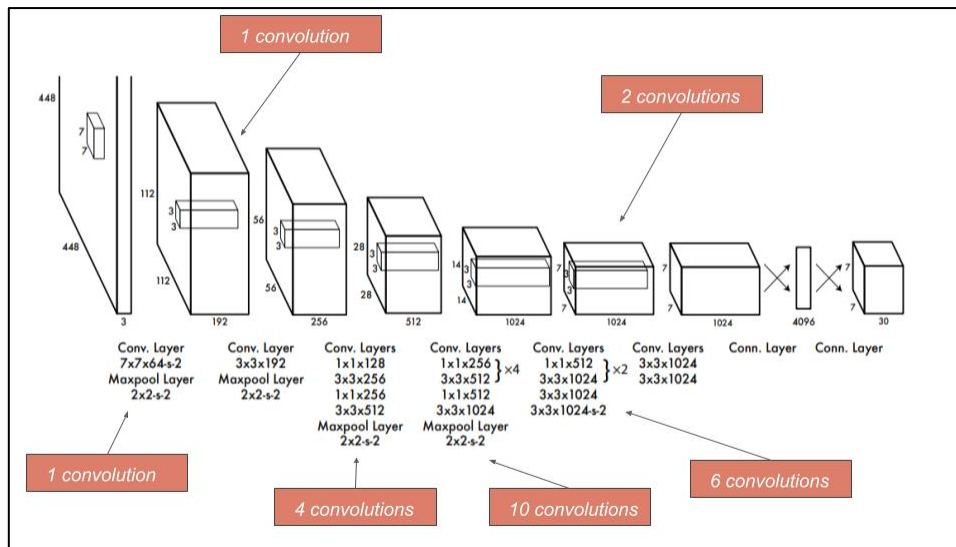


*Figure 18: Intersection over Union is calculated dividing the overlap area between both BB by the union of both BB.*

### 3.1.1 YOLO architecture

According to YOLO architecture, it looks like a normal CNN with convolutional and max pooling layers combined with two fully connected layers in the output layer. In Figure 20 we can see the full network with the filters applied in each layer. YOLO was created to use in the Pascal VOC dataset with the following values: S (number of grid cells per line or column) = 7, N (number of bounding boxes per cell) = 2 and M (number of classes of the dataset Pascal VOC). According to these values, each input image will be divided into  $7 \times 7 = 49$  grid cells, each grid cell will create two bounding boxes so there will be  $49 \times 2 = 98$  BB multiplied by 5 because of the BB components giving a total of 490 components. Moreover, each cell will calculate 20 class probabilities so the

final output of components for YOLO to process will be  $490 + 49 \times 20 = 1470$ . The size of the output for this network will be  $(S, S, (N \times 5 + M))$ .



**Figure 20:** Architecture of YOLO formed by 24 convolution layers along with 4 max pooling layers, followed by 2 fully connected layers at the end.

This architecture is very influenced by the GoogleLeNet network, the feature extraction is done in the 24 convolutional layers and the bounding box predictions and their respective object probabilities in the two fully connected layers at the end. Some convolutional layers use 1 x 1 reduction layers to reduce the depth dimension of the feature maps that were inspired by the GoogleLeNet model. All the layers use as activation function a leaky ReLU ( $\Phi(x) = x, \text{ if } x > 0; 0.1x \text{ otherwise}$ ) except the last layer that uses a linear activation function. YOLO is very easy to use, it takes an input image and passes through all the convolution layers until it passes the last one, this layer outputs a tensor of shape (7,7,1024). Then with two fully connected layers the tensor is flattened to give an output of 1470 parameters as it has been mentioned before, the last step is to delete the bounding boxes that have a confidence lower than the threshold. It is worth mentioning that there is a faster (155 fps) YOLO algorithm but less accurate (52.5 mAP) formed by 9 convolutional layers and 6 pooling layers called *Tinny YOLO*.

### 3.1.2 Loss function and training

YOLO predicts a subset of bounding boxes in each grid cell of the input image, most of this BB do not contain any object and the loss function must be build up so that these boxes interfere as little as possible in the approximation towards the global minimum. YOLO loss function is quite difficult to understand, in this part it will be explained step by step how we get to the final function. The function loss is composed of the **localization loss**, **confidence loss** and **classification loss**.

**Localization loss** function is shown in Figure 21 and 22 and computes the loss related to the predicted bounding box position  $(x, y)$  with respect to the real position  $(\hat{x}, \hat{y})$ . As we can see there

are two summations, the first summation goes through each grid cell of the input image and with the second summation we go through each BB of the cell we are analysing. In short, we do the sum of the distances between the predicted BB and the ground-truth BB of each grid cell. “ $\mathbb{1}_{obj}$ ” refers to only counting the box responsible for detecting the object:

- If  $\mathbb{1}_{obj} = 1$ , this means that an object is in the grid cell number  $i$  and the bounding box predictor number  $j$  is responsible of predicting the object.
- If  $\mathbb{1}_{obj} = 0$ , the bounding box number  $j$  is not responsible for that object or the cell number  $i$  does not contain any object.

This function will only sum the BB responsible of predicting an object inside cells that contain objects. The BB responsible of predicting an object is defined from the result obtained with the Intersection over Union metric. If we have a cell that contains an object, we will apply the IoU metric to all the bounding box predictors and the BB that has the highest ratio will be the responsible for predicting the object (i.e. Figure 19 show metrics for three bounding boxes, in that case the third BB will be chosen because it has the highest ratio).

Figure 22 shows the function related to compute the localization loss between the predicted box ( $w, h$ ) and the real width and height annotated in the training labelled file. The function looks like the one in Figure 21 apart from the square root applied to the width and height of the predicted and real values. “This happens because the loss function should reflect that small deviations in large boxes matter less than in small boxes. To partially address this, we predict the square root of the bounding box width and height instead of the width and height directly”. [9]

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$$

[Figure 21](#): Localization loss function related to  $(x, y)$ .

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2$$

[Figure 22](#): Localization loss function related to  $(width, height)$ .

**Confidence loss** is the loss associated to the confidence score calculated for each of the BB predictors. In previous pages, we have referred to the parameter  $C$  as  $M$  and it has been explained what it means. In Figure 23,  $C$  is the confidence score of a grid cell and  $\hat{C}$  is the Intersection over union between the predicted bounding box and the ground-truth bounding box. The value of  $\mathbb{1}_{obj}$  behaves as in the previous formulas and  $\mathbb{1}_{noobj}$  is the opposite (if  $\mathbb{1}_{obj} = 1$  then  $\mathbb{1}_{noobj} = 0$  and vice versa).

The  $\lambda$  parameter that appears in the localization loss and the confidence loss function are needed as follows. The  **$\lambda_{coord}$**  of the localization loss function is used to increase the loss of the



bounding box coordinates and to focus more on detection because the loss function only penalizes the location loss error in BB that in the IoU metric have the highest value with respect to their ground truth box. So, it is set to a constant value of  $\lambda_{coord} = 5$  in Figure 21 and 22. The  $\lambda_{noobj}$  parameter belongs to the confidence loss function shown in Figure 23. YOLO generates a huge number of bounding boxes per image and most of them do not contain any object, therefore, those BB have a confidence score of zero. It is easy to conclude that the number of cells containing objects are much lower than the ones not containing objects, we are training the model more with cells that do not have objects than with cells that do have objects. To deal with it,  $\lambda_{noobj} = 0.5$  to decrease the loss function with low confidence predictions (areas without objects).

$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

*Figure 23: Confidence loss function.*

**Classification loss** is like a normal sum-squared error that measures the error when classifying an object in a class. The function is the one shown in Figure 24 and it only penalizes grid cells containing objects, therefore the grid cells without objects are not considered by the parameter “ $\mathbb{1}_{obj}$ ” explained in confidence and localization loss.

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

*Figure 24: Classification loss function.*

Once the different functions have been explained, if we join the localization loss function that penalizes the bounding box coordinates, the confidence loss function that penalizes grid cells containing objects but have a low confidence value and the classification loss function that penalizes class-conditional probabilities of cells containing objects we obtain the total loss function. YOLO total loss function can be seen in Figure 25. The objective in the training will be to minimize the loss function with the dataset, to do so, the dataset must be big, accompanied by labelled files that describe the bounding boxes and their class.

As it can be read in paper [12], the researches explain that they have **pretrained** YOLO on the ImageNet dataset containing 1000 classes, they have done the first training for the first 20 convolution layers followed by an average-pooling layer and a fully connected layer with input image size of 224 x 224 for about 160 epochs. The technique used to optimize YOLO all through the training has been the stochastic gradient descent (explained in Chapter 2.3) with a starting learning rate of 0.1. After this first training, they increased the input size to 448 x 448 and they

trained the full neural network for about 135 epochs using the following hyperparameter values: Batch size = 64, momentum = 0.9, weight decay = 0.0005 and as an optimization algorithm the SGD with initial learning rate of 0.1. In both trainings, the learning rate was decreased dividing it by 10, once the training reaches epoch 60 (the initial training has been done with the same hyperparameters).

After training the network for about a week on the ImageNet dataset the researches of YOLO obtained a top 5 accuracy of 88% on the validation set using Darknet framework for all the training. Then, they modified the model for the prediction process adding four convolutional and two fully connected layers with randomly initialized weights, that is why in the second training they decreased the input size to 448 x 448 because the prediction model needs images of better resolution.

Finally, it has been mentioned above that YOLO uses the non-maximal suppression to delete duplicate bounding boxes of the same object with lower confidence score. This is done in the last step, once the bounding boxes lower than a threshold value have been deleted, it is supposed that only BB that have a very high confidence value remain. In most cases, for each object there will only be one bounding box, but in the case of duplicates, YOLO will apply the non-maxima suppression in the following way:

- A. Order the duplicate predictions by the confidence scores from higher to lower.
- B. Take each prediction in order and ignore it if there is a previous prediction that has a higher IoU than the current prediction.

$$\begin{aligned}
& \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}$$

*Figure 25: Loss function of YOLO.*

## 3.2 YOLOV2

YOLO, from here will be referenced as YOLOV1, faces localization errors because in the generalization process it localizes objects for new input images not very accurately. In addition, it faces low recall predictions and to improve this YOLOV2 model was developed, which is an improved version of YOLOV1 that seeks to improve on the accuracy regarding the prediction speed by improving real-time prediction. In this Chapter, the **improvements** applied to YOLOV1 resulting in YOLOV2 will be introduced and explained.

To understand the improvements, first, it is fundamental to understand the problems of YOLOV1, localization of objects has been explained on Chapter 3.2.1, but low recall has not been explained yet. Recall is the result of dividing the true positives over the sum of the true positives and the false negatives. The recall represents the ability of a model to detect the localization of all the objects.

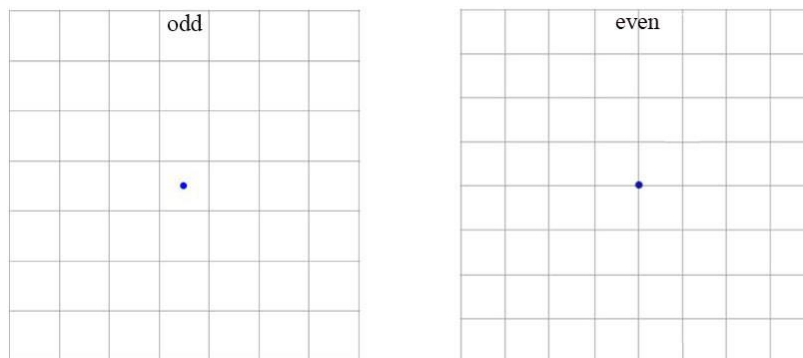
### Improvements

1. **Batch normalization:** It adds batch normalization to all the convolutional layers to scale each example to a similar range of values. This method is applied to every convolution layer and enables each layer to learn by its shelf and prevents the problem of dependency between layers. By normalising the output of each layer, the mAP score has increased by 2% because the dropout regularization has been removed.
2. **High resolution input classification:** YOLOV2 trains the classifier with input images size of 224 x 224 on ImageNet dataset. After this first training, the classifier is returned to support input images of 448 x 448 and it is trained again only for 10 epochs. With this second short training (just for 10 epochs) the model's filters adjust better to higher resolution images and can extract features for a better detection. Then the model is adjusted on the predictions. With this technique, the model is more confident with high resolution images, actually, it has been fully trained on lower resolution images of size 224 x 224 and then adjusted to support input images of higher resolution by a short training of 10 epochs. By this technique, accuracy increases up to 4% mAP.
3. **Anchor boxes:** YOLOV1 predicts BB coordinates with the fully connected layers located on top of convolutional layers that extract features and this limits each cell to predict a single object. YOLOV2 introduces an improvement through the anchor boxes that improves the recall by 7% increasing the prediction of positive cases. Anchor boxes are predefined bounding box shapes that represent width and height of real objects, for example, different cars from different images have a similar shape and therefore we can predefine BB to group the car object that will be as close as possible to the ground-truth box of the train set. By this way, when the model is predicting bounding boxes it will take the anchor boxes and it will adjust and refine the size of them to fit the objects.

Generally, what the model does is, from the training set it takes all the ground truth boxes of each type of object and groups them using K-means clustering algorithm according to their similarities, until it has all the ground-truth boxes grouped by their

similarities together. For example, if we are working with a dataset of cars and bikes, one group will contain all the ground-truth boxes of bikes and the other of cars. Then we apply IoU to each group and the result of each group will be the BB that best represents each object. Each representation of each object will be an anchor box. YOLOV2 to make use of anchor boxes makes a change on the architecture.

- Remove the fully connected layers that make the predictions of the BB together with the pooling layers to preserve the high quality of the images.
- The input size is changed to 416 x 416 in this way the image is divided in  $S = 7$  grid cells per line. Normally, most input images contain a big object in the centre of the image and in this way if we have an odd number as S parameter, we are more certain that the object will belong to the centre grid cell as it can be seen on Figure 26.
- The prediction of BB in this improved model it is done by predicting for each grid cell 4 parameters for the boundary box, 1 box confidence score and 20 class probabilities (Pascal VOC dataset). The class prediction was changed from the cell level to the BB level. For example, if we have 5 anchor boxes, we are predicting 4 coordinates per anchor box plus one confidence score (IoU between the adjusted anchor box and the ground-truth box) and 20 class-conditional probabilities that give a result of  $5 * (4 + 1 + 20) = 125$  (shown in Figure 27).



*Figure 26: The centre point of an object located in the centre of an image.*

4. **Direct Location Prediction:** For each anchor box, YOLOV2 predicts or adjusts the anchor boxes coordinates to the objects contained in the grid cells. To obtain this, the new model uses a logistic activation function to output coordinates to a range between 0 and 1. This technique is based on anchor boxes and the final predicted boxes (shown in Figure 28) of the model are formed by the parameters shown in Figure 27.

$$\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h} \\
Pr(\text{object}) * IOU(b, \text{object}) &= \sigma(t_o)
\end{aligned}$$

where

$t_x, t_y, t_w, t_h$  are predictions made by YOLO.

$c_x, c_y$  is the top left corner of the grid cell of the anchor.

$p_w, p_h$  are the width and height of the anchor.

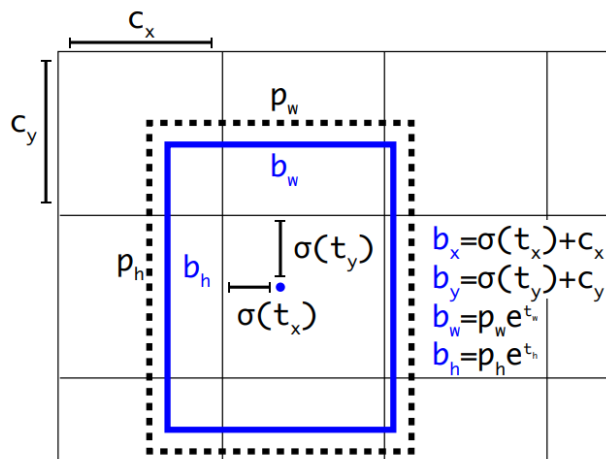
$c_x, c_y, p_w, p_h$  are normalized by the image width and height.

$b_x, b_y, b_w, b_h$  are the predicted boundary box.

$\sigma(t_o)$  is the box confidence score.

**Figure 27:** Formula that calculates the 5 parameters of a predicted bounding box based on the anchor boxes.

5. **Fine-Grained features:** YOLOV2 in each convolution layer outputs a 13 x 13 resolution. By this change, the model can detect large objects and by concatenating 26 x 26 resolution feature maps, it is able to detect small objects.



**Figure 28:** Example of the output BB calculated from an anchor box.

Once the improvements have been explained, in the following paragraphs changes on the network architecture and the new output shape will be explained. In addition, the multi-scale training technique (new training technique added to YOLO) how it works will be explained along with the YOLO9000 model.

## Network architecture:

The researchers that have developed YOLOV2, have proposed a new classification model called Darknet-19 that tries to adjust the problem of complexity and accuracy. Darknet-19 is a custom classification network that has been developed to improve YOLO model in order to improve real-time detection while increasing the speed of detection. This network is formed by 19 convolutional layers along with 5 max-pooling layers that use in the output of each of them the batch normalization technique to allow faster convergence. Each layer has 3 x 3 filters and 1 x 1 filters that the last ones are used to reduce the depth of the output of 3 x 3 convolutions. Darknet also gets very good results in accuracy since it reaches a top 5 accuracy of 91.2% on ImageNet. After all, Darknet-19 is like the backbone of the YOLOV2 model, it is the most important part.

## Output shape:

The output shape that YOLOV2 creates after making the detections of the bounding boxes for each grid cell of an input image is the following:  $13 \times 13 \times (k(1 + 4 + 20))$  where k is the number of anchor boxes. For example, if  $k = 8$  the output shape of the model will be **13 x 13 x 200**.

## Multi-scale Training:

The datasets that YOLOV2 model can use to train have images of different sizes, maybe some images have higher resolution than others. The multi-scale technique is used in the training process, no matter what the size of each image is because the network is trained for different input sizes. The network is not responsible for resizing the input image, every 10 batches it chooses a new image dimension size that is picked up randomly from this set {320, 352, 384, ..., 608}. Every time it chooses a new image dimension, the network is resized to that dimension and it continues with the training.

This resizing can be done because YOLOV2 architecture is made up of convolutional and pooling layers and every 10 epochs each layer can be resized to support new images resolution. By the use of this technique, after the training of the model, it will be able to predict images from the test set that are in different resolutions or have different input shapes. For example, the network has been resized 4 times during a training of the model, so it has been trained on 320 x 320, 384 x 384, 416 x 416 and 608 x 608 resolution images. Once the training has finished, the model will be very confident on detecting objects of different resolutions (especially in the ones it has been trained).

## Detecting 9000 or more classes with YOLO

Object detection applications must deal with the problem that object detection is restricted to a small set of objects due to the lack of datasets containing many objects. ML techniques such as regression or classification just need datasets where examples are labelled by one class (normally a number referring to a specific class). Therefore, those algorithms can pre-process millions of data because it is easier to find datasets labelled to train classification algorithms than datasets labelled with bounding boxes for object detection. On the internet there are less datasets for object detection because they require images containing a very high number of

objects where each object in each image is labelled by a bounding box, that is, it has an associated file containing the position (x, y ,width, height) of the BB and the class index. As a result, it is easier to find a dataset addressed to classification problems than to object detection.

To face this problem, **YOLO9000** model based on YOLOV2 was developed. It is a new model capable of detecting over 9000 different classes that it has been trained on by joining the COCO detection dataset and the ImageNet classification dataset. Its objective is to join two different datasets of which one oriented to classification problems and the other to object detection applications and reduce the gap between the two. YOLOV2 learns from the COCO dataset the bounding box positions and patterns to know how to localize the object in the image. From the ImageNet dataset, it learns new class categories that are not labelled in the COCO dataset, but it will be able to detect these new classes. The COCO dataset only contains common objects like “dog” or “car”; however, the ImageNet classification dataset has a deeper range of classes because for example for the class dog it has more than a hundred of dog breeds. The idea behind this model is that it will be able to detect objects and localize them with the bounding box and classify them with a class from the ImageNet dataset even though that class does not exist in the COCO dataset. It is worth mentioning that this model is very interesting and has been fully trained by the authors of YOLOV2, but it will not be useful for this work because our objective is to train a model capable of detecting just three objects. That is why in the next point we introduce and explain an improved version of YOLOV2 that will be the one used in our application.

### 3.3 YOLOV3

**YOLOV3** is the third version of the original YOLO real-time object detection model. It starts predicting the same way as the YOLOV2 model, using anchor boxes that have been created by grouping the ground-truth boxes with a K-means clustering algorithm and taking the box by the IoU between all the boxes of each group. Each bounding box is predicted by 4 coordinates and a confidence score that are calculated as shown in Figure 27. In addition, in Figure 28 it can be observed how adjusting anchor boxes, a predicted bounding box is obtained. To explain the new network advances compared to previous versions we will go from the most basic to the most complex:

#### **Architecture (Feature Extractor Network)**

As it has been mentioned in the paragraph of network architecture of YOLOV2, the developers combined YOLOV2 with a Darknet-19 custom network and in this new version it has been replaced by a Darknet-53 custom network. It is formed by 53 convolutional layers that are responsible for extracting the features along with an average pooling layer before the fully connected layer (shown in Figure 29). The features are extracted in the convolution layers composed by filters or kernels of dimension 3 x 3 and 1 x 1. 3 x 3 filters are used for the feature extraction, but 1 x 1 filters are mainly used to reduce the depth of the output of some convolutions. For example, in Figure 29 the first two convolutional layers are formed by a set of

3 x 3 filters, then the third convolution layer takes as input value the feature map that is the output of the second one and by applying the 1 x 1 filter it reduces the depth of the feature map. The 1 x 1 kernel converts the feature map of a convolutional layer of dimension 7 x 7 x 1024 into 7 x 7 x 125, in short, it converts an output of 1024 channels into 125. This channel reductions are very necessary to improve the speed of prediction because the YOLO (V1, V2, V3) model is aimed at real-time detection.

The architecture was also improved by adding residual networks that are built of Shortcut Connections that are necessary for the gradient descent. In very deep neural networks formed by large layers (the case of YOLOV3), the gradient descent faces problems to propagate backwards the error in the training process. This problem is solved by adding Shortcut Connections that create “short cuts” as the name implies, for gradients to propagate backwards faster and further. In this way, they allow an efficient training for the YOLOV3 model because the first layers get the weights updated much faster than before.

	Type	Filters	Size	Output
	Convolutional	32	3 x 3	256 x 256
	Convolutional	64	3 x 3 / 2	128 x 128
1x	Convolutional	32	1 x 1	
	Convolutional	64	3 x 3	
	Residual			128 x 128
	Convolutional	128	3 x 3 / 2	64 x 64
2x	Convolutional	64	1 x 1	
	Convolutional	128	3 x 3	
	Residual			64 x 64
	Convolutional	256	3 x 3 / 2	32 x 32
8x	Convolutional	128	1 x 1	
	Convolutional	256	3 x 3	
	Residual			32 x 32
	Convolutional	512	3 x 3 / 2	16 x 16
8x	Convolutional	256	1 x 1	
	Convolutional	512	3 x 3	
	Residual			16 x 16
	Convolutional	1024	3 x 3 / 2	8 x 8
4x	Convolutional	512	1 x 1	
	Convolutional	1024	3 x 3	
	Residual			8 x 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

*Figure 29: Example of Darknet-53 network architecture.*

## Class Prediction

In any type of problem where an example has to be classified in a class, if there are 5 distinct classes, the sum of the probabilities that the example belongs to each class must be equal to 1. In the first version of the YOLO model, when predicting the class confidence score of a bounding box, if the model was working with 5 classes, the bounding box prediction will return 5 class confidence scores which will add up to 1 in total. This means that the classes are mutually exclusive or independent from each of them, the intersection between any pair of classes equals zero. In the third version of YOLO, multi-label classification is presented, that is, the sum of the



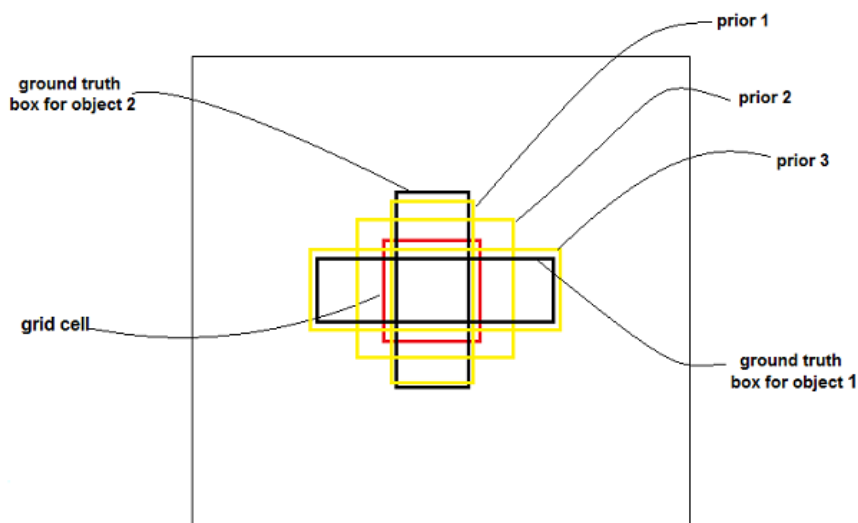
probabilities will be greater than one. For our specific problem of cars, we will work with three classes, therefore, the classes car, truck and motorcycle are not independent from each other and if a bounding box contains a car, the truck confidence score will be higher than usual because a truck and a car have similarities.

This peculiarity generates a couple of changes in the model, it changes the SoftMax activation function with an independent logistic classifier that calculates the probability an input has to belong to a specific class. Now the classification loss does not calculate the mean square error when classifying objects and it computes the binary cross-entropy loss per predicted BB.

## Bounding box prediction

For each bounding box predicted it calculates an objectness or confidence score (*assurance the model has that the specific bounding box contains an object*) using logistic regression. This computation is done by calculating the overlap between a ground truth object and the bounding box, if the overlap is the highest one for all the bounding boxes observed before, then the score will be 1. In the other case, if the overlap of a BB with respect to a ground-truth box is not the best but it is higher than a threshold = 0.5 the prediction is ignored. The bounding boxes with a score of 1 totally incur in the loss function, however, the rest of the bounding boxes, do not incur to the cost function.

In Figure 30 we can see that the prior 1 is the bounding box that most overlaps the ground truth box for object 2, this means that has the highest IoU with respect to the rest prior boxes in the image (prior 2 and 3). Prior 3 is the predicted box that most overlaps the ground truth box for object 1 with respect to the rest prior boxes in Figure 30. The model associates to each ground truth box object to one prior box, so if a prior bounding box has not been associated to any ground truth object it will not affect the loss function for the localization and classification loss, only on the confidence loss. Referring to our example in Figure 30, prior 1 and 3 will incur in the loss function because their overlap is the highest one and they are scored with 1, however, prior 2 box is not associated to any ground truth box and will only incur to the confidence loss inside the loss function.

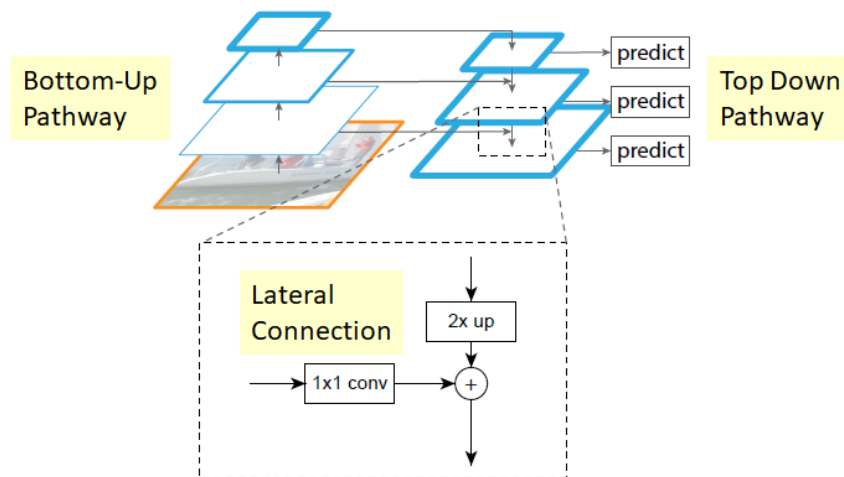


*Figure 30: Example of prediction for a grid cell of a YOLOV3 model.*

## Feature Pyramid Networks (FPN)

YOLOV3 makes three predictions or predicts three bounding boxes per grid cell, each prediction is composed of the coordinates of the location of the object, the confidence score that the box contains an object and 80 class scores (i.e.  $S \times S \times [3 \times (4 + 1 + 80)]$  predictions in total). YOLOV3 makes predictions at three different scales by extracting features from those scales like in FPN (shown in Figure 31 how the prediction process looks like):

- First prediction in the feature map of the last convolutional layer.
- The second prediction is done by going back two layers and augments the first predicted feature map by 2, then takes the feature map that has higher resolution and merging it with the augmented feature. Apply some filters to the resulting feature map.
- Repeat the previous step until the resulting feature map has high-level structure, for example, the localization of the objects is quite clear, the object resolution is high, etc.



*Figure 31: Example of predictions in an FPN.*

### Small objects detection:

YOLOV1 and YOLOV2 models are not very confident when detecting small objects, thanks to residual networks that contain Short Cut Connections and have been explained on the architecture paragraph (shown in Figure 29) YOLOV3 performs better when detecting small objects. By the use of Short Cut Connections, the feature extractor can obtain finer-grained information and be able of detecting smaller objects. Despite this, what it gains in precision for small objects it loses for larger objects because it has a worse performance comparing with YOLOV2.

After having thoroughly studied each YOLO version, we have concluded that the newest version is YOLOV3 and so we have decided that this is going to be the one used to develop our vehicle detection application. YOLOV3 is a very new model that has had a lot of improvements compared to the first version and therefore we believe that it will be the one with the best results. In the next Chapter SORT (Simple Online Realtime Tracking) tracker will be explained, how the tracking process of detected objects between frames is done and why is so efficient.

## 4. TRACKING WITH SORT

SORT (Simple Online Realtime Tracking) is an implementation of a tracking-by-detection framework that is oriented to multiple object tracking (MOT) problems in videos. This tracker by the help of an object detection model, detects objects in each frame and represents them by bounding boxes. Once the detection of the objects has been done by the model, to track the objects of the current frame, the tracker uses the objects detected from the previous frame along with the current frame. The detection quality has a big impact on the real time tracking performance and that is why SORT must be very well combined with the object detection model. During this chapter the different techniques that SORT uses for real-time tracking will be explained such as the Kalman Filter and the Hungarian algorithm focused on the tracking components.

### 4.1 Estimation model

SORT uses as a representation model a motion model to propagate a target into the next frame. For each object detected, it is associated to a “Target” (like the BB of YOLO) that stores all the necessary information about the state of the object in that particular frame. The state of each target is represented by 7 variables  $x = [u, v, s, r, u', v', s']$  where  $(u, v)$  is the horizontal and vertical pixel location of the centre of the detected object (pixel-centre), the  $s$  represents the area of the target and  $r$  is the aspect ratio of the bounding box. The coordinates  $(u', v', s')$  are the respective velocities of the centre coordinates and the scale (area) of the target. Targets travel through the different frames of the video and are an essential characteristic to track objects because when the model detects an object in a frame, SORT looks back to all the previous targets and can associate a previous target to the detected object or can create a new target.

When the model makes a detection in one frame, if it is associated to a previous target, the detected bounding box coordinates are used to update the state of the target, the target that comes from the previous frame is predicted in the current frame by the BB of the predicted object and the velocity components are predicted by a Kalman Filter. If the detection is not associated to any previous target, its new target state is predicted by the linear velocity model that is independent from other objects. When previous frames targets have not been associated to any detection for a long time, they are deleted because this means that the objects have left the video. Also, for the first few frames, it may happen that duplicate targets are created, to prevent this, a threshold value is set, and lower targets are deleted.

At the start, the video is divided into frames and is analysed frame per frame. In the first frames, new objects without any target are detected because they have not appeared yet, so the SORT tracker creates a target for each detected object and the state is predicted by the linear velocity model with the values of the position of the object  $(u, v, s, r)$  and their respective velocities  $(u', v', s')$ . After a few frames have been analysed, when predicting objects in new frames, some of this objects will be associated to a previous target because they are the same object than in the previous frame and some other objects appear for the first time so again a new target will be created for them.

## 4.2 Data association

To assign a detected object to a previous target, SORT predicts its new location in the frame by an estimated bounding box geometry. The detection of the object and the prediction of the previous target are independent from each other, this means that they need a metric to associate both. The IoU distance between each detection and all the predicted bounding boxes from the targets is the metric that SORT uses to quantify the association. The assignment cost matrix is another metric responsible for associating the data and it is calculated by applying the IoU (metric to quantify the association) mentioned before. Once this matrix has been computed, the Hungarian algorithm is used to solve the assignment between the predicted object and one of the previous targets.

## 4.3 How the Hungarian algorithm and the Kalman filter work

The **Hungarian algorithm** takes as input value a cost matrix  $C$  and computes  $C'$  as follows :

1. For each row it computes the minimum value and subtracts it to all the elements of the row like the formula that can be seen in Figure 32.

$$\Rightarrow C'_{i,j} = C_{i,j} - \min C_{i,j}$$

*Figure 32: Formula for each row with respect to the original matrix.*

2. For each column it computes the minimum value and subtracts it to all the elements of the column (To the matrix computed in the previous step).

$$\Rightarrow C'_{i,j} = C'_{i,j} - \min C_{i,j}$$

*Figure 33: Formula for each column with respect to the matrix computed in the previous step.*

3. If for all the rows of  $C'$  exist a column with a cost of 0 and has not been associated to any other row, the algorithm finishes, if not it repeats step 1 and 2 again.

The Hungarian algorithm is mainly used for assignment problems, for the SORT tracking algorithm it is very useful in the data association process because it resolves the assignment cost matrix using the process described above and gives a solution to the problem.

The **Kalman filter** algorithm is a recursive algorithm that is very used to track moving objects, it is mainly used to estimate the velocity and the acceleration of an object that is moving between frames. The computation of the velocity and acceleration is done because the object has a localization in the image and therefore it can be calculated by the Kalman filter formulas. This algorithm is used in the estimation model and it is used to predict the future location of an object that has been predicted in a frame, to reduce the noise that is introduced by inaccurate detections and to facilitate the process of association of multiple objects to their trackers. Figure 34 show in what step is the Kalman Filter applied in a multi object tracking algorithm and the Figure 34 shows how the Kalman Filter computes the future position of a detected object.

Figure 35: computation of the future position of a prediction

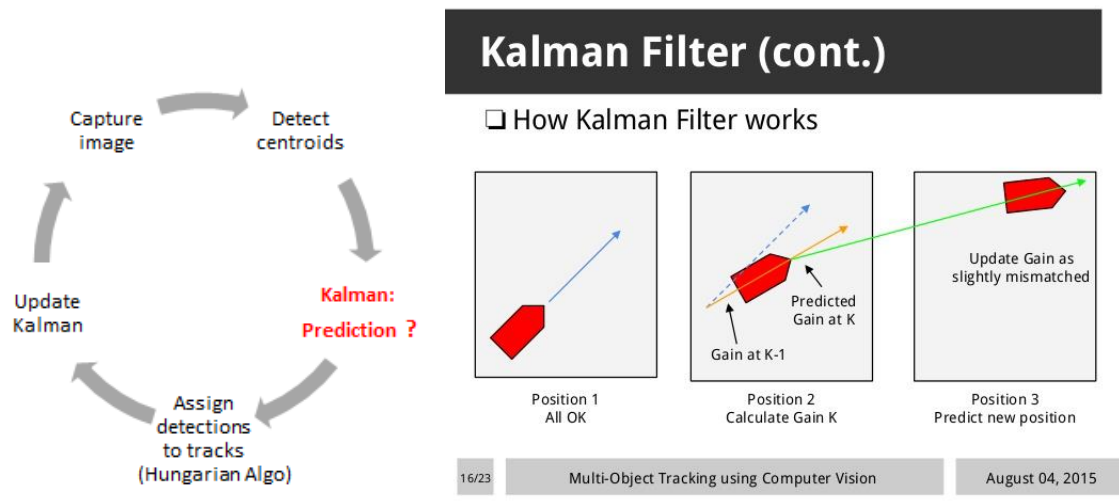


Figure 34: Steps of a multi object detection algorithm.

#### 4.4 Creation and deletion of track identities

When an object first appears in a video, this means that when the model is analysing the video frame per frame and detects for the first time a particular object, a unique identity has to be created for that object, that will be called a tracker. The track is initialised with the coordinates of the predicted bounding box and the velocity variables of the tracker are set to zero. When the tracker of a new predicted object is created the velocity cannot be computed and therefore the covariance of the velocity is initialised with large values. After the new tracker has been created in a frame, it is necessary that in the following frames it is associated to other predicted objects in order to ensure that it has been created correctly and to prevent false positives.

The tracker is deleted when it has not been associated to any detection in a threshold of frames since it was created. This threshold has been set to 1 ( $no\_associated > 1$ ) because if in one frame a tracker has been created and associated to a predicted object, then the next frame is studied. If in the next frame the tracker has not been associated to any object it will not be deleted yet as it may be a tracker bug, so the model will analyse the following frame and if there is also no object associated to the tracker then it will be removed. In the Figure 36 it is shown how a tracker tracks an object frame per frame.

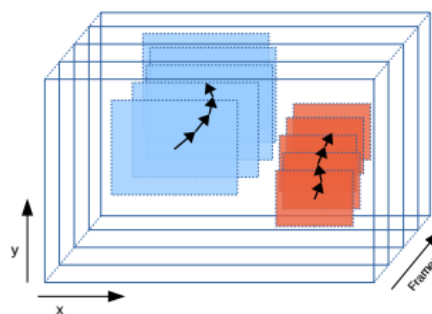


Figure 36 : Example of a tracking in a video.

## 5.YOLO AND SORT FOR OBJECT-DETECTION AND TRACKING IN REAL TIME

The aim of this project is to develop an application fast and accurate enough to accomplish the objective of tracking and detecting vehicles in real-time. YOLOV3 is going to be the object-detection model that is going to be trained and optimized to detect our custom objects (car, truck, motor) and the SORT algorithm will be responsible of monitoring frame per frame the vehicles detected. The application that is developed in this end-of-degree work is specially designed to improve the video vigilance cameras of autonomous vehicles and for improving the management of the traffic of smart cities. The application could be used in video vigilance cameras located in no pedestrian streets; our model will count in a video sequence how many different cars have passed. If we compare the result obtained to the average number of cars that normally pass on that street at that time, we can find out whether there is a lot of traffic or not so that we can notify a user to avoid that street.

Another possible application for the model described in this work could be to fit our model in the camera of an autonomous vehicle in order to follow a predetermined vehicle that will go directly in front. By this way, our application will be capable all times to recognize the predetermined vehicle in the pursuer car's camera and thus be able to perform the same route as the vehicle in front does. In the following Chapters will be explained what steps have been followed to develop the vehicle detection and tracking model.

### 5.1 Dataset

From now on, when we talk about vehicles, we will only refer to cars, trucks and motors. The fundamental part for realizing a model of vehicle detection is to find a database that contains cars, trucks and motorcycles that have been properly labelled. YOLOV3 is an object-detector model and it has a feature extractor part that is formed by convolutional layers. The feature extractor part takes as input data the images from the dataset and learns patterns and features that finally end up in a general overview of the classes in the dataset. The model is going to be trained to detect cars, trucks and motors so it is fundamental to find a dataset that at least has those classes. To make this learning process as accurate as possible, the images of the dataset must contain the following characteristics:

- Most of the images of the dataset must contain vehicles, at least one, but also is very important to have a few images that do not have any vehicle because in this way the model will learn to not detect vehicles in images that do not have them. For example, analysing traffic videos, it can happen that some frames do not contain vehicles, so the model needs to learn not to detect vehicles in those frames.
- The vehicles of the images must be of different sizes, shapes and heights. For example, some pictures may contain vehicles very well cantered and that can be perfectly

appreciated in the image, but others perhaps are very far away and are more difficult to appreciate.

- The images must be populated by vehicles of different shapes, colours and brands. Also, it must contain vehicles from different angles and positions.
- Summarizing, the dataset must contain vehicles in all kinds of situations where they can be found in real life, so that our model can detect them in any circumstances. After all, the model is aimed at real life and therefore has to be trained with all the possible situations that a vehicle can be found in real-life.

Apart from containing images our dataset also must be composed of labelled files. During the career, the datasets that have been used are focused for classification or regression problems where each example was labelled just by pointing out the class they belong to. For this work, we need a txt file for every image where each row corresponds to a vehicle that appears in the image. Each row is the bounding box of the object that is formed by a vector (class, x-coordinate, y-coordinate, width, height). X and y coordinates are the point centre of the object and the four coordinates have been normalized. For example, if our dataset is composed of 200 images, each image has to be accompanied by a bounding box labelled file, so we will have 400 files both images and plain files (.txt) for the training process. It is very important to have the labelled files because the model apart from classifying the detected object, it has to be capable of locating it in the image, therefore, if the dataset contains images without the labelled file, we will have to label them manually.

The dataset is called the Berkley Deep Drive Dataset and has been obtained from the following link: <https://bdd-data.berkeley.edu/portal.html#download>. As it has been mentioned in Chapter 3, YOLOV3 networks backbone is called darknet, and darknet only supports .txt files. Each image of the Berkley dataset must be accompanied by a plain text file containing the objects and their corresponding bounding box localization in each row. This dataset contains 100k images inside a folder that is divided into three subfolders, the training folder, the validation folder and the test folder. Moreover, it contains a unique labels file that is in json format, so it is fundamental to convert to the darknet format by a python script and after the conversion, we end up with 100k images and 100k .txt files. The amount of data is too high to be processed by a normal computer that has an ordinary intel processor and a ram with a capacity of 8 GB. Reducing the number of images to 31260, the dataset will be more manageable by our computer and we will be able to host it on a free server such as GitHub. Apart from the 31260 images we have another 31206 files that give a total of 62466 training files (there are less files than images because some images do not contain any object, they just contain background).

The final dataset has been hosted in a GitHub repository and can be accessed with the following link, <https://github.com/urtzi98/yoloTrainFiles>. We have concluded that it is essential to host it here so it can be accessed from any tool, be it Google Collab or from a Python notebook. In Table 1 it can be seen how the images have been divided into the training and the validation set, for testing with images without labelled files we will take some images from the test set and we will see if the model is good at generalization and can detect all the vehicles that appear.

Type of set	Number of images	% from the total size
Train set	21260 images	68%
Validation set	8000 images	25.6%
Test set	2000 images	6.4%

Table 1: Train, validation and test set sizes.

The training set will be used to train the model and the validation set and test set will be used to evaluate how good are the network configuration hyperparameters selected. The dataset will be changed only once during the different trainings (the change will be explained in Chapter 5.2) and we will play with the parameters to see which ones work better.

## 5.2 Train, evaluate and test YOLOV3 to detect vehicles

The first step is to train YOLOV3 model using the dataset that has been explained before, the model will be trained with different parameters and after all the best configuration will be chosen to make the detection process. Before we start talking about the training process we have followed for the network, we must first explain what tools and frameworks have been used for this part of the work.

### 5.2.1 Tools and frameworks:

For the training process it is very important to use a GPU because the network is going to process 62466 files of which 31260 are coloured images. Training a CNN model is very hard in terms of time and complexity, if the model is trained using a CPU it may last days and even weeks. To avoid so, if the model is executed and trained supporting GPU the runtime is reduced severely and can run trainings in few hours depending on the size of the dataset. The computer that has been used during this process does not have a GPU that could support all the process of a Deep Learning model and therefore we have been forced to look for hosts on the internet that offer to run our models in powerful GPU.

- **Google Collab:** It is a free service where Python notebooks and Linux commands can be executed and it offers a free GPU of 12 GB that will be the one used for training, evaluating and testing our model. It has many Python libraries installed and if we need more libraries, it is possible to install them as in a Linux platform by writing installation commands in the shell. Also, it is possible to clone GitHub repositories and store them on the server giving you access to them from the platform, keep in mind that Google deletes all the stored data every 12 hours.

YOLOV3 can be implemented with many libraries that Python supports such as Keras, PyTorch (we will talk about it in the next Chapter), TensorFlow, etc., or also it can be implemented in the programming language C++. For this part of the work we will use Darknet, it is an open source neural network framework that has been written in C and CUDA and the most interesting part is that it supports GPU computation. To implement our model, we have based ourselves on the Darknet framework (<https://github.com/AlexeyAB/darknet/>) but making numerous changes to



the code in order to adapt the model to our dataset and thus be able to carry out a good training and a following evaluation. Darknet must be built to support GPU (default off), CUDNN (default off) and OpenCV (default off) in order to be run on the GPU of Google Collab.

### 5.2.2 Training and evaluation:

The dataset in the start is labelled for 10 classes {bus, traffic light, traffic sign, person, bike, truck, motor, car, train, rider} and has been trained for 60 epochs using the network configuration of Table 2. The network has been trained for 60 epochs that are 20000 iterations (4 epochs are 1328 iterations) because darknet framework trains a maximum of 2000 iterations for each class and calculates the mAP@0.5 every 4 epochs for the validation set and every 3 epochs for the test set unless the difference between the cost of an iteration and the cost of the next iteration is lower than 0.0005. In Figure 37, it can be observed how the loss decreases as the epochs go by, the lowest loss value obtained during the training is 6.08. The lowest value is quite high because complex datasets like the one we are using, usually obtain a lowest value between 2 and 4 when the training has been done correctly. In the graph of Figure 37, the loss function does not vary much from epoch 40 onwards and stays between 6 and 7 on the average loss. In addition, the best values for the loss function are obtained in epochs 57 and 60 as it can be seen in the first graph.

Parameter	Value
Batch size	64
Subdivisions = sub batches: number of mini batches it is split up the batch	32
Width x height of the input images	416 x 416
IoU minimum threshold between predicted BB and ground-truth	0.5
Data augmentation	Saturation and exposure = 1.5 angle = 0 hue = 0.1
Learning rate	0.001
Max_batches	Number class x 2000
Momentum	0.9
Decay	0.0005

Table 2: Initial values for the YOLO network parameters.

After that first training, the dataset has been modified by removing the classes that are not necessary for the application. Rows of .txt files that contain BB of objects that are not necessary have been removed and the index of the car, truck and motor have been updated as follows {'0': car, '1': truck, '2': motor}. This change on the dataset has been done to see if when training the network with the new dataset and same parameters as in Table 2 the loss gets values around 3 or 4. This second training has been done for 18 epochs and the lowest loss value that has been obtained is 3.7 that is much lower than the one obtained in the first training. For this reason, we have decided to keep the modified version as the final dataset because we get better results and the training is much faster than with 20 classes (the model is just trained to detect 3 classes

instead of 20). The best loss value is obtained in the epoch 17 as it can be seen in the second graph but from the epoch 12 the loss function takes very low values.

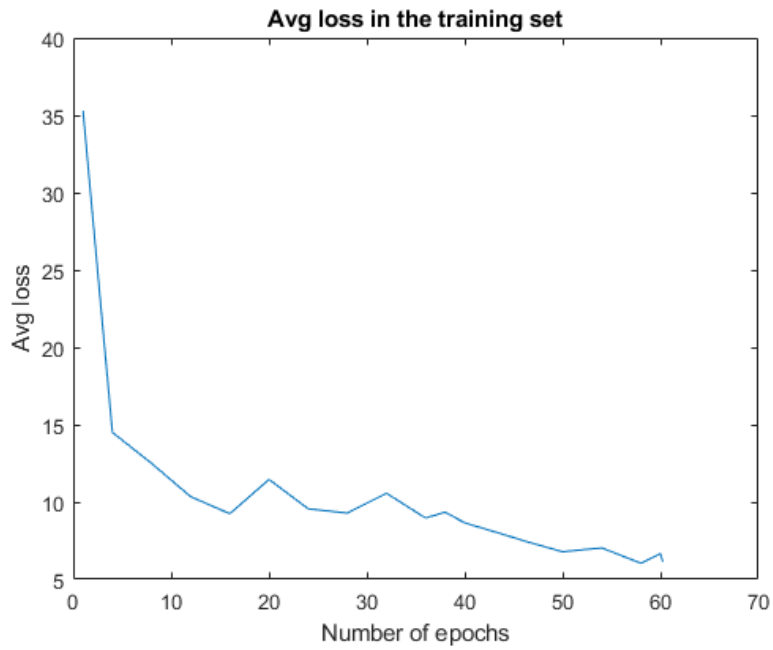


Figure 37: Graph of the average loss for the dataset of 20 classes with parameters on Table 2.

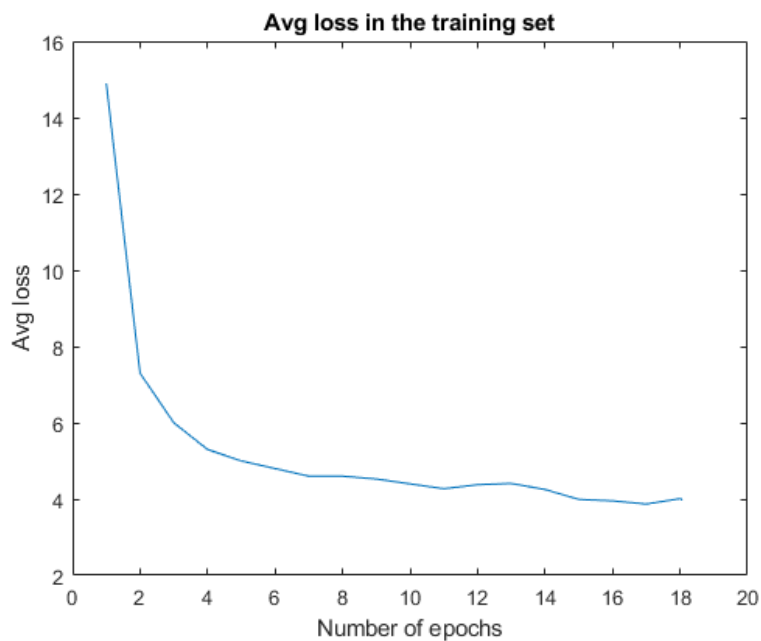


Figure 38: Graph of the average loss computed on every epoch for the dataset of 3 classes with parameters on Table 2.

The training with the dataset of the three classes with the network configuration of the table 2 has been evaluated on the validation set and test set. YOLOV3 on the darknet framework saves the weights files every three epochs, so for this training we will have stored 6 weights files that

correspond to the epochs 3, 6, 9, 12, 15, 18. During the training the mAP has been calculated on the validation set and it is computed every 4 epochs (epochs: 4, 8, 12, 16, 18). For computing the mAP on the test set it is possible to compute it after the training has been finished, evaluating the network with the weight files stored every 3 epochs.

Epochs	Ap car	Ap truck	Ap motor	Precision	Recall	F1-score	mAP@0.50
3	38.35%	12.08%	0.00%	0.52	0.42	0.46	16.81%
4	39.09%	20.01%	0.03%	0.52	0.43	0.47	19.71%
8	49.63%	29.08%	2.25%	0.58	0.52	0.55	26.99%
12	50.99%	35.48%	6.77%	0.58	0.54	0.56	31.08%
16	57.31%	36.94%	12.30%	0.64	0.57	0.60	35.52%
18	57.03%	37.48%	11.33%	0.64	0.57	0.60	35.27%

*Table 3: Evaluation of the model on the validation set with an IoU threshold of 50% that is the Area-Under-Curve for each unique Recall.*

Epochs	Ap car	Ap truck	Ap motor	Precision	Recall	F1-score	mAP@0.50
3	39.26%	13.38%	0.00%	0.52	0.42	0.47	17.55%
6	48.70%	26.60%	2.75%	0.59	0.48	0.53	26.02%
9	53.65%	29.12%	10.61%	0.58	0.55	0.57	31.13%
12	54.39%	34.70%	12.36%	0.59	0.56	0.58	33.82%
15	58.23%	37.81%	16.02%	0.64	0.58	0.61	37.36%
18	57.77%	38.69%	15.14%	0.63	0.58	0.60	37.20%

*Table 4: Evaluation of the model on the test set with an IoU threshold of 50% that is the Area-Under-Curve for each unique Recall.*

For evaluating which weight file is the best one to use for detecting vehicles in new images we will guide by the mAP value computed in the validation and test set. In the Table 3 and 4 we can see which weights files give best value on the mAP. The training has been done for 18 epochs, but it can happen that the weights computed in previous epochs give a better value on the precision and the mAP. This may happen due to overfitting; the model is very good detecting on the training set but falls to detect objects in new images. As it can be seen in Table 4, the weight file stored in the epoch 15 and the epoch 18 obtain the best mAP value of 37.20% and in the Table 3 the weight file obtained in the epoch 16 gets a 35.52% mAP best value.

In the Figure 39 we can see the graph that shows the average loss computed for every iteration and the mAP calculated for every 4 epochs (1 epoch = 332 iterations so 4 epochs = 1328 iterations). The value of the loss remains between 3.5 and 4 and the mAP has a value of 35% from the iteration 5000 (epoch 15) to the iteration 6000 (epoch 18). With the Figure 38 and 39 and the Tables 3 and 4 we can conclude that the best weight file for detecting vehicles in new images is obtained in the epoch 15 or 18. Note that the dataset is quite complex as it has many labelled small vehicles in each images that are not visually easy to see, this complexity affects the mAP value by obtaining a relatively low one. This happens because the model fails to detect some vehicles in the validation set that it is difficult even for the human eye to detect them. On the next pages, to test the capability of generalizing of our model for this training, we will get one image containing cars, other image containing trucks, other image containing motors and a

final image containing cars, trucks and motors and pass them from our model to see if it is able to detect all the vehicles in the images.

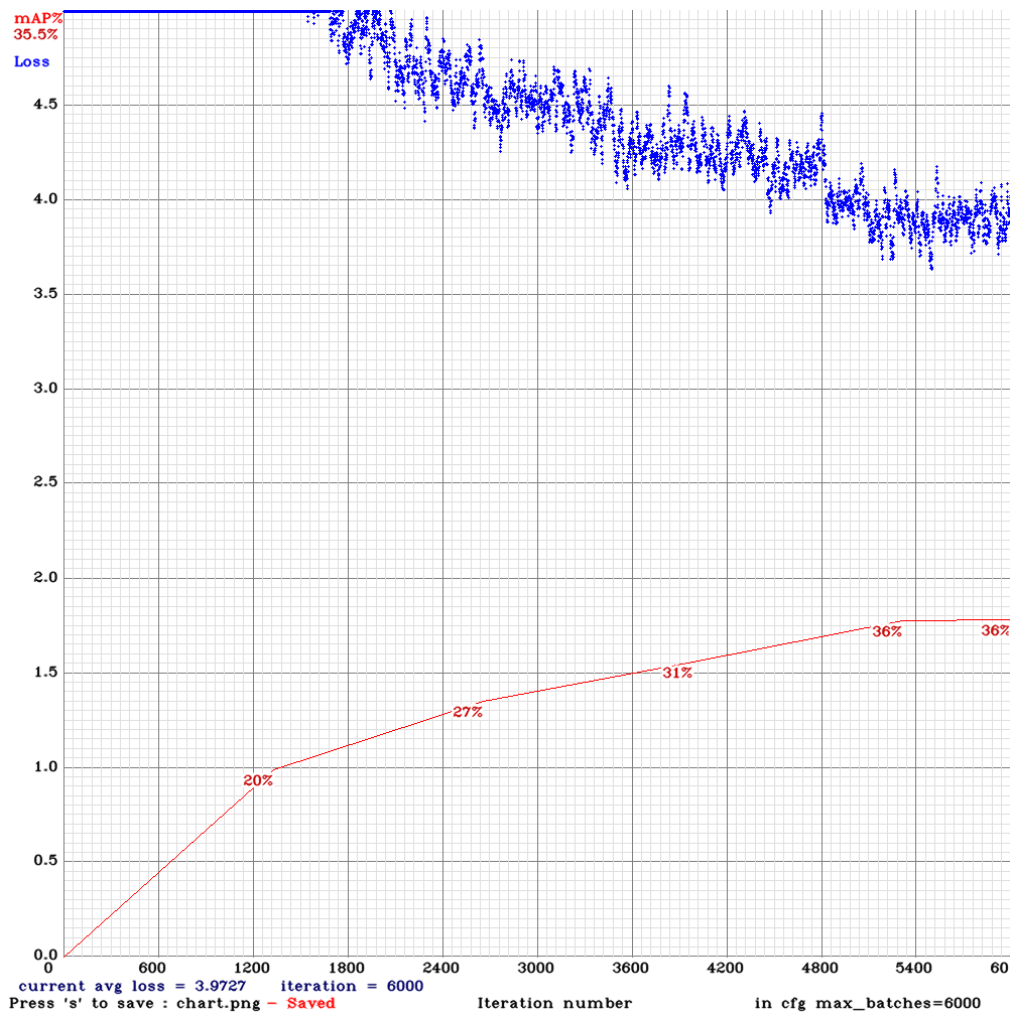


Figure 39: Graph computed during the training; the mAP is computed over the validation set.

Our YOLOV3 model outputs in the detection process, for each input image, the output image with the objects predicted fitted in bounding boxes and the class confidence score for each object. The Model rejects predicted objects where their class confidence score is below 25%, the threshold value to detect a class object is 0.25. In Figure 40 it is shown how the model generalizes when detecting cars in an image, the input image is composed of four cars and the background is the street of a village. The model has been very well trained to detect cars because the class confidence score for the four cars detected is higher or equal than the 80% for all of them and the predicted bounding boxes localize extremely well the four cars. It should be added that the first three cars are appreciated quite easily by the naked eye, but the fourth car may look fuzzy and distant and nevertheless the model detects and locates it perfectly.

In Figure 41 we can see that the model is good at detecting trucks, but the class confidence score is quite low because the truck that is further, the model detects it with a confidence of 47%. In Figure 42, we can see how the model detects motors in an image, in this case it detects just one motor although two motors are clearly visible in the image. In Figure 43, we can see that the

model is quite good at detecting different class objects in the same image because it is capable detecting three cars and one truck. We will try other trainings by modifying the initial values of the network configuration in order to see if the accuracy of the model is improved.

```
car: 95%  
car: 96%  
car: 98%  
car: 80%
```

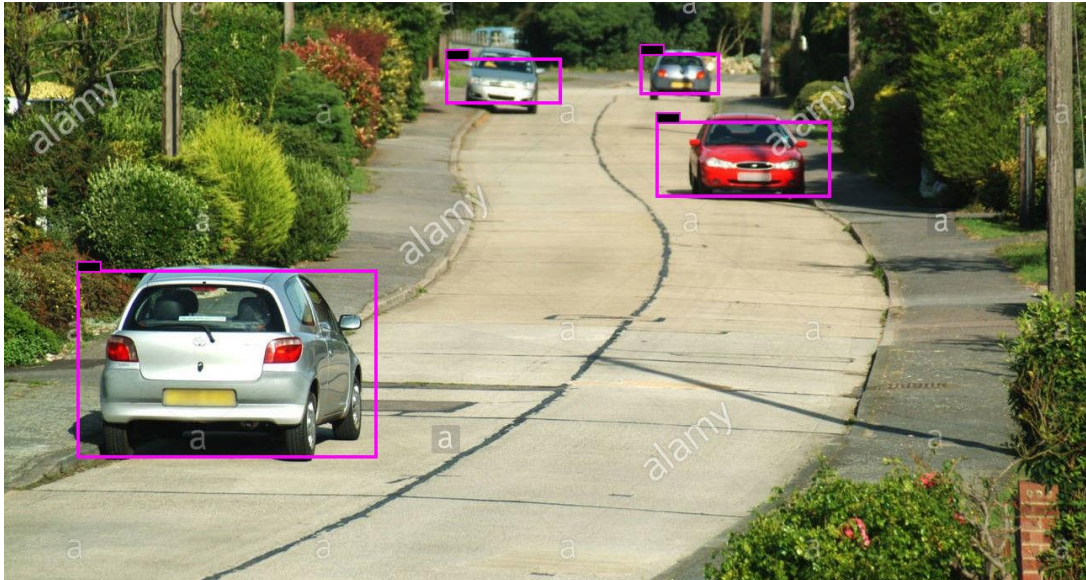


Figure 40: Prediction of cars in an image and the class confidence score the model has that the object detected is a car.

```
truck: 79%  
truck: 47%
```



Figure 41: Prediction of trucks in an image and the class confidence score for each truck detected.

motor: 30%



Figure 42: Prediction of motors in an image and the class confidence score.

car: 96%  
car: 68%  
truck: 89%  
car: 87%

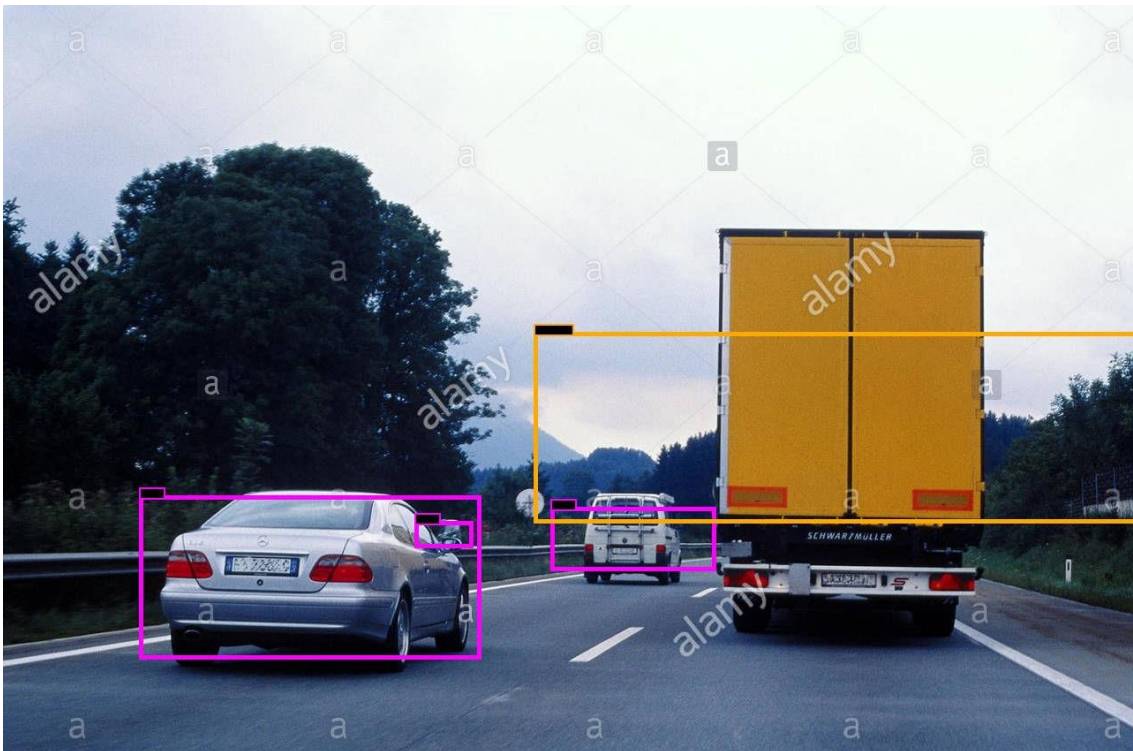


Figure 43: Prediction of cars and trucks in an image with their corresponding class conf. score.

The next training has been done by doing Fine-tuning instead of transfer learning, the initial weight file has been changed by a partial weight file obtained from the pre-trained model of YOLOV3 (trained on the ImageNet and COCO dataset). Stop backward technique has been activated for all the layers in order to speed up the training and the evaluation results on the validation set can be seen on Table 5.

Epochs	Ap car	Ap truck	Ap motor	Precision	Recall	F1-score	mAP@0.50
4	20.57%	2.87%	0.00%	0.33	0.32	0.33	7.82%
8	39.68%	12.78%	0.03%	0.51	0.44	0.47	17.50%
12	44.91%	17.41%	1.80%	0.54	0.48	0.51	21.39%
16	49.08%	21.98%	5.23%	0.58	0.51	0.54	25.43%

Table 5: Evaluation of the model trained on Fine-tuning with IoU threshold 0.5.

The results obtained are much worse than in the previous training and therefore it was decided to stop the training at the epoch 16 as it can be seen in the graph of Figure 43. The loss value is much higher than in the other training, rounding a value of 4.5 at the end as best value. Trainings that do not get average loss values between 2 and 4 are not accurate as it can be seen in the best mAP@0.5 obtained on Table 5 and therefore the weights obtained in this training will not be useful.

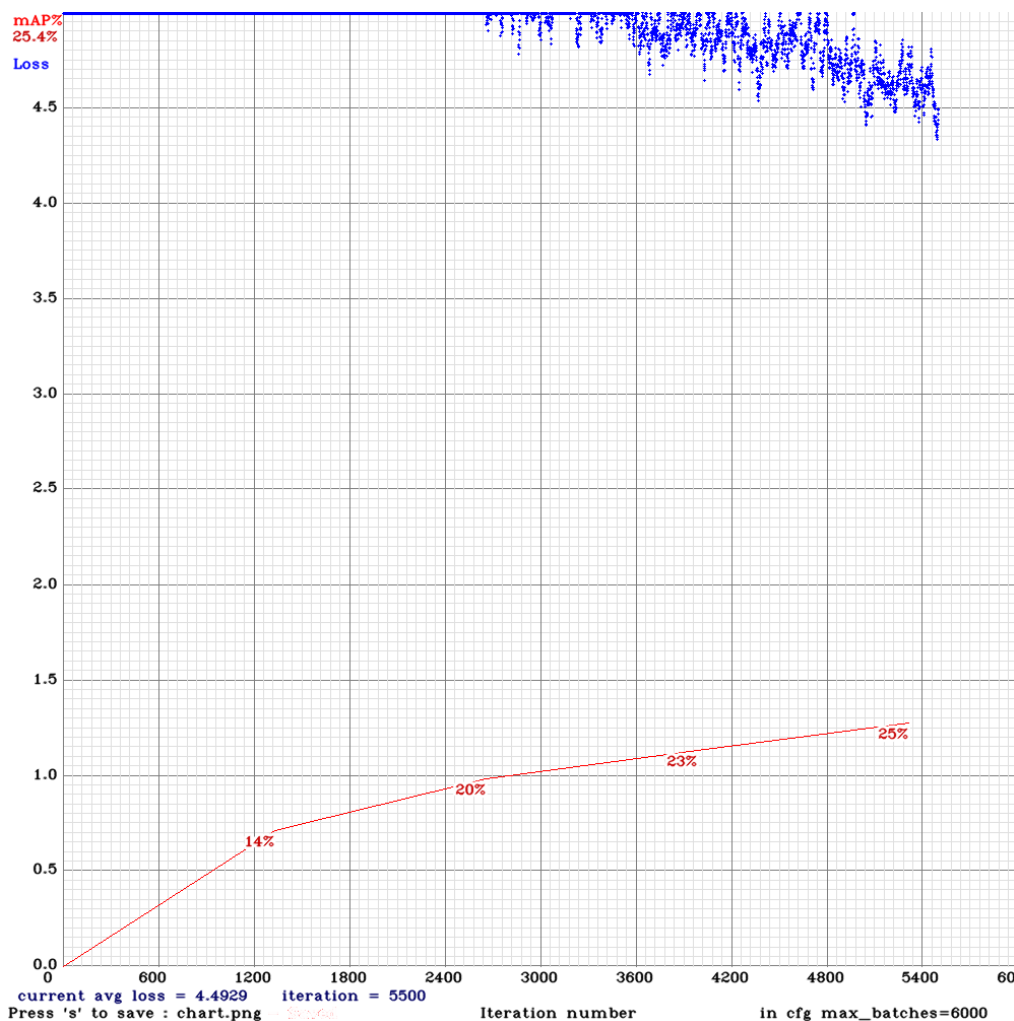


Figure 44: Graph computed during the training; the mAP is computed over the validation set.

Some other trainings have been done by augmenting the width and height of the input images to 608 or 832 (increasing the network resolution) in order to increase the precision of the network and make it possible to detect smaller objects but all these attempts have failed as the trainings stopped when they took only 300 iterations. Also, it has attempted to train the network by setting the subdivisions value to 8 or smaller values but the network breaks out the training again.

One last training has been done to see if the model can be improved to localize better the trucks and motors and this time it has not given any problem. The objective of this training is to obtain a model that has a higher mAP with respect to the previous best training model where a mAP = 35.52% in the validation set and a mAP = 37.36% in the test set has been obtained. In this case, we have changed some initial values of Table 2 as follows:

- Subdivisions = 16 (subsets that the batch size is divided into) and the network does not break out the training.
- Data augmentation values have been set as angle = 0, saturation = 1.8, exposure = 1.8 and hue = 0.3.

This is the best configuration of data augmentation values for complex datasets formed of images with a big number of objects in each image. For example, our dataset is quite complex because it has a big subset of images where each image contains more than 20 objects labelled in the txt files. This means that some objects that are in images with lots of objects might be difficult for even the human eye to distinguish, and with this data augmentation configuration we are helping the network to better see these small objects. Comparing Table 6 and Table 7 with Table 3 and Table 4 we can see that the improvement of this training compared to the other is very small because in the epoch 15 on the test set we get a 37.93% mAP that is a little better than the previous best training, 37.36% mAP. Therefore, we could use both the weight file of the epoch 16 of this training or the weight file of the same epoch of the previous best training.

Epochs	Ap car	Ap truck	Ap motor	Precision	Recall	F1-score	mAP@0.50
4	40.12%	13.64%	0.00%	0.58	0.38	0.46	17.92%
8	43.31%	25.07%	0.27%	0.53	0.47	0.50	22.88%
12	52.75%	31.96%	7.67%	0.60	0.54	0.57	30.79%
16	57.27%	37.48%	11.13%	0.64	0.57	0.60	35.46%
18	56.63%	38.00%	11.33%	0.62	0.57	0.60	35.21%

Table 6: Evaluation of the model on the validation set with an IoU threshold of 50% that is the Area-Under-Curve for each unique Recall.

Epochs	Ap car	Ap truck	Ap motor	Precision	Recall	F1-score	mAP@0.50
3	41.04%	14.05%	0.37%	0.58	0.39	0.46	18.49%
6	44.22%	24.74%	1.87%	0.53	0.48	0.51	23.61%
9	51.40%	31.53%	7.70%	0.56	0.53	0.55	30.21%
12	53.58%	32.23%	8.38%	0.60	0.55	0.57	31.40%
15	58.30%	37.42%	18.52%	0.64	0.58	0.61	37.93%
18	57.77%	38.69%	15.44%	0.63	0.58	0.60	37.27%

Table 7: Evaluation of the model on the test set with an IoU threshold of 50% that is the Area-Under-Curve for each unique Recall.



All the trainings we have done so far have been done with the anchor boxes by default brought by darknet. Finally, it has been decided to calculate the anchor boxes for our specific dataset, to compute the anchor boxes we have set the number of clusters to 9 and the width and height to 416 as at all times the network has been trained for this input resolution. The calculated anchor boxes (width, height) are as follows: (9, 6), (17, 11), (28, 20), (40, 33), (60, 51), (99, 33), (93, 77), (150,102), (225,149). The objects of the dataset have been clustered into 9 different groups with the 9-means clustering algorithm, then for each group a particular width-height ratio has been calculated which fit as closely as possible to the objects of the group they represent. Then, the network configuration has been updated in order to support the new anchor boxes and a final training has been done to see if we are able to get a better model for vehicle detection.

Epochs	Ap car	Ap truck	Ap motor	Precision	Recall	F1-score	mAP@0.50
4	43.12%	14.32%	0.08%	0.53	0.43	0.48	20.56%
8	49.73%	29.90%	2.03%	0.54	0.52	0.53	27.22%
12	55.03%	31.66%	4.87%	0.60	0.55	0.57	30.50%
16	57.78%	37.25%	7.5%	0.64	0.56	0.60	34.09%
18	59.02%	38.87%	9.12%	0.65	0.57	0.61	36.53%

Table 8: Evaluation of the model on the validation set with an IoU threshold of 50% that is the Area-Under-Curve for each unique Recall.

Epochs	Ap car	Ap truck	Ap motor	Precision	Recall	F1-score	mAP@0.50
3	41.02%	13.59%	0.00%	0.59	0.37	0.46	18.21%
6	50.48%	26.01%	2.72%	0.57	0.51	0.54	26.38%
9	55.70%	29.97%	10.64%	0.64	0.53	0.58	32.10%
12	55.77%	31.94%	9.34%	0.60	0.55	0.58	33.03%
15	58.55%	38.77%	12.77%	0.64	0.58	0.61	36.69%
18	59.34%	39.10%	12.87%	0.65	0.58	0.62	38.04%

Table 9: Evaluation of the model on the test set with an IoU threshold of 50% that is the Area-Under-Curve for each unique Recall.

If we compare this last training with the first one we did, we can observe through Figures 39 and 45 that the loss value takes values close to 3.5 in this last training. However, in the first training, the loss is a little bit higher and does not decrease as much as in the last training. Moreover, the best mAP computed in the test and validation set is obtained at the epoch 18 and is higher than the best mAP obtained in the first training, therefore we will test the model to detect objects in the same images as before and we will observe if the model is able to make detections with greater precision and recall.

Training the model with specific anchor boxes for the dataset instead of anchor boxes by default is better because the Ap of the car, Ap of the truck, precision, recall, F1-score and the mAP gets higher values and this means that the model will be more accurate at detecting vehicles for new images. The comparison can be done by contrasting the results obtained in Tables 6 and 7 (default anchor boxes) and the results obtained in Tables 8 and 9 (specific anchor boxes of the dataset), so the weight file used to see the capability of generalization in the same images of Figures 40, 41, 42, 43 will be the one obtained on the epoch 18 that has a mAP@0.5 on the validation set of 36.53% and a 38.04% on the test set. This values of mAP@0.5 obtained in this last training are the best obtained so far, so this should be the best vehicle detection model.

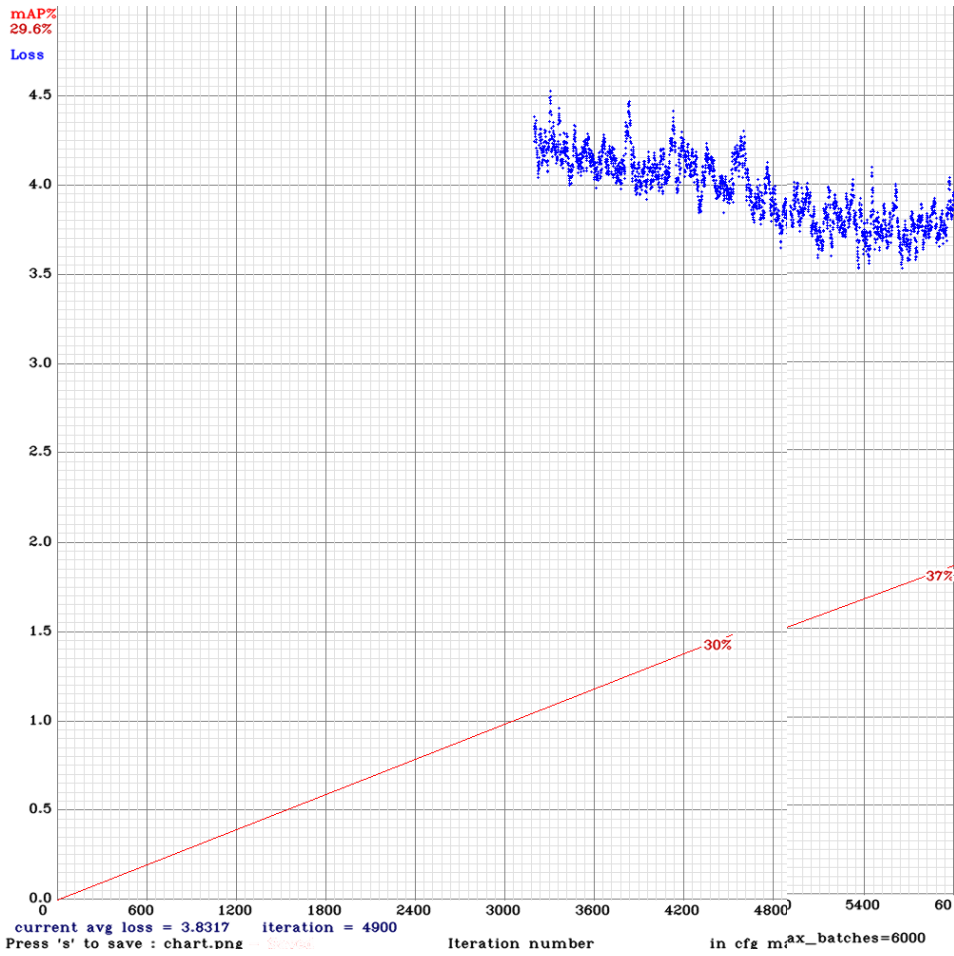


Figure 45: Graph computed during the training; the mAP is computed over the validation set.

car: 98%  
 car: 78%  
 car: 99%  
 car: 96%

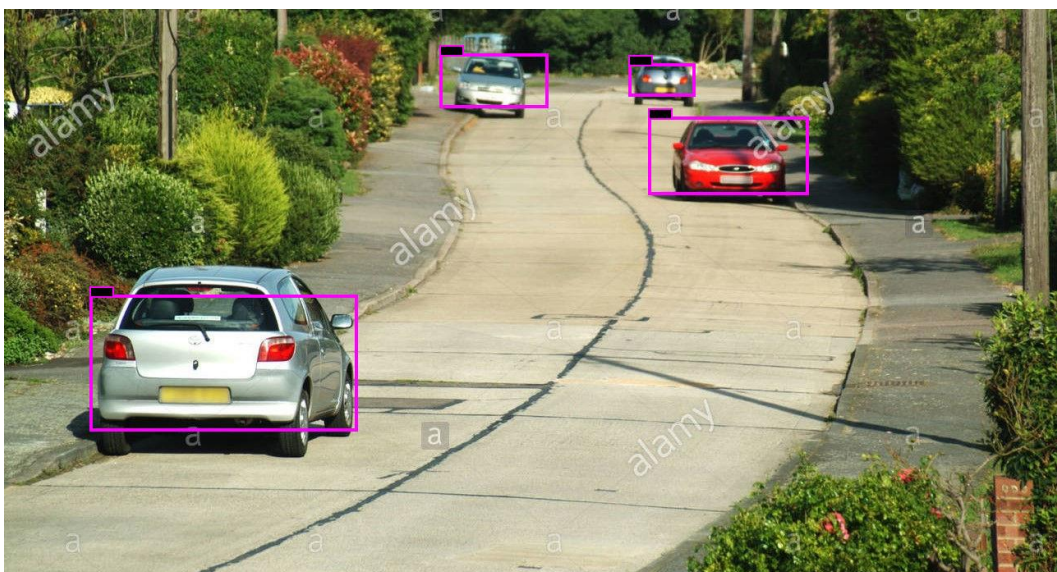


Figure 46: Example detecting cars for the model trained with generated anchor boxes.

For the detection of cars, in the Figure 46 we can see that the confidence the model has that the four objects are cars is a little bit higher than the detection done by the other trained model in the Figure 40. Both models are very good and accurate detecting cars, but this last model is even more accurate because the three closest cars are detected with a probability of 99%, 98% and 96%. In the case of the Figure 47 that contain two trucks, the model has a confidence of the 90% that the closest object is a truck and a 51% that the second object is a truck. Here we observe a greater improvement regarding the Figure 41 because the other model detects the trucks with a confidence of 79% and 47% that is lower than the one obtained with this last trained model. It also detects two cars with a 35% and 33% of accuracy that are very difficult even for the human eye to see, so the accuracy improvement is remarkable. In the case of the image of Figure 48, we can see that the model detects one motor again out of two but in this case, with a higher probability that is 45% compared to the Figure 42 that obtains a confidence probability of 30%. It also detects two cars in the background like the other model (purple boxes), so both models are capable of detecting cars and motors in the same images.

For the last picture in Figure 49, the closest car and the unique truck are detected with an assurance of 99% and 98% respectively. The other trained model detects these two objects with an assurance of 96% and 89% so definitively the accuracy has been improved in the last training. Also, although the new model does not detect a fourth car (it is not very necessary since it is hardly seen in the picture) the third car is detected with a probability of 91% better than the 87% of the other model.

Figures 50 and 51 show more detections of new images to test how good the vehicle detector is and if it could be used to along with the SORT algorithm.

```
car: 35%  
truck: 51%  
car: 33%  
truck: 90%
```

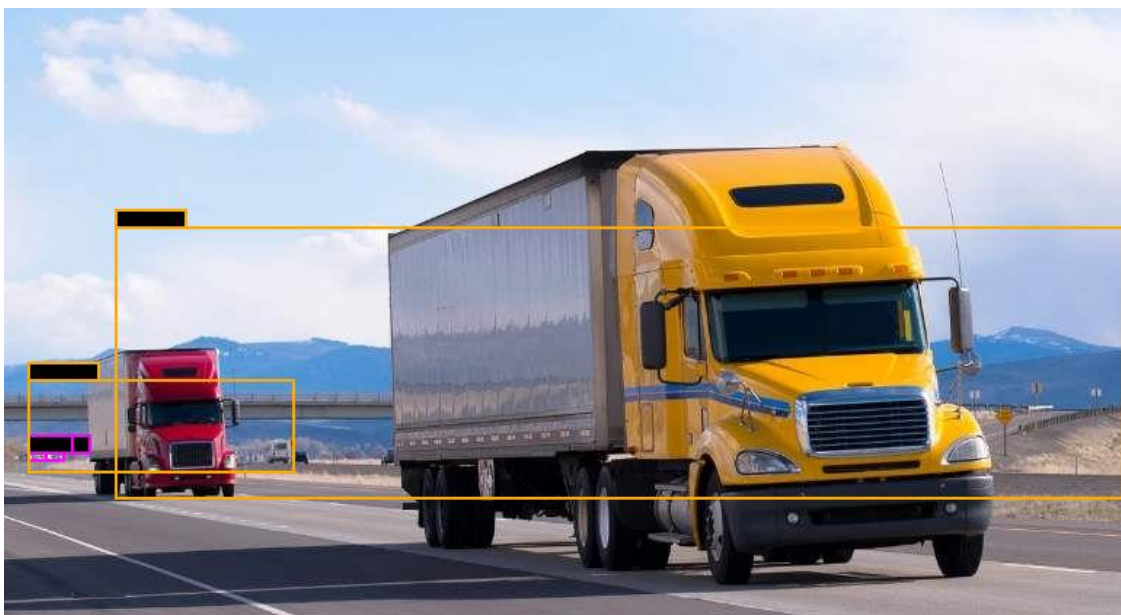


Figure 47: Example of detecting trucks for the model trained with generated anchor boxes.

motor: 45%

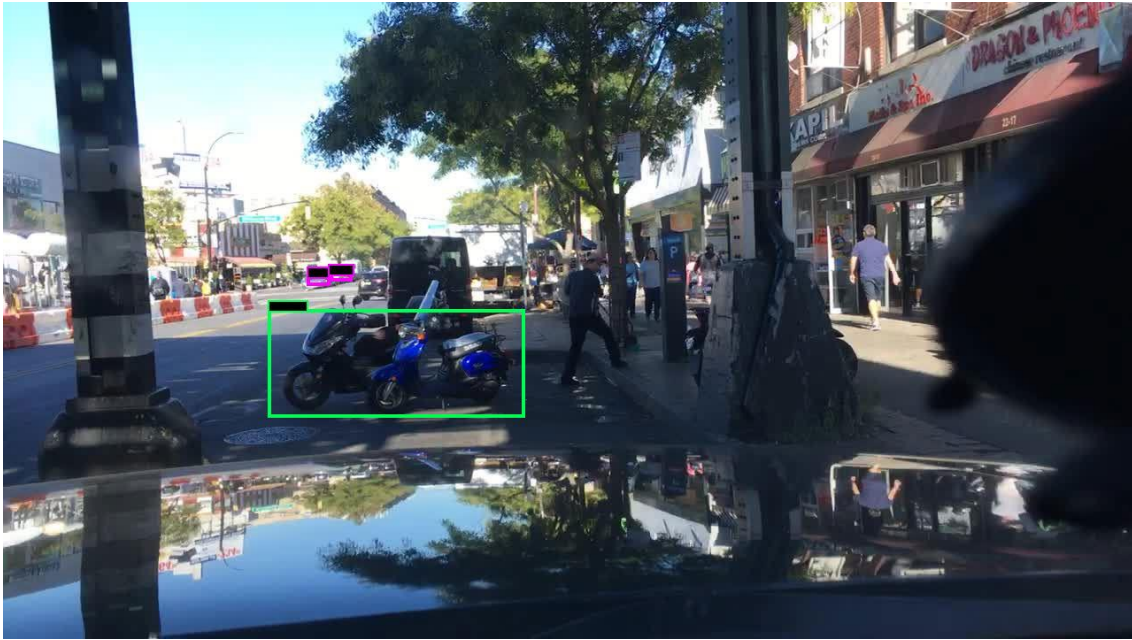


Figure 48: Example of detecting motors for the model trained with generated anchor boxes.

car: 99%  
truck: 98%  
car: 91%

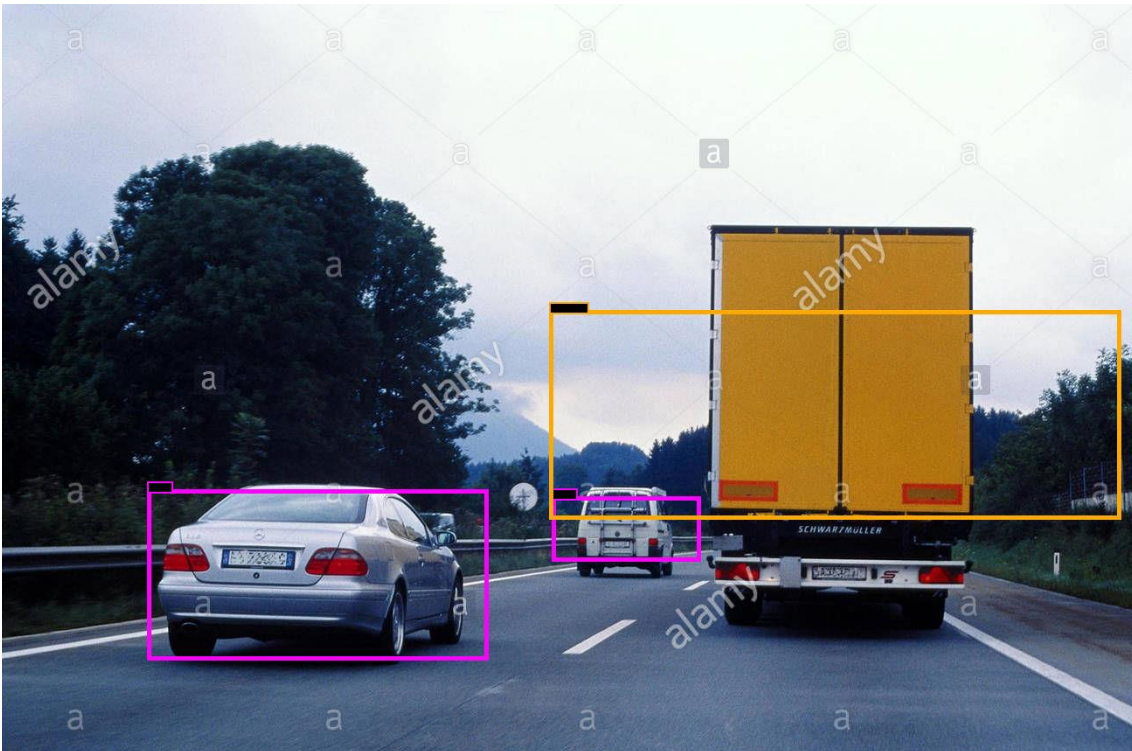


Figure 49: Example of detecting cars and trucks for the model trained with generated anchor boxes.

car: 38%  
car: 88%  
car: 85%  
car: 96%  
car: 96%  
car: 67%

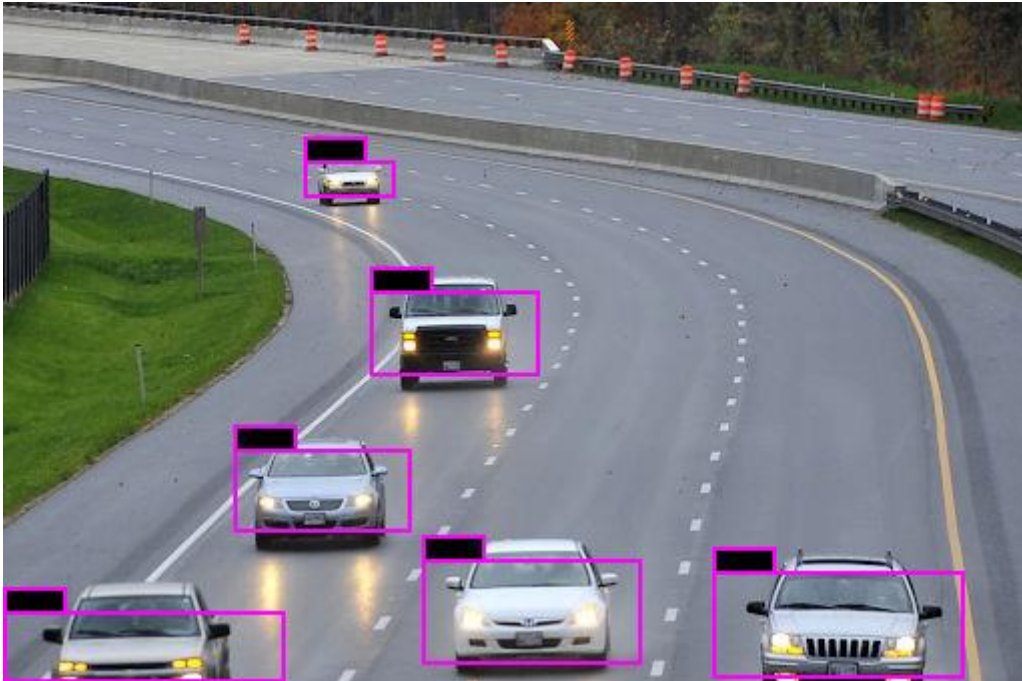


Figure 50: Example of detecting cars.



car: 53%  
car: 88%  
car: 98%  
car: 45%

Figure 51: Example detecting cars with their respective assurance.

In the next Chapter we will join YOLOV3 best detection model obtained here and the SORT tracker. Also, we will make some tests in order to obtain a good tracking model that can detect and track vehicles in real time.

### 5.3 Join YOLOV3 and SORT to track vehicles

The last step to develop the application of this final-degree-work is to combine the object detection model with the SORT tracker model. Table 10 shows the best network configuration of the object detection model that has been obtained in the Chapter before and adds two new parameters for the object detection model that are the confidence threshold and the Nms\_threshold. The confidence threshold will be set to 0.6 at the start of the testing, it is the class confidence score and it rejects detections that have lower values than the threshold, its value will be changed all through this chapter to see which is the best one. The Nms\_threshold refers to the minimum IoU between different BB predicted to consider that they are predicting the same object, but also it will be changed all through the tests to see which is the best value. YOLOV3 and SORT will be combined using the PyTorch library based on the Torch library of Python used for ML and DL techniques.

Parameter	Value
Batch size	6
Subdivisions = sub batches: number of mini batches it is split up the batch	1
Width x height of the input images	416 x 416
IoU minimum threshold between predicted BB and ground-truth	0.5
Data augmentation	Saturation and exposure = 1.8 angle = 0 hue = 0.3
Learning rate	0.001
Max_batches	6000
Momentum	0.9
Decay	0.0005
Confidence threshold	0.6
Nms_threshold	0.3

Table 10: Best network configuration for test mode.

### PyTorch framework

It is a Python package designed to perform numerical calculations using tensor programming. It also allows its execution in GPU to speed up the calculations. Normally PyTorch is used both to replace NumPy (simple matrix and vectors library) and to process calculations in GPU because this framework was built for researching problems in the field of Deep Learning, focusing on the development of neural networks. PyTorch uses internally CUDA, an API that connects the CPU with the GPU that has been developed by NVIDIA in order to make faster all the operations that are needed by a neural network.

For developing the whole application, we will base on the code that appears in the following link: [https://github.com/cfotache/pytorch\\_objectdetecttrack](https://github.com/cfotache/pytorch_objectdetecttrack), but we will make some changes to the code in order to adapt it to our project. In the URL, darknet framework has been implemented in Python using the PyTorch library and will be the one used to load the best

weight files obtained in the Chapter of training YOLOV3, the network configuration file and the class names file in order to make the detection process. When processing the video we have made some changes: reduce the video resolution, change the frame-by-frame analysis, reduce the video shape in order to focus the analysis on the interest point, add the confidence threshold variable, add the OpenCV library (processes videos at 30 frames/sec) and implement functions to load videos and to store them in Google Collab, implement functions to show the analysed frames in the command line, etc.

In the following pages we will discuss the different changes and tests that have been made to the parameters of the network. For example, we will see every how many frames the video must be analysed in order to track the vehicles preserving a fast running time. Also, we will play with the video resolution as our model is able to detect vehicles very well in low resolution videos and in this way the speed of detection and tracking is improved. Finally, we will focus the detection and tracking process to a specific area of the videos where the vehicles circulate leaving aside areas of little interest for our work.

### 5.3.1 Tests on the detection and tracking model

The testing will be done for two different videos made from different points of views, one video has been taken from a camera located in a bridge of a highway and the other video has been taken from the co-driver's seat. The model will be analysed for both videos to detect and to make the tracking. The tracking must be almost perfect since it has to count the number of cars that appear in the video (especially in the highway video). It is very important to set a good frame threshold because OpenCV library analyses videos at 30 fps and this is too much for a real time vehicle detection and tracking model.

#### Highway Video

The video records a highway with lanes in the left with cars that go in one direction and lanes in the right with cars that go in the other direction. It is fundamental to focus on one direction, for this case we will focus on the right lanes, this means that every time the model takes a frame it will reshape it only to the part the right lines are. The video has a total size of 1920 x 1080 at the start and our model will reshape the video to a size of 800 of width and 600 of height (800 x 600) to decrease the initial resolution. Reducing the video resolution, the speed of the model is increased when detecting and tracking and therefore we can approach to the real time object detection and tracking. In Figure 52 we can see how a frame looks like once it has been reduced.

To adjust the frame threshold, we will make some tests by taking only the part of the Figure 52 that goes from 400 to 800 in the width and from 0 to 600 in the height (right half of the frame). During the 12.46 seconds that the video lasts, 4 different cars appear, the confidence threshold will be set to 0.6 and we will see with what frame threshold we can track each car with a unique tracking number and the processing time is the lowest one. In the Table 11, we can see that the best results are obtained when making the detection and tracking process every four frames because the processing time is the lowest one obtained, and 3/4 cars are tracked with a unique tracking number. The second car of the video has not been detected in any of the tests, so we

will make new changes to the frame shape and the confidence threshold value to see if we can detect and track the car. For future tests the frame threshold will have always a value of 4.

Frame threshold	First car tracking numbers	Second car tracking numbers	Third car tracking numbers	Fourth car tracking numbers	Processing time
1	1,2	-	27,28,35	33,34,40	23.17 seconds
2	69	-	70,72	73	12.47seconds
3	94	-	95,98	97	9.71 seconds
4	101	-	103	106	8.12 seconds

Table 11: Results obtained when testing for different values for the frame threshold.

The next step is to reduce the frame only to the area we are interested. In the Table 12 we can see what results we obtain on the detection and tracking method, reducing the area to a height between 200 and 500 or 300 and 600 with a width between 400 and 800 we obtain the best values. The new frame shape can be seen in Figure 53 and will be used to make the last tests on the confidence threshold value, the new shape of the frame with respect to figure 52 will be [400:800,300:600]. The new frame has less background and is much smaller, that is why the model has less detections and it has gained time that with it the speed of the model has improved.

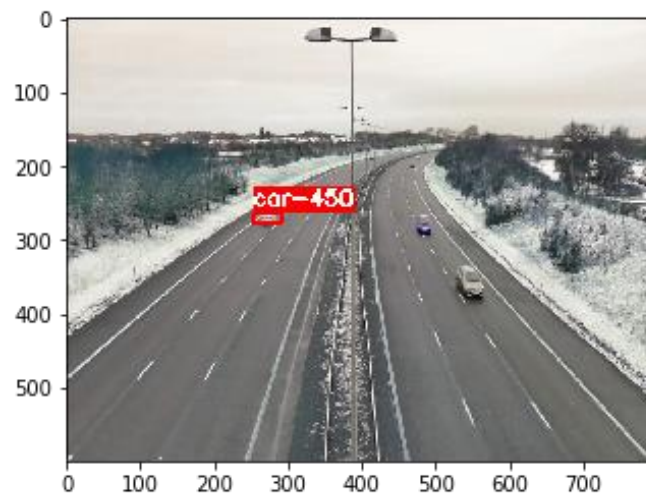


Figure 52: General view of the video reduced to (800,600)

Reduced area respect Figure 52 [width, height]	First car tracking numbers	Second car tracking numbers	Third car tracking numbers	Fourth car tracking numbers	Processing time
[400:800,0:400]	-	122	123	-	7.99 seconds
[400:800,100:400]	131	132,133	-	143	9.93 seconds
[400:800,150:450]	148	150,151	-	-	8.09 seconds
[400:800,200:500]	153	158,159	160	-	7.76 seconds
[400:800,300:600]	161	168	173	-	7.85 seconds
[400:800,350:600]	245	246,245	-	-	7.75 seconds

Table 12: Results obtained testing for different interest areas in a frame with respect to Figure 52.





Figure 53: Reduced frame to the right lanes with shape (400,300)

Due to the reductions made to the frames, the model will detect the vehicles with less confidence score, this means that it needs to be reduced in order to detect more vehicles in each frame. The objective will be to set a frame threshold (every how many frames the model analyses) where the four cars are detected preserving a processing time equal to or little less than the duration of the video. In the Table 13 it can be seen that the best confidence threshold value is between 0.23 and 0.25, all the cars are tracked only by one tracking number and the processing time is quite low. We will choose the confidence threshold of 0.23 seconds to see without printing the analysed frames what is the processing time and we obtain 7.33 seconds (373 frames at 0.019 s/frame). The objective of detecting and tracking vehicles in real time has been obtained because the model processing time is less than the duration of the video, in addition, the model detects 4 cars that are the total that appear in the video.

Confidence threshold	First car tracking numbers	Second car tracking numbers	Third car tracking numbers	Fourth car tracking numbers	Processing time
0.5	-	251	-	253	7.99 seconds
0.4	-	260	-	262	8.23 seconds
0.3	271	272	-	282	9.43 seconds
0.25	285	284	288	293	10.67 seconds
0.23	296	295	299	304	10.12 seconds

Table 13: Results obtained testing for different interest areas in a frame with respect to Figure 52.


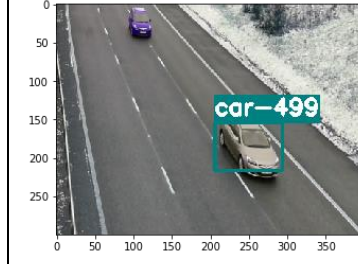
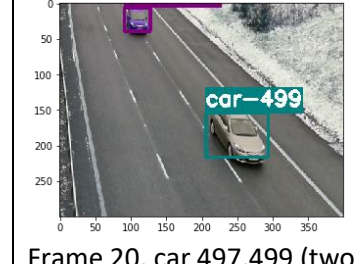

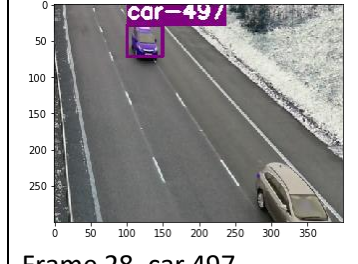
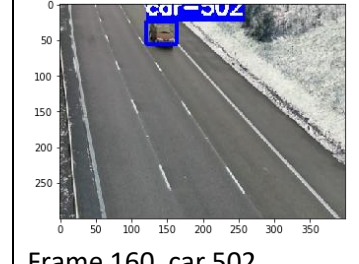


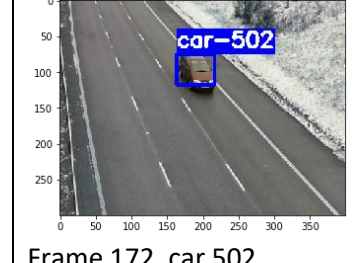
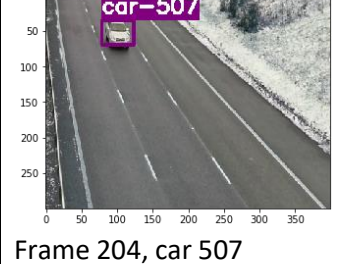
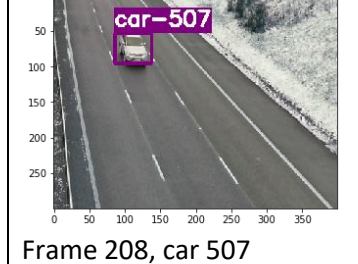
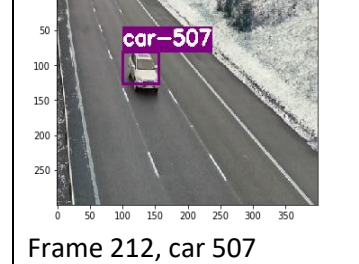
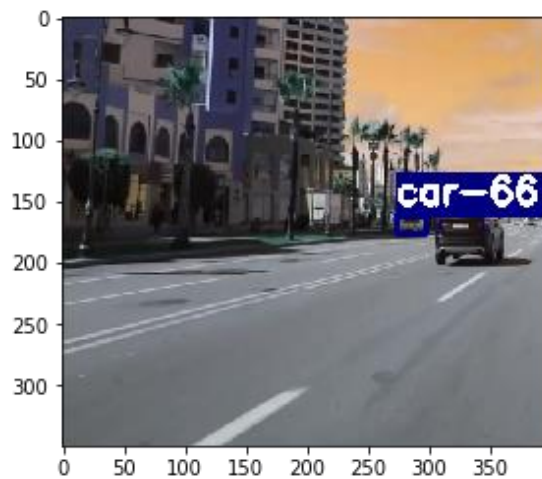
 Frame 16, car 497	 Frame 20, car 499	 Frame 20, car 497,499 (two cars detected in one frame)
 Frame 24, car 497	 Frame 28, car 497	 Frame 160, car 502
 Frame 164, car 502	 Frame 168, car 502	 Frame 172, car 502
 Frame 204, car 507	 Frame 208, car 507	 Frame 212, car 507

Table 14: Cars detected and tracked with the id in the highway video.

In the Table 14 we can see how the four cars are tracked by the model using a confidence threshold of 0.23, a frame threshold of 4 and the area of interest point of [400:800,300:600]. The first car detected by the model is the car number 497 that can be seen in the frames number 16, 20, 24 and 28. The car 497 has been detected first in the frame 16 as it can be seen in Table 14 (the car in the top that is surrounded by a purple box). In the frame 20, a new car has been detected and associated to the tracker number 499, but in that frame the 497 tracker (the tracker associated to the target of the car detected in the previous frame) has not been associated to any car. In the frames 24 and 28 the number 497 has been associated to the same car previously detected in the frame 16 so the SORT algorithm is working well. In the next frames that appear in Table 14 it is very visual to see how the car number 502 and the 507 are tracked by the model. In the frames of the third row (164, 168, 172) appear frames where a red car is associated to the tracking number 502 and the SORT tracker can track the car properly. In addition, in the fourth row we can see the frames 204, 208, 212 that show how a new white car with tracking number 507 is being tracked.

## Co-driver's seat video

This video is much shorter than the other one because it contains more vehicles and all the examinations for this video will be done by detecting and tracking every 4 frames. The video resolution is 1920 x 1080 and has been reduced to 800 x 600 by default. When making previous tests we have seen that vehicles that are very far away and are very difficult to see with the human eye, the model is able to detect them if the confidence threshold is as low as the one used in the video before. We have decided to increase the confidence threshold to 0.7 in order to focus the model to detect and track vehicles that are clearly visible. The interest area on this video is different to the one analysed before, we want to focus on the vehicles that are located on the left side of the frame. For this case the interest area will be reduced to a dimension of 400 width and 350 of height (width coordinates between 25 and 425 and height coordinates between 250 and 600) as it can be seen in Figure 54.



*Figure 54: Example of the interest area.*

The video lasts 1.44 seconds and in the region of interest we can observe that they appear three different cars, one black car that goes a few meters ahead in the same direction of the recorder, other white car that is coming in the opposite direction and a final white car which is parked on the curb of the other direction. The Table 15 shows how the model analyses frames every 4 frames and what tracking numbers gives to each vehicle detected, each row represents the detection in a frame multiple of 4 and each image represents the detection of one car, the last image of the row contains all the vehicles tracked. There are some rows with two or more different frames, but it is because all that frames detect the same vehicle with the same tracking number.

The model has analysed 43 frames at a speed of 0.025 s/frame, that give a total time to process the video of 1.093 seconds. The processing time takes less time than the duration of the video so we are obtaining a model that can detect and track vehicles in real time. The speed of the model has been proved to be very fast and the accuracy because in this case we can observe three different cars and the model detects and tracks three cars with different tracking numbers of 65, 66 and 67.

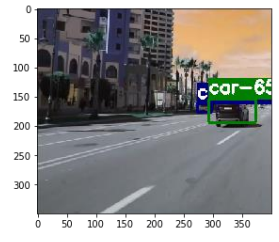
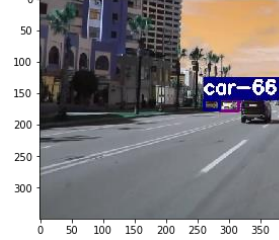

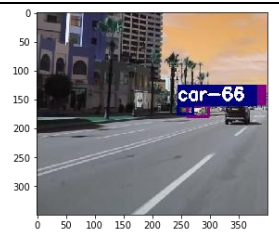
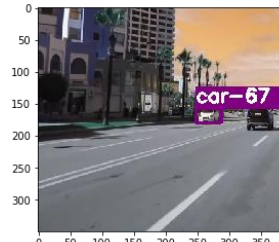
Frame Number	Car Number detected	Car Number detected	Car Number detected
0	 Car 66 (blue box)	 Car 65 (green box)	
4,8,12	 Car 66 (blue box)	 Car 65 (green box)	
16	 Car 67 (purple box)	 Car 66 (blue box)	 Car 65 (green box)
20,24	 Car 66 (blue box)	 Car 65 (green box)	
28	 Car 67 (purple box)	 Car 66 (blue box)	 Car 65 (green box)
32,36,40	 Car 67 (purple box)	 Car 65 (green box)	

Table 15: Example of detection and tracking in a video.

## 6. CONCLUSION AND FUTURE LINES

After having carried out this study, we have reached the following conclusions. Every training is unique and must be evaluated and tested in order to obtain a good detecting and tracking model. YOLOV3 is the third version of the YOLO object detection model and the one that has been used all through this work to detect vehicles. The most important thing when training YOLOV3 is to obtain a wide dataset of vehicles in all the type of positions, point of views, image resolution, forms, brands, etc. By this way, the model learns the concept of vehicle in all its forms that would appear in real life and therefore it is going to generalize well the concept of vehicle and is going to be able to detect vehicles in new images that have not been used to train the model. It is better to train the network in the first steps with low resolution images (416 x 416) and then picking randomly network resolution values and changing the network configuration in order to obtain a model capable of detecting vehicles in images of high and low resolution.

It is preferable to train YOLOV3 with anchor boxes obtained from the dataset because the prediction accuracy has been increased. An attempt has been made to reduce the training time by using initial weights obtained from the pre-trained weights for the COCO and ImageNet dataset, but this has meant a worsening in all areas. We have made some attempts to see if the network can be trained with a lower resolution than 416 or with a subdivision value lower than 16 but the network fails in the training. Finally, after analysing all the trainings we have concluded that with the last training we get the best values according accuracy, recall and mAP@0.5. This training has been done with the lowest value the training accepts for the subdivision parameter and the best data augmentation parameters.

The most important thing about a vehicle tracking system is the capability the model has to detect vehicles because if it is not able to detect some vehicles it will fail to track them. Our system is very accurate at detecting vehicles and that is why we have obtained so good results in the tracking process. The model that contains the YOLOV3 object detector and the SORT tracker must be adjusted to the area that the camera is recording. For example, a model that detects vehicles very well on a highway will not work so well in a camera installed on a street of a city because the interest point will have changed and we will have to modify some parameters of the model to suit it to the interests of the recorded area. That is what has happened with the highway video and the co-driver's seat video, the model for the first video must be adjusted with a lower confidence threshold than for the second one and also, the interest area is different in both videos so the adjustment of the frame is different. In short, the model once installed in a camera must be configured at the beginning depending on the background and the interest point and once we have obtained the parameters where the accuracy and speed are the best it will work alone.

This study could be continued for example by adapting the model to a camera that is installed in a drone. The model has to be trained with images of vehicles taken from the top in order to make a model that is able to detect vehicles from a drone that is flying in the sky. The tracking algorithm can be improved by implementing Deep SORT tracking algorithm that is an improved version of the SORT tracking algorithm that has been studied and implemented in this work. It would also have to be tested in other scenarios and implemented in true real-time.

## Figure references

Figure 1:

<https://i.stack.imgur.com/ibYr3.png>

Figure 2:

<https://i.imgur.com/McMOhuQ.png>

Figure 3:

<https://www.michaelchimenti.com/wp-content/uploads/2017/11/Deep-Neural-Network-What-is-Deep-Learning-Edureka.png>

Figure 4:

[https://miro.medium.com/max/1255/1\\*vkQ0hXDaQv57sALXAJguxA.jpeg](https://miro.medium.com/max/1255/1*vkQ0hXDaQv57sALXAJguxA.jpeg)

Figure 5:

[https://miro.medium.com/max/526/1\\*Gcl7G-JLAQjEoCON7xFbhg.gif](https://miro.medium.com/max/526/1*Gcl7G-JLAQjEoCON7xFbhg.gif)

<https://i1.wp.com/timdettmers.com/wp-content/uploads/2015/03/convolution.png?resize=500%2C193>

Figure 6:

<https://i.imgur.com/EolgnXi.jpgg>

Figure 7:

[https://media.springernature.com/original/springer-static/image/art%3A10.1007%2Fs00521-019-04296-5/MediaObjects/521\\_2019\\_4296\\_Fig3\\_HTML.png](https://media.springernature.com/original/springer-static/image/art%3A10.1007%2Fs00521-019-04296-5/MediaObjects/521_2019_4296_Fig3_HTML.png)

Figure 8:

[https://miro.medium.com/max/470/1\\*81Y95NKxLzXLEut7nepmZA.png](https://miro.medium.com/max/470/1*81Y95NKxLzXLEut7nepmZA.png)

Figure 9:

<https://www.jeremyjordan.me/content/images/2018/02/Screen-Shot-2018-02-24-at-11.47.09-AM.png>

Figure 10:

<https://www.learnopencv.com/wp-content/uploads/2018/07/ModelAccuracy.png>

Figure 11: Figure made by me

C:\Users\Urtzi Sotes\Desktop\fotos tf\object detection.png

Figure 12:

<https://machinethink.net/images/yolo/Grid.png>

Figure 13:

<https://machinethink.net/images/yolo/Boxes.png>

Figure 14:

[https://hackernoon.com/hn-images/1\\*oXSVPOHPVlaZqPpSinxsRQ.png](https://hackernoon.com/hn-images/1*oXSVPOHPVlaZqPpSinxsRQ.png)

Figure 15:

[https://miro.medium.com/max/1624/1\\*0IPktA65WxOBfP\\_ULQWcmw.png](https://miro.medium.com/max/1624/1*0IPktA65WxOBfP_ULQWcmw.png)

Figure 16:

<https://machinethink.net/images/yolo/Scores.png>

Figure 17:

<https://machinethink.net/images/yolo/Prediction.png>

Figure 18:

[https://www.pyimagesearch.com/wp-content/uploads/2016/09/iou\\_equation.png](https://www.pyimagesearch.com/wp-content/uploads/2016/09/iou_equation.png)

Figure 19:

[https://www.pyimagesearch.com/wp-content/uploads/2016/09/iou\\_examples.png](https://www.pyimagesearch.com/wp-content/uploads/2016/09/iou_examples.png)

Figure 20:

[https://manalelaidouni.github.io/assets/img/pexels/YOLO\\_arch.png](https://manalelaidouni.github.io/assets/img/pexels/YOLO_arch.png)

Figure 21:

[https://hackernoon.com/hn-images/1\\*rDTomC-BsvHUqPUzAV8XKg.png](https://hackernoon.com/hn-images/1*rDTomC-BsvHUqPUzAV8XKg.png)

Figure 22:

[https://hackernoon.com/hn-images/1\\*wY4F7t3gkf4f0Xjhgo23Pg.png](https://hackernoon.com/hn-images/1*wY4F7t3gkf4f0Xjhgo23Pg.png)

Figure 23:

[https://hackernoon.com/hn-images/1\\*MxoBslVDuvxkX\\_gl0B2xHQ.png](https://hackernoon.com/hn-images/1*MxoBslVDuvxkX_gl0B2xHQ.png)

Figure 24:

[https://hackernoon.com/hn-images/1\\*pT50zzLRyJm6CTuGmoYBxA.png](https://hackernoon.com/hn-images/1*pT50zzLRyJm6CTuGmoYBxA.png)

Figure 25:

[https://miro.medium.com/max/509/1\\*smK6Jgarqw09nA7vnQexAg.png](https://miro.medium.com/max/509/1*smK6Jgarqw09nA7vnQexAg.png)

Figure 26:

[https://miro.medium.com/max/822/1\\*89qezEeLKJLpD8\\_fm\\_H4qQ.jpeg](https://miro.medium.com/max/822/1*89qezEeLKJLpD8_fm_H4qQ.jpeg)

Figure 27:

[https://cdn-images-1.medium.com/max/1000/1\\*38-Tdx-wQA7c3TX5hdnwpw.jpeg](https://cdn-images-1.medium.com/max/1000/1*38-Tdx-wQA7c3TX5hdnwpw.jpeg)

Figure 28:

<https://manalelaidouni.github.io/assets/img/pexels/Direct%20location%20prediction.png>

Figure 29:

[https://miro.medium.com/max/334/1\\*biRYJyCSv-UTbTQTa4Afgg.png](https://miro.medium.com/max/334/1*biRYJyCSv-UTbTQTa4Afgg.png)

Figure 30:

[https://miro.medium.com/max/614/1\\*FI-TbhgQBvfZ9x-8BV77CA.png](https://miro.medium.com/max/614/1*FI-TbhgQBvfZ9x-8BV77CA.png)

Figure 31:

[https://miro.medium.com/max/1380/1\\*D\\_EAjMnIR9v4LqHhEYZJLg.png](https://miro.medium.com/max/1380/1*D_EAjMnIR9v4LqHhEYZJLg.png)

Figure 32:

[https://wikimedia.org/api/rest\\_v1/media/math/render/svg/544734ddf1cab7b0a1397d852c7aa59ebd220319](https://wikimedia.org/api/rest_v1/media/math/render/svg/544734ddf1cab7b0a1397d852c7aa59ebd220319)

Figure 33:

[https://wikimedia.org/api/rest\\_v1/media/math/render/svg/ece203d4105b4ba16b24c8281964fe0a4844961e](https://wikimedia.org/api/rest_v1/media/math/render/svg/ece203d4105b4ba16b24c8281964fe0a4844961e)

Figure 34:

<https://i.stack.imgur.com/zS2OB.png>

Figure 35:

<https://image.slidesharecdn.com/103001cse820p1-160606065212/95/multi-object-tracking-presentation-1-id-103001-16-638.jpg?cb=1465196340>

Figure 36:

<https://www.move-lab.com/media/pages/blog/tracking-things-in-object-detection-videos/3750104712-1581072269/iou-tracker.png>



## References

- [1] Y. Kim and H. Bang, "Introduction to Kalman Filter and Its Applications," in *Introduction and Implementations of the Kalman Filter*, IntechOpen, 2019.
- [2] "Tracking multiple objects with OpenCV - PyImageSearch." [Online]. Available: <https://www.pyimagesearch.com/2018/08/06/tracking-multiple-objects-with-opencv/>. [Accessed: 19-Feb-2020].
- [3] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, 2019, doi: 10.1016/j.neucom.2018.09.038.
- [4] "Object detection with deep learning and OpenCV - PyImageSearch." [Online]. Available: <https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/>. [Accessed: 19-Feb-2020].
- [5] "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way." [Online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. [Accessed: 13-Feb-2020].
- [6] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," *Proc. - Int. Conf. Image Process. ICIP*, vol. 2016-Augus, pp. 3464–3468, 2016, doi: 10.1109/ICIP.2016.7533003.
- [7] "Understanding YOLO - By." [Online]. Available: <https://hackernoon.com/understanding-yolo-f5a74bbc7967>. [Accessed: 06-Feb-2020].
- [8] "Understanding YOLO and YOLOv2 | Manal El Aidouni." [Online]. Available: [https://manalelaidouni.github.io/manalelaidouni.github.io/Understanding YOLO and YOLOv2.html](https://manalelaidouni.github.io/manalelaidouni.github.io/Understanding%20YOLO%20and%20YOLOv2.html). [Accessed: 06-Feb-2020].
- [9] J. Redmon and A. Farhadi, "YOLO v.3," *Tech Rep.*, pp. 1–6, 2018.
- [10] R. E. Burkard, M. Dell'Amico, S. Martello, and Society for Industrial and Applied Mathematics., *Assignment problems*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009.
- [11] "Hungarian algorithm - Wikipedia." [Online]. Available: [https://en.wikipedia.org/wiki/Hungarian\\_algorithm](https://en.wikipedia.org/wiki/Hungarian_algorithm). [Accessed: 19-Feb-2020].
- [12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.