FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Feature Extraction and Object Classification in Video Sequences for Military Surveillance

Pedro José Ascensão Ramalho

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador(a): Prof.Maria Teresa Andrade

Co-Orientador(a): Eng.Luís Lucas

2018/2019

Resumo

Embora já existam muitos modelos que realizam a detecção e reconhecimento de muitos objetos ou conceitos diferentes com graus razoáveis de confiança e taxas de sucesso, um dos objetivos desta dissertação é desenvolver um sistema altamente especializado e eficiente para identificar um grupo limitado e particular de objetos. Isto será alcançado usando *transfer learning*, que é um processo que usa o conhecimento adquirido por um desses modelos enquanto resolve um problema (isto é, enquanto reconhece um conjunto de conceitos) e o aplica a uma questão diferente. Basicamente, ele tira proveito das saídas do processo de extração de características e utiliza-as para aprender a identificar outro tipo de objetos.

A deteção e reconhecimento automático de objetos em dados visuais pode ser alcançado recorrendo a um sistema de aprendizagem que analisa e processa informações visuais e identifica automaticamente um grupo de objetos, independentemente dos dados de entrada. Para poder realizar este tipo de identificação, este sistema precisa analisar previamente um grande conjunto de dados, para que ele possa memorizar características específicas de diferentes objetos.

Este processo é a chamada fase de treino e é o primeiro passo em todos os processos de detecção e reconhecimento de *machine learning*.

A extração de características é um processo aplicado aos dados de entrada com o objetivo de reduzir o volume de dados a serem processados pelo modelo, criando, desta foram, menores conjuntos de informações não redundantes. Essencialmente, o volume de dados é reduzido e as redundâncias que normalmente existem em dados visuais são eliminadas. Estes conjuntos de dados, são mais fáceis de gerir e ainda fornecem uma descrição completa do grupo de dados original. Desta forma, os recursos necessários, tanto durante a fase de aprendizagem como durante o processo de reconhecimento, podem ser reduzidos.

Neste contexto, os dados a serem analisados serão capturados por uma câmara implementada num ponto estacionário ou num veículo. Ao lidar com a captura de informações visuais, é normal que o resultado seja um grande número de dados. Por isso, é importante analisá-lo com eficiência e obter uma identificação de informações relevantes. Esta dissertação foca-se em usos militares, pois estas operações serão usadas para identificar automaticamente objetos no campo militar, isto é, tanques, armas, pessoas e veículos, alcançando vigilância territorial.

Abstract

Although there are already many models that perform detection and recognition of many different objects or concepts with reasonable degrees of confidence and success rates, one of the goals of this dissertation is to develop a highly specialized and efficient system to identify a limited and particular group of objects. This will be achieved by using transfer learning, that is a process that uses the knowledge gained by one of these models while solving one problem (i.e., while recognizing a set of concepts) and applies it to a different one. Basically, it takes advantage of the feature extraction procedure outputs and use them to learn how to identify other kind of objects.

Automatic object detection and recognition in visual signals can be achieved by resorting to a learning system that analyses and processes the visual data and automatically identifies a group of objects independently of the input data. To be able to perform this kind of identification, this system needs to previously analyze a large set/corpus of data, so it can memorize special features of different objects. This procedure it is the so-called training phase and it is the first step in all the detection and recognition processes of machine learning.

Feature extraction is a process applied to the visual input data, with the goal of reducing the volume of data to be processed by the machine learning model, by creating smaller sets of non-redundant information. It essentially reduces the volume of data by eliminating redundancies that typically exist in visual signals. These smaller groups are more manageable and provide a full description of the original data set. This way, the resources necessary both during the learning phase as well as during the actual recognition process, can be greatly decreased.

In this context, the data to be analyzed will be captured by a camera implemented at a stationary point or in a vehicle. When dealing with the capture of visual information, it is normal that a large number of data is generated. So, it is important to analyze it efficiently and achieve relevant information identification. This dissertation focuses in military uses, therefore these operations are going to be used to automatically identify objects in the military field, that is, tanks, guns, people and vehicles (cars and trucks), achieving territorial surveillance.

Agradecimentos

Aos meus orientadores, Professora Maria Teresa Andrade e Engenheiro Luís Lucas, por toda a disponibilidade e conhecimento que me trasmitiram.

A todos os meus amigos, por todos as vitórias, derrotas e empates que me proporcionaram. Em especial ao João, Baltasar, Tiago e Henrique, que desde cedo acompanharam todo este meu percurso. Agradeço ao António por me ter acompanhado em tantas batalhas, as quais nunca teria vencido sem ele.

À Critical Software pela oportunidade que me dispuseram e por todas as condições que me foram fornecidas para a elaboração desta dissertação.

À minha familía que me ensinou que se acordar a sorrir, os dias tornam-se mais fáceis.

Pedro Ramalho

“The true sign of intelligence is not knowledge but imagination.”

Albert Einstein

Contents

List of Figures

List of Tables

Abreviaturas e Símbolos

AI	Artifical Inteligence
ANN	Artifical Neural Network
API	Application Programming Interface
AP	Average Precision
BLOB	Binary Large Object
CNNs	Convolutional Neural Networks
CPU	Central Processing Unit
CSV	Comma Separated Values
FEUP	Faculdade de Engenharia da Universidade do Porto
GPS	Global Posiontining System
IoU	Intersection over Union
k-NN	k-nearest neighbour
ML	Machine Learning
NN	Neural Network
ReLU	Rectified Linear Unit
ROI	Region Of Interest
RPN	Region Proposal Network
SIFT	Scale-invariant feature transform
SURF	Speeded-Up Robust Features
SSD	Single Shot Detector
UAV	Unmanned Aerial Vehicle

Chapter 1

Introduction

Features extraction is a set of processes with the goal of simplifying big groups of data by creating small groups of non-redundant information. These small groups are more manageable and can fully describe the original data set. Basically, by using features extraction processes or algorithms, is possible to decrease the resources necessary to analyze a large group of data.

Nowadays, machine learning models use feature extraction processes, in order to perform object detection and recognition. Through deep learning, artificial neural networks adapt and learn from large amounts of data. Bigger the amount, higher the classification accuracy of a model.

Building a feature extractor system from scratch can be a difficult task. In order to avoid this labour, machine learning models applied to object detection, can be used. Since the main focus is military surveillance, it is necessary to detect some specific objects in this field (tanks and guns). In order to achieve this, techniques such as transfer learning are used, where pre-trained models are used as the starting point in a training procedure. Basically, pre-trained models are adapted to perform specific tasks.

This dissertation will be developed in partnership with Critical Software as an internship in their facilities.

1.1 Context

In this context, it is fundamental to develop a study about techniques of features extraction and machine learning models that perform automatic object detection.

Machine learning is at the forefront of the nowadays technology so, it is important for Critical Software to keep up with all the possible applications of this area.

There are a large number of examples that can be studied and this dissertation aims to analyze their behaviour while performing feature extraction and object identification, concluding which ones, achieve it in a more efficient way.

1.2 Motivation

In the past few years, the scientific community has been increasingly dedicating their time to the development of visual information processing algorithms leading to the existence, at present times, of an extended group of tools and approaches that allow to extract a wide range of features. Likewise, in the last years, there have been significant breakthroughs in techniques of artificial intelligence and their application in features processing with the intent of performing content classification and identification.

When the focus is to capture visual information, it's normal that a large number of data, with various levels of quality, is generated. So, it would be important to eliminate, in a efficient way, all the unnecessary information that overloads the user. It's possible to achieve this by doing an automatic analysis that can identify relevant information, eliminating the need of storing redundant data.

1.3 Goals

Regarding this dissertation, the main goal is to develop a software library that is capable of performing features extraction, in an automatic way, on a set of videos or images. The visual data will be captured by a camera implemented at a stationary point or in a vehicle. The library will be implemented and perform automatic analysis to the visual information being captured. This analysis, in addition to features extraction, will attempt to identify and detect objects.

Chapter 2

State of the Art

This chapter provides information related to techniques for feature extraction, machine learning and object detection methods. Initially, a small introduction about image feature extraction and processing is given in section 2.1. The following sections analyze algorithms that have a good performance when dealing with features extraction.

In section 2.3, an overview to SIFT is given, the stages of this technique are presented and the operation is explained. Subsections 2.3.1 and 2.3.2, explain the stages of this technique in a specific way.

In section 2.4, SURF technique is analyzed. From subsection 2.4.1 to subsection 2.4.3, the main stages of this algorithm are explained.

Section 2.2 gives an introduction to the feature space analysis and, the following section 2.5, analyzes a technique that can be used for object recognition and tracking.

Section 2.6, divides machine learning overview into some topics. It is possible to find specific details about types of learning, artificial neural networks and object detection methods.

Finally, section 2.7 represents what was concluded from this topics, what technique/method is going to be used and why.

2.1 Image Feature Extraction and Processing

Image feature extraction techniques and learning based on features, are commonly used by computer vision applications. This applications are able too automatically extract features and make predictions based on them. Usually, this systems are composed by artificial neural networks that analyze large amounts of data and perform three main tasks: Extraction, Selection and Classification.

The extraction task generates features from images content that are selected by the selection task, in order to achieve a specific goal. In this way, the number of features provided to the classification task are reduced since, feature that were not selected, are discarded. Since the efficiency of the classification task depends on particular available features, the extraction task is the most critical [20].

In Figure 2.1, it is possible to observe an example of feature extraction and image processing techniques applied to object detection and segmentation.

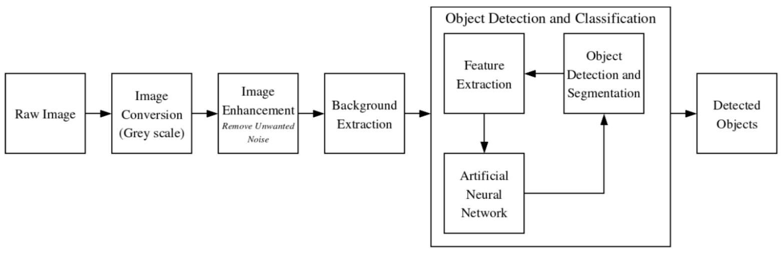

Raw Image
Image Conversion (Grey scale)
Image Enhancement Remove Unwanted Noise
Background Extraction
Object Detection and Classification
Feature Extraction
Object Detection and Segmentation
Artificial Neural Network
Detected Objects

Figure 2.1: Feature extraction and Image processing, adapted from [1].

2.2 Feature Space Analysis

An image feature space is a high dimensional set of values that have been obtained from applying processing techniques to images. Such high-dimensional set may contain a variable number of values associated to different features of the image, such as color, average pixel intensity or gradient magnitude. Normally, the features that are included in the space depend on the application/objective in view. For example, in the real-world, a geographical location can be characterized by a pair of GPS coordinates (N-S,E-W) in an two-dimensional linear space. If the objective is to compare locations, a two-dimensional vectors will be used for each location. Now suppose that is also necessary to compare the altitudes of the geographical locations. In this case, a three-dimensional space would be built, with three values for each location (N-S, E-W, A). Then the comparison would be done using three-dimensional vectors. So, if in image object detection there are objects that have 5 features, then a 5-dimensional feature space would be used. Normally, the values of the features included in such high-dimensional space of an image, are in a much lesser number than the number of pixels and their base grey-level scale intensity feature.

The global nature of the derived representation of the input, brings advantages and disadvantages into the feature space paradigm. All the evidence of the presence of a significant feature is pooled together, causing excellent tolerance to a noise level which may render local decisions unreliable. On the other hand, in spite of being salient for the tasks, features with lesser support may not be detected in the feature space. This can be avoided by either augmenting the feature space with additional parameters from the input domain or by robust post-processing of the input domain leaded by the results of the feature space analysis.

In Figure 2.2, an example of a feature space is shown. It is possible to observe that the color image is mapped into a three-dimensional color space. Also, a continuous transition between the clusters arising from the dominant colors is seen and severe artifacts are introduced by the decomposition of the space into elliptical tiles.

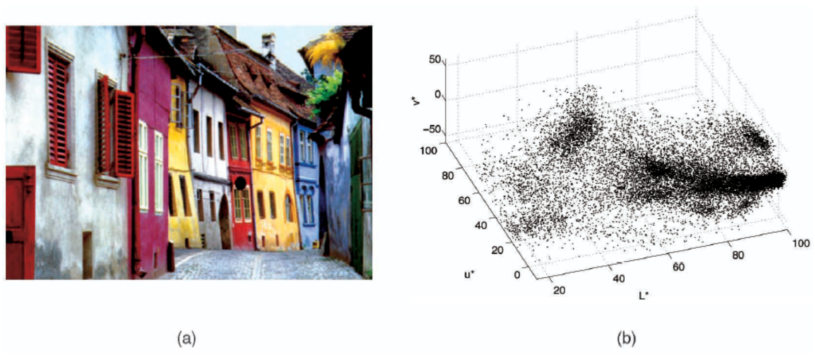


Figure 2.2: In (a) a color image is shown and in (b) the corresponding color space, adapted from [2].

Feature spaces that are arbitrarily structured are only analyzed by non-parametric methods, once these methods do not have embedded assumptions. Non-parametric clustering methods can be reunited into two classes: hierarchical clustering and density estimation.

The first class, either aggregate or divide the data based on some proximity measure. This kind of methods tend to be computationally expensive and the definition of a stopping criterion for the fusion of the data is not simple. In the density estimation clustering methods, the feature space is interpreted as the empirical probability density function of the represented parameter. The local maxima of this function represents the dense regions in the feature space or modes of unknown density [2]. Once a mode is located, the cluster associated with it is defined based on the local structure of the feature space [21].

One approach to dense regions detection and clustering, is the mean shift method proposed by [22] and reinforced, later, by [23].

2.3 SIFT

SIFT (Scale-invariant feature transform) is a computer vision algorithm used for performing features detection and description. Image data is transformed into scale-invariant coordinates relative to local features [3].

SIFT is divided in four major stages [3]:

1. Detection of Space-scale extrema.
2. Key-point localization.
3. Orientation Assignment.
4. Key-point descriptor.

The detection is implemented efficiently using a difference-of-Gaussian function and the description uses gradient oriented histograms to describe the region around the interest point. Various

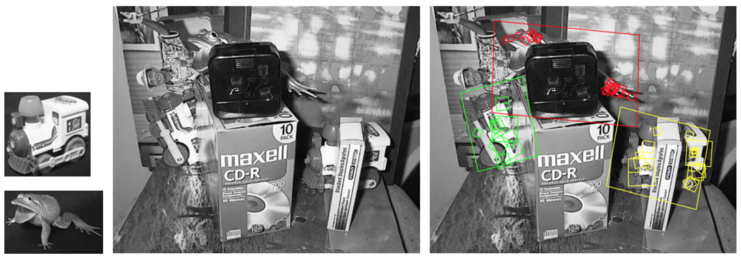


Figure 2.3: Object recognition using SIFT algorithm, adapted from [3]

key-points are detected in an image and their descriptors are determined. Then, this group of descriptors can be used to match different images.

SIFT has the advantage of being capable to generate a large number of features, covering the full image in terms of location and scales. From a typical image it is possible to extract a wide range of stable features. Then, these features are stored in a database to be used in future comparisons with features extracted from a new input image. The comparisons perform feature matching using the Euclidean distance between their feature vectors [3].

The quantity of features extracted is a very important aspect in object recognition, since ,in some cases, it is required that at least 3 or more features are accurately matched, between objects, to have a well grounded identification. A single feature has a high probability to find its correct match in a large database, due to the high distinctiveness of the key-point descriptors [3]. On another hand, various features from the background will not have a correct match in the database causing the growth of false matches. Through a series of tests and computations, it is possible to filter, from the full set of matches, the ones that are correct. In Figure 2.3, it is possible to observe the use of different key-points from the object images, on the left, to identify them on the right figure.

2.3.1 Detection of Space-scale extrema & Key-point localization

This operation is performed through an approach based on cascade filtering to identify candidate locations to be analyzed in further detail [3]. The first step is to locate points that are invariant to scale changes under different views of the same object. This is achieved by searching for stable features in all possible scales through a continuous function of scale known as scale space [24]. Under these conditions, the only scale-space possible is the Gaussian function, as stated in [25, 26]. So, the calculated scale space is given by:

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y), \tag{2.1}$$

a convolution (*) between a variable-scale Gaussian, $G(x,y,\sigma)$, and an input image $I(x,y)$, where

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}. \tag{2.2}$$

A stable detection of key-points locations is accomplished by using scale-space extrema in the difference-of-Gaussian function convolved with the input image [27], $D(x,y,\sigma)$, that can be calculated from the difference between two nearby scales separated by a constant factor k:

$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma). \tag{2.3}$$

In Figure 2.4, it is shown an efficient way to build $D(x,y,\sigma)$. The input image gets convolved with Gaussians to create a group of images that are separated by a constant k in scale-space. Adjacent scaled images are subtracted to produce the difference-of-Gaussian images.

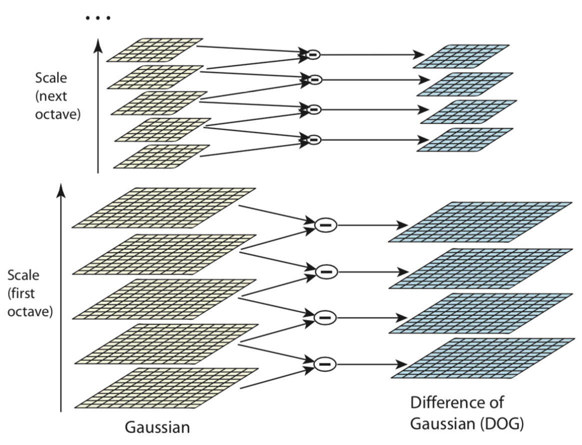


Figure 2.4: Difference-of-Gaussian representation, adapted from [3]

The local maximum or minimum of $D(x,y,\sigma)$, can be detected by comparing the intensity of each point with the intensity of its neighbours in the current image and with the neighbours in the scale above and below. Then, it is selected if it is smaller or larger than all the other neighbours. In Figure 2.5, it is possible to observe this detection process.

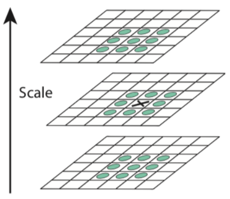


Figure 2.5: Detection of the local maximum and minimum of $D(x,y,\sigma)$,adapted from [3]

2.3.2 Orientation Assignment & Key-point descriptor

Each key-point has a orientation assigned in order to build their descriptors. A orientation histogram is created using the gradient orientations from points within a nearby region around the key-point. Each point of that region is weighted and added to the histogram with a certain value. The peaks in the histogram represent the dominant directions of the local regions gradients and are used to define the key-point orientation. Then, the key-point descriptor is determined using the magnitudes and orientations gradient from the regions around the key-point. In Figure 2.6, it is possible to observe that the key-point descriptor generated (right side), has four orientation histograms, each one with eight directions represented by arrows.

The determined features vectors alongside the technique k-NN (k-nearest neighbour) [28], can be used to perform object recognition in an image.

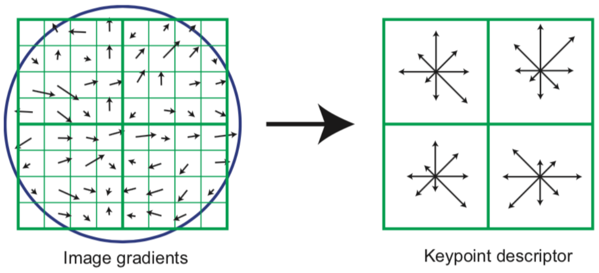


Figure 2.6: Creation of the Key-point descriptor, adapted from [3].

2.4 SURF

SURF (Speeded-Up Robust Features) is an algorithm that detects interest points in different locations of the image. These interest points are invariant to scale and rotation operations.

The detector is based on the Hessian matrix but uses the determinant of this matrix to select the scale and the location, instead of using a measure, like in a Hessian-Laplace [4]. The descriptor describes the distribution of responses to Haar-wavelet in a nearby region of the interest point.

2.4.1 SURF Detector

SURF detector, in terms of object recognition, uses the interest points to perform the identification, instead of searching for an object as a whole. This approach has different advantages such as the computational cost for large data and high levels of redundancy. These levels of redundancy outcome from independence of the pixels and from their correlation degree.

There are different methods that can be use to perform the detection and definition of the interest points.

2.4.2 Fast-Hessian Detector

This detector is based on the Hessian matrix since it shows a good performance in terms of computation time and accuracy. Considering a point x=(x,y) in an image I, the Hessian matrix for that point at scale σ is determined by [4]

$$\mathscr{H}(\mathrm{x},y) = \begin{bmatrix} L_{xx}(\mathrm{x},\sigma) & L_{xy}(\mathrm{x},\sigma) \\ L_{xy}(\mathrm{x},\sigma) & L_{yy}(\mathrm{x},\sigma) \end{bmatrix}, \tag{2.4}$$

$L_{xy}(\mathrm{x},\sigma)$ represents the convolution of the Gaussian second order derivative with the image I in point x (similar to $L_{xx}(\mathrm{x},\sigma)$ and $L_{yy}(\mathrm{x},\sigma)$).

As stated in [25, 26], Gaussians are the optimal approach to perform scale-space analysis. However, in practical cases, the Gaussian needs to be cropped and discretised leading to, as a consequence, a loss of repeatability over the rotations of the image. This is one the reasons why the Hessian matrix is chosen because of its high repeatability rate for each rotation angle [29]. Since Gaussian filters are not ideal to use in any case, they are approximated by box filters 2.7. Thus, the calculations can be performed with constant time, while using a group of integral images.

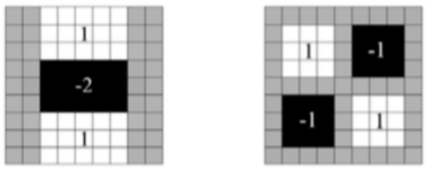


Figure 2.7: Approximations using box filters, adapted from [4].

To implement scale spaces, usually image pyramids are used. To achieve a higher level of the pyramid, images are repeatedly smoothed with a Gaussian and sub-sampled. With the use of box filters, there is no need to iteratively filter the image, instead it is possible to apply such filters with different sizes at the same speed, directly on the original image. Higher layers of the pyramid are reached through the application of gradually bigger filters. A scale space is divided in octaves and each one of them is composed by the responses to the filter, when the input image is convolved with a filter of growing size.

A non-maximum supression in a $(3 \times 3 \times 3)$ neighbourhood is applied to locate interest points in the image and over the scales. Using the method proposed in [30], the maxima of the determinant of the Hessian matrix is interpolated in image scale and space.

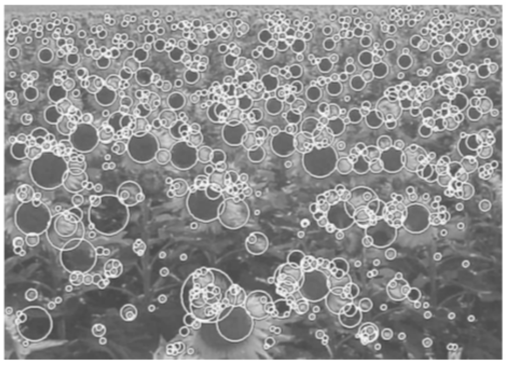


Figure 2.8: Detected Interest points using a Hessian based detector , adapted from [4].

2.4.3 SURF Descriptor

After the detection of the interest points the next stage is the description. This process initiates with the attribution of one orientation to the point, so it can be distinguishable in any perspective and invariant to rotation. For this, the Haar-wavelet responses are calculated in a x and y direction in a circular neighbourhood that has $6s$ radius around the interest point, where s is the scale of the interest point when it was detected. *Wavelet* responses in directions x and y with a sampling step equal to s and wavelets with side length equal to $4s$, are used.

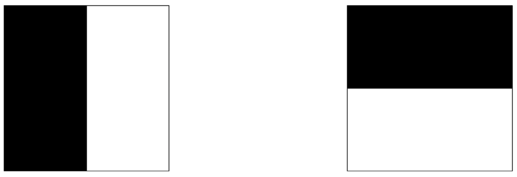


Figure 2.9: Haar wavelet types , adapted from [4].

Through the calculation of the sum of all the responses within a sliding orientation window, the dominant orientation is estimated. Vertical and Horizontal responses of the sliding window are summed, creating a vector. The longest vector determines the orientation of the interest point.

Then, a square region is centered around the interest point, with the orientation determined previously and a size of $20s$. This region is split up into (4×4) square sub-regions. In this way, important spatial information is kept. Through a Haar filter of size $2s$, features at (5×5) regularly spaced sample points are computed in horizontal and vertical directions. The responses to this filter are determined in the image without rotation and then they are interpolated. Using a Gaussian ($\sigma = 3.3s$) centered at the interest point, responses d_x and d_y are weighted. Then, the responses are summed over each sub-region and form the first entries to the feature vector. Absolute values of the responses are also computed, $|d_x|$ and $|d_y|$, to bring in information about the polarity of the intensity changes. The result is a descriptor vector for all (4×4) sub-regions with a length of 64.

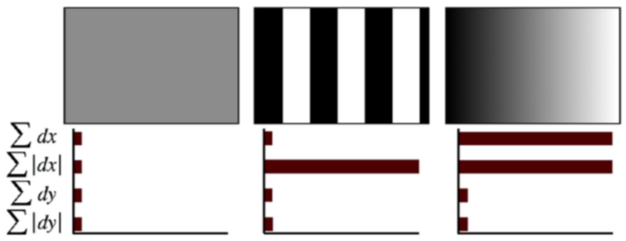


Figure 2.10: Descriptor entries of a sub-region, adapted from [4].

SURF is less vulnerable to noise due to obtaining information through sub-regions gradient, instead of using individual gradients. The processing time is reduced due to the use of 64 dimensions the decreases the computational effort.

2.5 Mean Shift

Mean Shift is an algorithm for identification and tracking of objects. This algorithm moves, in an iterative way, a window to the location of the image where it's possible to reach the maximum points distribution,that is, the area with the higher density of points. Given n data points x_i in the n-dimensional space R^n, the kernel function $K(x)$ indicates how much x contributes for the average estimation. For this method, the radially symmetric kernels that satisfy [2]

$$K(x) = c_{k,d}k(\| x \|^2), \tag{2.5}$$

are often more suitable. To make this possible, the *profile* function $k(x)$, is defined, only, for $(x > 0)$. Also, the normalization constant $c_{k,d}$ is assumed strictly positive, making $K(x)$, the *d-variate* kernel, integrate to one. When only one bandwidth parameter is employed, it's possible to obtain the kernel density estimator using [2]

$$\tilde{f}_K = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - xi}{h}\right). \tag{2.6}$$

The average of samples $m(x)$ with kernel K at $x \in R^n$ is defined as [23]

$$m(x) = \frac{\sum_{i=1}^{n} K(x - xi)xi}{\sum_{i=1}^{n} K(x - xi)}. \tag{2.7}$$

The difference $m(x) - x$ it is called mean shift [22]. This operation moves the point iteratively to its average and stops when m(x) = x.

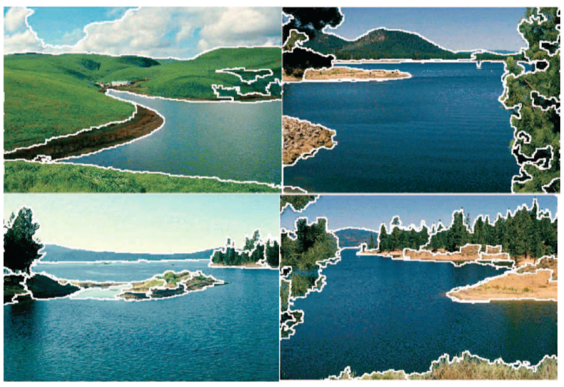


Figure 2.11: Image segmentation by using Mean Shift , adapted from [2].

2.6 Machine Learning

Machine learning (ML), normally, is strongly associated to the changes in systems that perform tasks related with artificial intelligence (AI). These tasks are performed without following any explicit instructions and can involve recognition, diagnosis, robot control, between others. Changes in these type of systems can be enhancements to the already performing structure or components of the beginning synthesis of a new system. In Figure 2.12, it is possible to observe an example of an AI system. Any changes that may occur to some components in Figure 2.12, might count as learning [5]. Basically, there's a model defined with some parameters and learning it is the optimization of these parameters by using training data or past experience. The model can be either predictive (predicts future actions) or descriptive (gains knowledge from data), it can also be both.

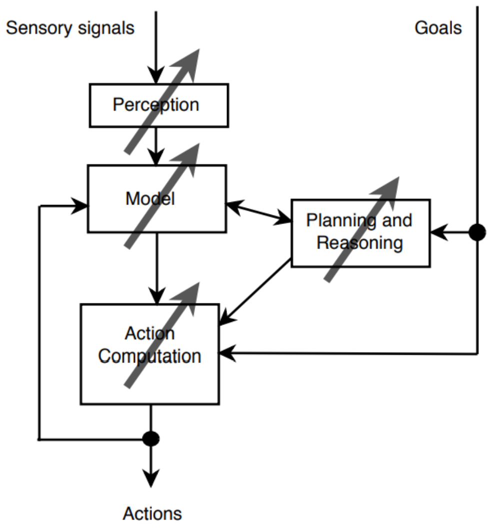


Figure 2.12: AI System, adapted from [5].

Through statistics, machine learning builds mathematical models, due to the core task making inference from a sample. There is two major situations where it is needed maximum efficiency. First, in training, since an optimization problem is solved and a large amount of data is processed and stored. Second, in the representation and algorithmic solution for inference, once a model is learned [31].

In certain applications, machine learning has an important role. For example, it can happen that hidden among large groups of data are important relationships and correlations. ML methods are used to extract these correlations/relationships, that's called *data mining*. Also, machine learning can provide adaptation to machines that need to keep up with the changes of an environment. Reducing, in this way, the need for a constant redesign [5].

2.6.1 Types of Learning

Different types of learning can outcome from various scenarios. They can occur due to the training data available to the learner, the order and the method that control the reception of the training data and the test data used to the evaluation of the learning algorithm.

Supervised Learning

In supervised learning, a mathematical model is built using a data set that contains inputs and outputs. Basically, there's a set of labeled examples as training data and the algorithm generates an inferred function capable of predicting outputs associated with new inputs. This prediction is possible due to an iterative optimization of an function, which can be either a loss function or the opposite (profit function). In a certain way, the environment is the teacher of the algorithm so, when it improves the accuracy of the predictions over time, it is possible to say that it learned to perform the task that was meant to develop.

Supervised learning issues can be grouped into Regression and Classification problems. The only difference between these two groups is that the dependent attribute is numerical for regression and categorical for classification [32, 33].

Unsupervised Learning

In unsupervised learning only unlabeled training data is received and, for all the unseen points, predictions are generated. The goal of this learning is to study how to learn to represent input patterns by reflecting the statistical structure, of the overall collection, of these patterns. It is difficult to quantitatively evaluate the execution of a learner due to the nonexistence of labeled examples in the setting. In comparison with supervised and reinforcement learning, explicit target outputs connected with each input, don't exist. Aspects of the structure of the input should be captured in the output.

Clustering and dimensional reduction are examples issues of unsupervised learning [32, 34].

Reinforcement Learning

In reinforcement learning, information is collected through an active interaction between the learner and the environment and, in some cases, the environment is affected by the learner and a reward is received for this action. The learner has the goal of maximize his reward through a set of actions and iterations with the environment. The exploration versus exploitation dilemma appears due to the fact of the environment doesn't provide a long-term reward feedback. So, the learner needs to choose between exploring unknown actions to obtain more information or exploiting the information already collected. Basically, reinforcement learning is the problem faced by an agent that must learn through trial and error interactions with a dynamic environment [32, 35].

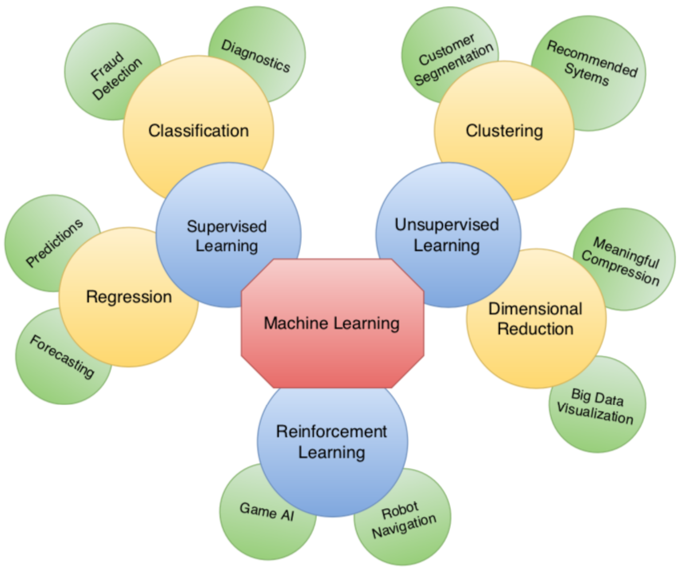


Figure 2.13: Machine Learning Algorithm Overview.

2.6.2 Feature Learning

Feature learning or Representation Learning is represented by learning representations of the data that makes easier the extraction of useful information when developing classifiers or predictors. Basically, is a set of techniques that are able to learn a feature by generating a representation, through transformation of raw input data, that can be effectively used to perform a specific task. Manual feature engineering is replaced and the machine is enabled to both learn a task (using the features) and learn the features themselves.

Normally, classification tasks in machine learning require an input that is computationally convenient to process. Real-world data such as images or videos, usually, are very complex and redundant so, it is necessary to uncover features or representations from raw data. Manual feature analysis requires expensive human labor, relies on expert knowledge and, in some cases, results in a bad generalization. All of these situations motivate the design of feature learning techniques that are capable of a efficient automation and generalization.

As shown in Figure 2.14, feature learning algorithms have an advantage because they learn representations that capture underlying factors, a subset of which may be relevant for each particular task. Also, generalization is improved since these subsets overlap and share statistical strength [6].

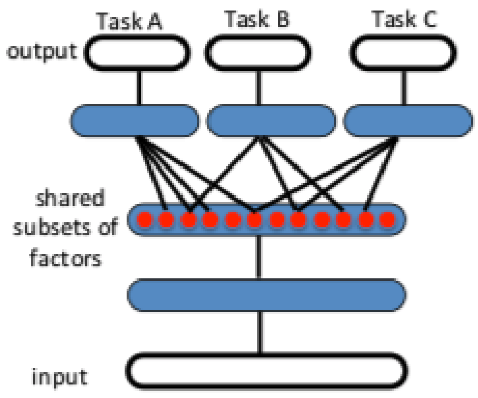


Figure 2.14: Representation Learning finding descriptive factors which explain the input and the target for each task, adapted from [6]

2.6.3 Decision Trees

Decision trees are a set of techniques for predicting and explaining the relationship between observations about an item and his target value. Derived originally from management, statistics and logic, decision trees, are used in areas such as data mining, pattern recognition, machine learning, between others.

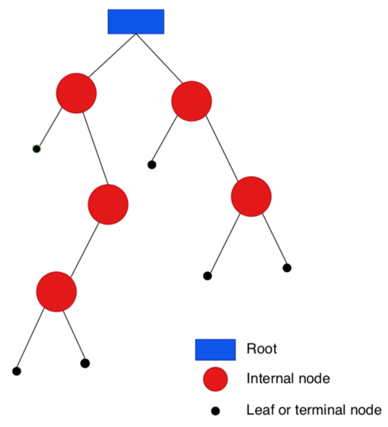
Decision trees have different advantages such as:

- Flexibility when handling a variety of input data;
- Non-parametric, a range of numeric or categorical data layers are easily incorporated;
- Very useful when dealing with large datasets;
- Easy to follow and self-explanatory;
- Adaptability when operates with datasets that may have errors;
- Versatility for a large variety of data mining tasks.

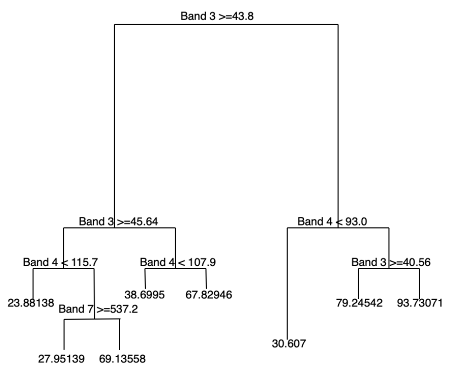
There are two main types of decisions trees, they are classification and regression trees.

Classification trees use training data to build the model and the tree generator coordinates which variable needs to be split at a node, the decision to stop and the terminal nodes assignment to a class. This type of trees are used for the classification of an object or an instance to a predefined set of classes that is based in the values of their attributes. Basically, in classification trees, class labels are represented by the leaves while the features combination, that lead to those class labels, are represented by the branches.

In regression trees, the input space is mapped into a real-valued domain and, for instance, it is possible to predict the demand for a certain product given its characteristics. This type of decision trees presents a structure very similar to the classification ones. The relationship between the predictor and the response variables is calculated and terminal nodes are predicted function values. So, the predicted values are limited to the values in the terminal nodes [36, 7].



(a) Classification Tree



(b) Regression Tree

Figure 2.15: Examples of Decision Trees, adapted from [7]

2.6.4 Artificial Neural Networks

Artificial Neural Networks (ANN's) are composed by a large number of "neurons" that can be often organized into layers. "Neurons" consist of simple linear or nonlinear computing elements that are interconnected frequently in a complex way. These structures receive a signal, process it and then transmit it to another "neuron".

ANN's were developed with the goal of simulate biological nervous systems by combinations of many simple computing elements ("neurons") . These elements belong to systems highly interconnected with the ambition that, through self-organization or learning, a complex phenomena such as "intelligence" would appear. In ANN implementations, at the connections, called edges, between the "neurons", the signal is represented by a real number (normally). The output of each "neuron" is determined by a non-linear function of the sum of the respective inputs. "Neurons" and edges of the ANN, usually, have a weight that is being adjusted during the learning procedure and that increases or decreases the strength of the signal. These weights are obtained through a process that involves highlighting the contribution of particular aspects of a data set over others, creating a final outcome or result. Basically, some of the data variables are modified to have a larger contribution for the result than others.

There are several ways to use neural networks, but there are three that are the most common, such as, data analytic methods, models of biological nervous system and real-time adaptive signal processors [37].

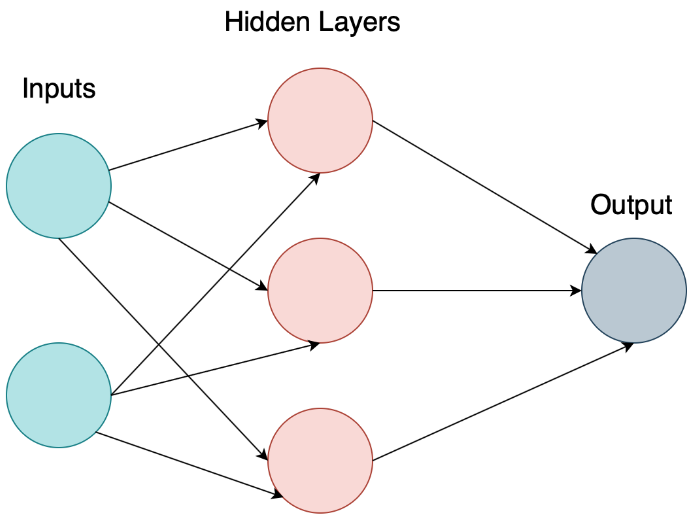


Figure 2.16: Artificial Neural Network example.

The complexity around neural networks is originated by the accumulation of several layers. It's possible to divide the working procedure of neural networks into to two main processes: learning and prediction.

In learning process, while the inputs and the desired outputs are being received, the internal state is being updated respectively, with the goal of obtaining an output very similar to the desired one.

In the prediction process, inputs are received and, using the internal state, the most plausible output is generated by using past "training experience".

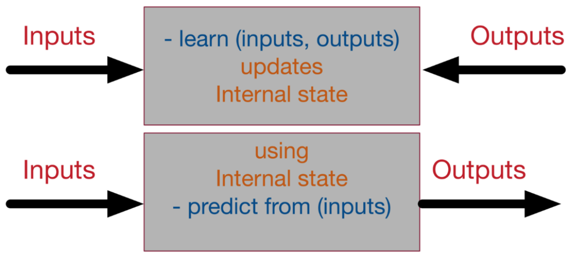


Figure 2.17: Processes involved in Neural Networks, adapted from [8].

Loss Function

At some point, the model has the actual and the desired output. To evaluate the performance of the neural network in achieving outputs as close as possible to the desired ones, a loss function is calculated. This loss function is simply calculated by subtracting the actual output to the desired output. This function can result into negative values, when the NN undershoots (predicted output< desired), and into positive values when the NN overshoots (predicted > desired). Its possible to only focus in the absolute error, regardless of the over and undershooting, by calculating the loss function using the absolute function

$$loss = |desiredoutput - actualoutput| \tag{2.8}$$

In some cases, the total sum of the errors can be the same in different situations. This happens since there's a lot of small errors and few large ones, leading to the same sum of errors. So, it's better to have a distribution of many small errors than to have one with few large errors. For the NN to converge to this distribution, the loss function is defined as the sum of squares of the absolute errors, therefore, small errors are counted much less than large ones. Basically, the goal is to minimize the overall error over the whole dataset. The loss function turns to be an error instance that calculates an indicator that reflects how much precision is lost, when replacing the desired output by the one that is generated by the neural network model.

Machine learning transforms itself into an optimization problem with the goal of minimizing the loss function [8].

Differentiation

To optimize the loss function, it's possible to use different techniques that can modify the internal weights of the neural network. These techniques aim to deduce which weight result in the smallest sum of the squares of the errors over the dataset. Genetic algorithms or brute force search are examples of techniques that can be used.

Differentiation is a mathematical concept that can strongly guide this optimization problem in the right way. This concept deals with the derivative of the loss function, since this value indicates the rate of which the loss function is changing its values at a certain point. The advantage of using the derivative is that it's more precise to calculate and much faster. Basically, if the weight is decreased the total error will decrease too. So, when the network is initialized randomly, the learning process checks the derivative and takes some decision [8]:

- If the derivative is positive, the error increases if the weights increase, therefore, the weight should be decreased;

- If the derivative is negative, the error decreases if the weights increase, therefor, the weight should be increases;

- If the derivative is zero, it's a stable point;

Back-propagation

In order to obtain more variations in the functionality of the neural network, more layers are needed between the inputs and the outputs. Since the derivative is decomposable, it is possible to back-propagate it. There's a starting point of errors (loss function) and it's known how to derivate each function from the composition, therefore, it's possible to propagate back the error from the end to start. Through the creation of a library of differentiable functions or layers, for each function, it is possible to forward-propagate by directly applying the actual function. Also, back-propagate becomes possible, by calculating the derivative of this function. So, any complex neural network can be composed. For this to happen, it's only necessary to keep, in a stack, the function calls throughout the forward pass and their parameters. Then, through the calculation of the derivatives of these functions, it's possible to back-propagate the errors. There's a technique that can be used, called auto-differentiation, that de-stacks the function calls. It only requires that each function is given with the implementation of its derivative. Basically, any layer can forward its result to many other layers and, to back-propagate, the errors from all the target layers are summed [8].

Usually, for the derivation of the back-propagation algorithm, the *sigmoid* function is used, since it has nice properties [38].

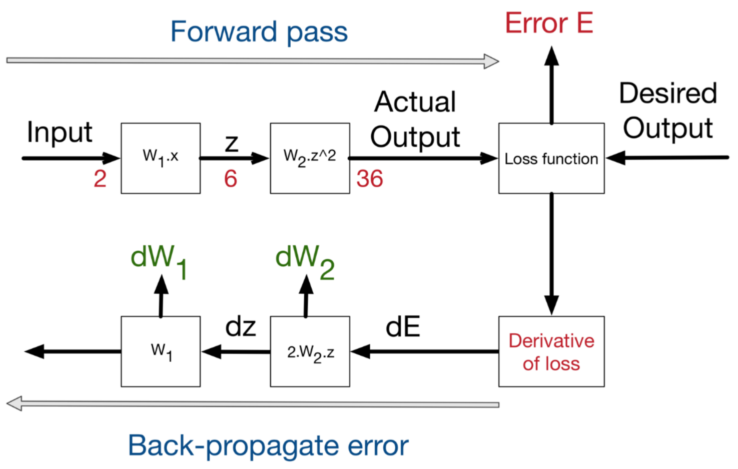


Figure 2.18: Process of back-propagating errors, adapted from [8].

The weight is updated through [8]:

$$weight = oldweight - DerivativeRate \times learningrate \tag{2.9}$$

The learning rate is a constant with the aim to force the weight updating slowly and smoothly.

2.6.5 Deep Learning

Deep learning is a subgroup of machine learning in which the tasks are divided and distributed to algorithms that organize everything into layers. Each layer receives the information from the previous layer, builds up and transmits the data to the next one. In this type of learning, artificial neural networks adapt and learn from large amounts of data.

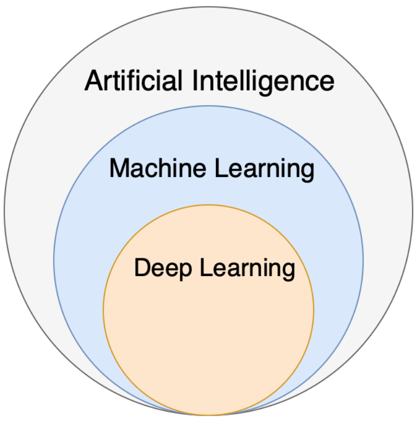


Figure 2.19: Relation between AI, ML and Deep Learning, adapted from [9].

Deep learning models show a good performance when dealing with large groups of data instead of stopping to improve after a saturation point, like old machine learning models. The difference between machine learning and deep learning is at the feature extraction process. In Figure 2.20, it is possible to observe that feature extraction, in ML, is done by a human and in a deep learning model is figured out by the model itself [39].

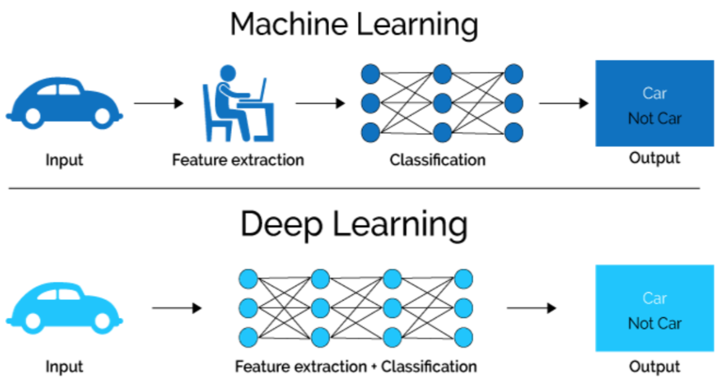


Figure 2.20: Feature extraction process in machine learning and in deep learning, adapted from [9].

Basically, deep learning has successive layers of representations and the depth of this type of models depends on how many layers are being used. So, a deep learning model is a directed and acyclic graph of layers [40].

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are an architecture of deep learning and are mainly used in areas such as image recognition and classification. In CNN's, an image is taken as an input, processed and classified with a category.

Features are extracted through convolution by the convolutional layer. This operation is applied in the beginning because it preserves the relationship between pixels, by using small squares of input data to learn the image features. Convolution takes two inputs, the image matrix and a filter/kernel (sometimes called "Feature Map"). Different kind of operations such as blur or edge detection, can be performed by applying different filters to the convolution of an image.

The goal of CNN's is to reduce images into a form much easier to process, without loosing important data, in order to achieve a good prediction [10].

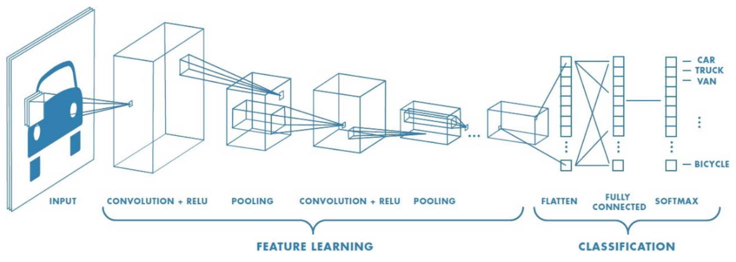


Figure 2.21: Neural Network with convolution layers, adapted from [10].

Padding & Rectified Linear Unit

In some cases, the filter applied to the input image doesn't fit correctly. Too solve this situation, it's possible to pad the image with zeros (zero-padding) to make the filter fit. Also, the part of the image where the filter doesn't fit can be dropped, keeping only the valid part of the image.

Activation functions are used, in artificial neural networks, to define the output of a node using an input or a group of inputs. Rectified Linear Unit (ReLU) is the activation function commonly used in deep learning. This function returns zero if a negative input is received and, for any positive value received, it returns the same value back. ReLU can be represented by

$$f(x) = max(0,x). \tag{2.10}$$

The goal of applying ReLU is to introduce non-linearity in convolution networks since, for real world data, non-positive values doesn't matter for the learning procedure.

There are other non-linear functions such as sigmoid that can be used. Although, ReLU has a better performance [11, 10].

Stride & Pooling Layer

Stride is represented by the number of pixels shifts over the input matrix. If the stride is one, the filters move one pixel at a time so, when the stride is, for example, three, the filter moves three pixels at a time.

When images are too large, in order to reduce the number of parameters, pooling layers are used. The dimensionality of each map is reduced but all the important data is kept. There are two types of spatial pooling: Max and Average pooling.

Max pooling returns the largest element from the rectified feature map. Average pooling returns the average of all the values.

Noisy activation is discarded and de-noising along dimensionality reduction is performed by max-pooling. Also, this type of pooling works as noise suppressant. Average pooling only performs as noise suppressant by applying dimensionality reduction. So, max pooling has a better performance than average pooling [10].

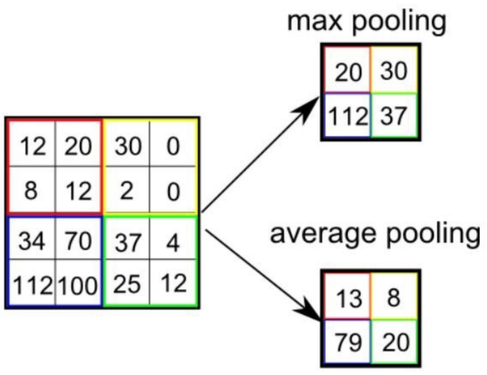


Figure 2.22: Max and average pooling operations, adapted from [11].

The number of layers that perform pooling and convolution operations can be increased to capture low-level details, depending on the complexity of an input image. However, more computational effort is needed [11].

After all the above procedures, the model understands the features and it's possible to feed the output to a regular NN (fully connected layer) for classification.

The feature map is converted into a vector and taken as an input of the fully connected layer. This NN combines the features and creates a model and, in the end, an activation function such as *sigmoid* or *softmax* is used to classify the outputs with a category (car, person).

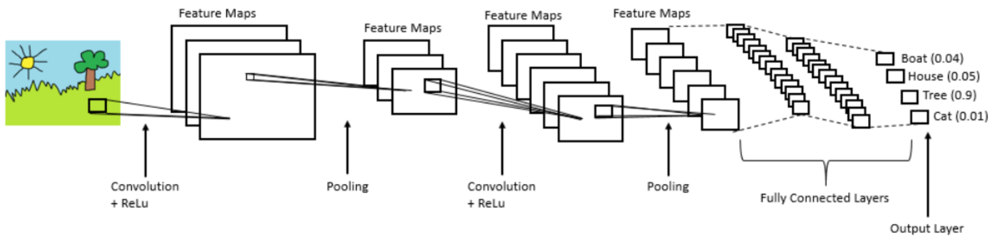


Figure 2.23: CNN architecture, adapted from [10].

2.6.6 Object Detection Methods

Object detection has been, since a long time, a common problem in computer vision. Usually, this type of methods aim to identify and classify objects that appear in an image. The existence of a range of objects to identify and the necessity of predicting bounding boxes that surround the object, can difficult the task of detecting objects. To define a bounding box, four parameters $[x,y,w,h]$ are used. (x,y) refer to a spatial position in the box (center or upper-left corner) and (w,h) refer to the width and height of the corresponding box.

Through convolutional neural networks, deep learning models had a positive impact in this area.

Initially, object detection tasks were performed by the technique of “the sliding window”, where a rectangle, with different dimensions, moves over the whole image with the goal of finding relevant objects. However, computational effort is large since a classifier, for each class, needs to be applied to the window. CNNs also implement this technique but in a more efficient way.

2.6.6.1 Fast-RCNN

This object detection system is composed by two modules. The first one is a deep fully convolutional network, where regions are proposed, and the second one, which uses this regions, is the Fast-RCNN detector. These modules combined form a unified network for object detection. Basically the region proposal networks (RPN) module indicates where the Fast-RCNN should observe.

Region Proposal Network

Region proposal network (RPN) is a convolutional network that aims to detect regions where objects can be found. The deep fully CNN is used as the feature map and is arranged into $H \times W$ nodes. These nodes are related to receptive field that has a certain size in the input image. Feature map depth determines the actual size.

A convolutional layer is placed after the feature map, with the goal of learning 256 features for each node. This is possible since a 256 channel convolutional is connected through a 3×3 kernel. A new feature map results from this and is used to predict the presence of an object.

Finally, this feature map is passed to a predictor. It's possible to say that, this predictor, divides into two. The first one, predicts if the region contains or not an object, and, if it contains, the second one predicts the bounding box that will surround that object. Basically, one performs as a classifier and other as a regressor.

Anchors

Anchors are a set of reference regions that the region proposal network uses per node, to accelerate the training. This structures represent a rectangular region in the input image and is located in the center of the respective field, in each node, in the feature map. Scale and the aspect ratio define an anchor.

For each sliding window position, multiple region proposals are predicted. In each location, the number of maximum possible region proposals is defined as k. So, the classification layer outputs $2k$ scores that indicate the probability of existing an object or not. The regression layer will have $4k$ outputs that encode the coordinates of k boxes.

Classical *cross-entropy* loss is used, by the classifier, in the training. This type of loss measures the performance of a classification model that has a probability value, between zero and one, as an output. It increases as the predicted probability diverges from the actual label. A *smooth-L1* loss function is used by the regressor [41, 12].

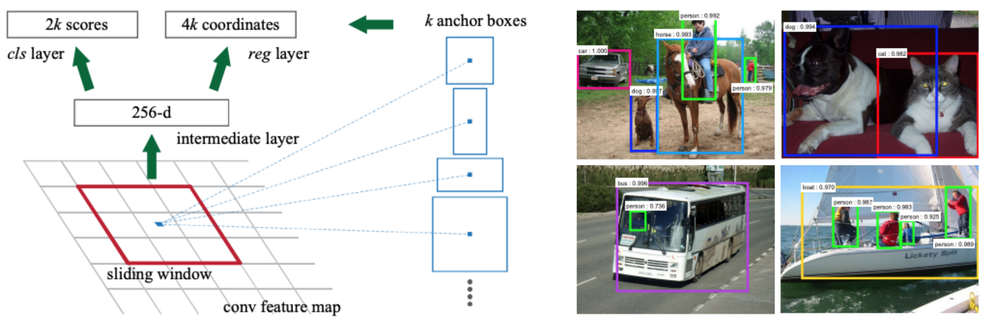


Figure 2.24: Region Proposals Network and an example of detections done by using RPN proposals, adapted from [12].

Detection

The detection is made by a Fast-RCNN model and the set of candidate objects is provided by the RPN. The goal of the Fast-RCNN is to predict the most plausible class (through a set of provided classes) and a class that also represents the background. For this purpose, a convolutional network is used since it can produce discriminative features for each region of interest (ROI).

Fast-RCNN extracts a subregion from the candidates regions of interest proposed by the RPN, using the feature map computed from the input image. The diversity of sizes within the region of interest can define the architecture of the convolutional model. To solve this, Fast-RCNN suggests a special layer called “ ROI Pooling Layer” that, through the average pooling operation, transforms the feature map of the given ROI into a feature map of $H \times W$ nodes. Finally, the result is passed to a couple of fully connected layers and then to a set of classifiers that predict the most probable class. Also, a set of regressors adjust the input region of interest, using semantic information.

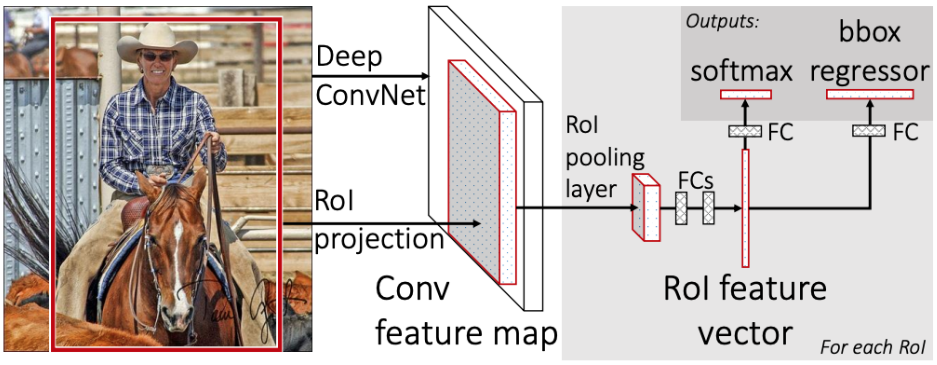


Figure 2.25: Fast-RCNN architecture, adapted from [13].

Fast-RCNN has shown high performance results, however, for real-time implementations, has very slow results [12, 41].

2.6.6.2 YOLO-Real-Time Object Detection

YOLO (You Only Look Once) is a minimalist approach to object detection that also uses a convolutional network to generate a feature map from the input image. This feature map has $S \times S$ nodes and each one of them is related to a respective area in the input image. Also, each node predicts B bounding boxes that are weighted by the predicted probabilities. This weight represents the confidence of the model in finding an object in that box. Five predictors are needed for each bounding box, in order to estimate the (x, y) coordinates (center of the box), the width (w), height (h) and the confidence of the prediction. Also, a conditional class probability, C, is predicted. One set of class probabilities per each node is predicted, despite of the number of boxes.

Finally, the regressors are estimated through

$$S \times S \times (B \times 5 + C). \tag{2.11}$$

So, it’s possible to refer to this model as a regression problem. In Figure 2.26, it’s possible to observe the division of the input image in a $S \times S$ grid, the prediction of the bounding boxes, the confidence for those boxes and the C class probability map. Finally, predictions are encoded using the equation referenced above(2.11), resulting into the final objects detection [14].

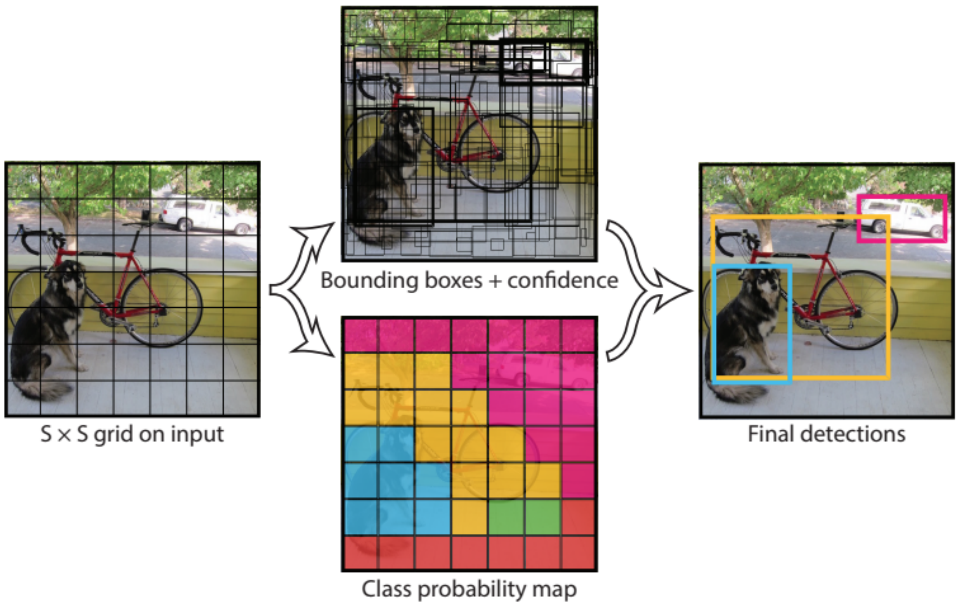


Figure 2.26: YOLO Model, adapted from [14].

2.6.6.3 SSD - Single Shot Detector

Single Shot Detector is an approach to object detection based on a feed-forward convolutional network. This network produces a collection of bounding boxes and scores that measure the presence of an object class and a non-maximum suppression step provides the final detections. Basically, it is only necessary one single shot to detect multiple objects within an image. While approaches based on RCNN need two shots, one to generate region proposals and other to detect the object in each proposal, methods that use SSD are much faster since, it is only necessary one single shot to achieve object detection.

Feature extraction is achieved by a group of convolutions and a feature layer is obtained. For each location, there are k bounding boxes with different sizes and aspect ratios. For each bounding box, c class scores and four offsets relative to the default bounding box, are computed [15].

In order to achieve a better accuracy in object detection, different layers of the feature map go through a small *3 X 3* convolution, as it is possible to observe in Figure 2.27.

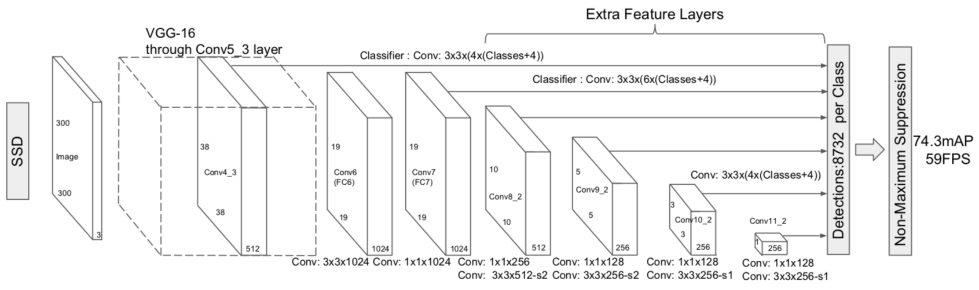


Figure 2.27: SSD Architecture, adapted from [15]

2.7 Conclusion

In this chapter, features extraction techniques, machine learning basics and methods to perform object detection were described. In an initial stage, features extraction techniques were introduced as a way to understand how algorithms identify and extract points of interest from an input image. Also, the way they performed this task was explained and demonstrated. However, in this project, to achieve object classification, machine learning models have a better performance and feature extraction techniques are applied in an implicit way. Therefore, the main focus will be in machine learning object detection methods.

In section 2.6.6, it is possible to find examples of objection detection methods. From these examples, the one that was chosen to help developing this dissertation is referred in section 2.6.6.3. Pre-trained models with this type of architecture will be used in conjunction with a software library called TensorFlow. This library was chosen due to the abstraction provided by it. Instead of dealing with particular details or implementing algorithms, the main focus becomes the overall logic of the application. Basically, this system deals with details in a implicit way or "behind the scenes". Therefore, this library will be studied and used to retrain a pre-trained SSD model, in the next stages of this dissertation.

Chapter 3

Methodology

In this chapter, the software library referenced in section 2.7 is presented and its implementation steps are demonstrated. Finally, a real-time implementation of this software library object detection API is referred.

3.1 TensorFlow

TensorFlow is a framework that builds deep learning models and it was created by Google. This framework allows the creation of trained production models and was invented thinking about processing power limitations. It provides a front-end API that builds applications by using *Python*, however, executes these applications in high-performance *C++*. Basically, the actual math operations are not performed in Python but in *C++*. So, *Python* just coordinates traffic between pieces and high-level programming abstractions hook these pieces together.

The main operation of TensorFlow is to create computational or dataflow graphs. It's possible to think about a computational graph as a network of nodes where, each node, represents an operation that could be either a simple addition or a complex equation. The operation can be referred as *op* and it returns zero or more tensors that can be used in the graph. Each operation can receive a constant or a n-dimensional matrix. So, each edge or connection between the nodes is a multidimensional array (tensor).

In Figure 3.1, it is possible to observe the creation of a computational graph by multiplying two constant tensors and outputting the result. All the inputs necessary to the operation are run automatically and, usually, the inputs run in parallel [16, 17].

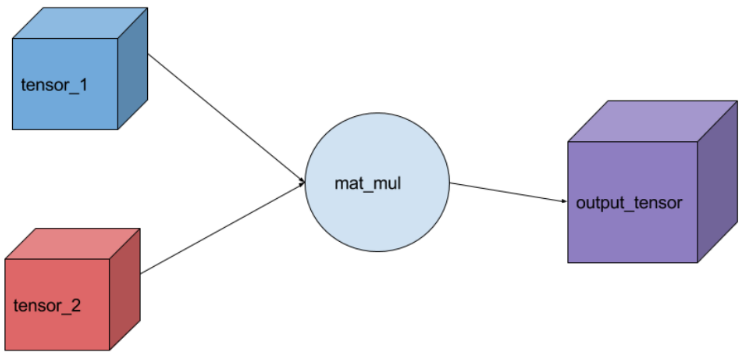


Figure 3.1: Computational graph example, adapted from [16]

Figure 3.2, displays TensorFlow as two parts. One represented by the multidimensional array (tensor) on left and other represented by the operations applied to data. Also, TensorFlow provides an object detection API that makes easy the construction, train and deployment of object detection models.

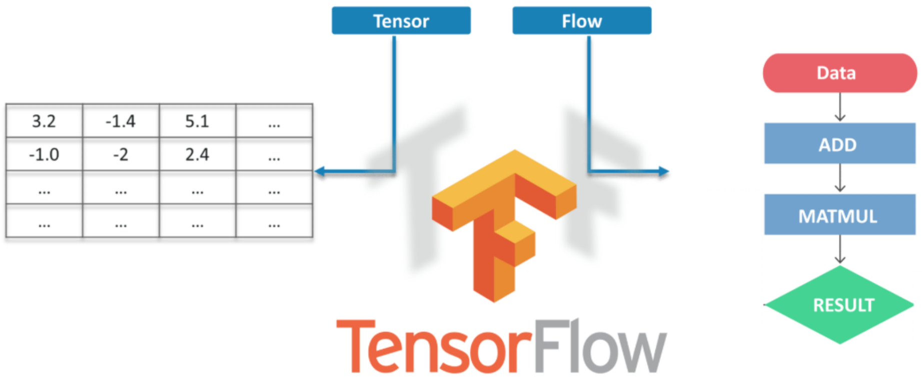


Figure 3.2: TensorFlow procedure, adapted from [17]

3.2 TensorFlow Configuration

3.2.1 Computational Graph

Every variable and operation created is added automatically to this graph. By TensorFlow importing the library, the default graph is initialized. When dealing with the creation of multiple models, creating a graph object instead of using the default one can be useful. However, in most of the situations, it is better to use the default graph.

3.2.2 Session Objects

There are two types of session objects in TensorFlow: **tf.Session()** and **tf.InteractiveSession()**.

tf.Session()
The environment where the operations and tensors are executed is encapsulated by this session object. Each session has its own variables, queues and readers so, it is important to use the close method when the session is over. When this function is called, all the dependencies that are necessary for the graph, are executed.

Each session has three optional arguments:

- graph, the graph that needs to be launched;
- target, execution engine to connect to;
- config, protocol buffer with configurations for the session.

tf.InteractiveSession()
This type of session it is more related to the use of *IPython* or *Jupyter Notebooks*. It allows to evaluate tensors and run operations without needing to always run the session, therefore there's no need of explicitly create a session object.

3.2.3 Variables

Each session manages the variables and are able to persist between them. This is very useful since tensors and operations are immutable, that is, they're unchangeable objects. To initialize the variables, TensorFlow variable initialization function is used. Then, this function is passed to each session, therefore, is possible to have multiple sessions but with the same variables. Also, it is possible to update the value of a variable inside a session.

3.2.4 Scopes

In a way to reduce models complexity, TensorFlow uses scopes. These structures are very simple and can be nested inside other scopes. Usually, these structures are used in collaboration with other TensorFlow tools.

3.2.5 Setting Up the Environment

To set a proper environment for the execution of TensorFlow, different libraries are necessary. By consulting [42], the dependencies that need to be installed are:

- Python-tk;
- Pillow 1.0;
- lxml;
- Cython;
- Protobuf;
- Matpolib;
- contextlib2;
- TensorFlow;
- Jupyter notebook;
- pycocotools;

Then, TensorFlow models folder is cloned from [43] and the directories that will be used are organized in the following way:

models $>$ research $>$ object_detection

PROTOBUF

Protocol buffers are a mechanism used to serialize structured data. It is Google language-neutral and is smaller, simpler and faster. In every TensorFlow execution, the protocol buffer needs to be installed at an initial state. It should be executed in the same directory of object_detection folder.

PYTHONPATH

PYTHONPATH is an environmental variable that defines a search path. It is used by the *Python* interpreter to find out where the modules, that need to be imported, are located. So, for every new terminal used to run TensorFlow, it is necessary to indicate the path to these modules. This is done by running this environmental variable in the research directory.

3.3 Transfer Learning

Training a new model from scratch is a very hard task requiring large amounts of data and computational effort. Transfer Learning uses a pre-trained model as an initial point of a new model, that is, reuses the pre-trained weights as the starting weights of the new model.

In another words, imagine there's a Task A and a Task B and it is necessary to perform transfer learning between them (from A to B). This only makes sense if:

- Both of the tasks share the same type input;
- There's more data for Task A than for Task B;

- Low-level features from Task A can be helpful for the learning procedure in Task B.

Transfer Learning can be done in two different ways, by fine-tuning or feature extraction [44].

3.3.1 Fine-Tuning

Sometimes, it is necessary to perform a task that is very similar to another. This other task, is performed by a trained and designed model. Assuming that the tasks are similar, it is possible to use this pre-trained model to perform a new task. This is done by taking advantage of the feature extraction process in the top layers of the model instead of creating a new feature extraction network. Basically, the output layer is replaced by a new one that recognizes the number of classes required. Also, this new output layer, is trained to use low-level features and map them to the desired output.

3.3.2 Feature Extraction

In this way of performing transfer learning, a pre-trained network is used to compute features for an input image. A new classifier is added and trained on top of the pre-trained model, that is, the previous feature map is repurposed. The original model it is not modified and new tasks benefit from previous learned features.

In Figure 3.3, it is possible to observe a comparison between fine-tuning and feature extraction methods.

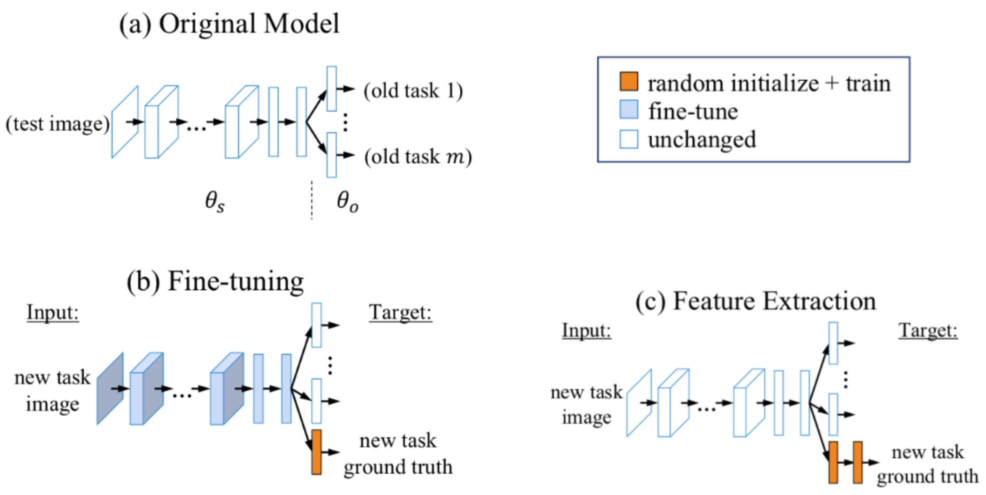


Figure 3.3: Fine-tuning and Feature Extraction methods, adapted from [18].

As it was referred in section 1.3, the goal is to identify and classify a particular group of objects. This group of objects are vehicles, people, guns and tanks. To achieve this, transfer learning fine-tuning method will be explored.

The idea it is to use a pre-trained model and fine-tune the layers in order to recognize particular objects. Since there are already trained models that detect people and vehicles, it is only necessary to fine-tune them, in order to start detecting guns and tanks too. Fine-tuning it is the best approach since there's no need of building a feature extraction network from scratch and there's different pre-trained models that can be used.

TensorFlow has different pre-trained models available at [19]. In table 3.1, there are some examples of pre-trained models along with the respective speed and COCO mAP values. The speed value stands for the running time of the pre and post-processing. COCO mAP is a measure metric to evaluate the performance of the model detector when dealing with COCO (Common Objects in Context) dataset. .

From table 3.1, the pre-trained model *ssd_mobilenet_v1_coco* was used in the next stages of this dissertation. SSD stands for "Single Shot Detector", that is, it only needs one shot to detect multiple objects within the image. This model detects a large range object classes and seems to be good choice to use as pre-trained model.

Model	Speed (ms)	COCO mAP
ssd_mobilenet_v1_coco	30	21
ssd_mobilenet_v2_coco	31	22
ssd_mobilenet_v2_quantized_coco	29	22
ssdlite_mobilenet_v2_coco	27	22

Table 3.1: Examples of some pre-trained models provided by TensorFlow, adapted from [19].

In section 3.4, the steps needed to perform the model fine-tuning, by using TensorFlow Object Detection API, are demonstrated.

3.4 TensorFlow Object Detection API

As it was mentioned above, TensorFlow Object Detection API is a very powerfull tool that facilitates the creation and deployment of object detection models. Figure 3.4 represents a summary of the steps taken in the implementation of this API. The next subsections will explain these steps.

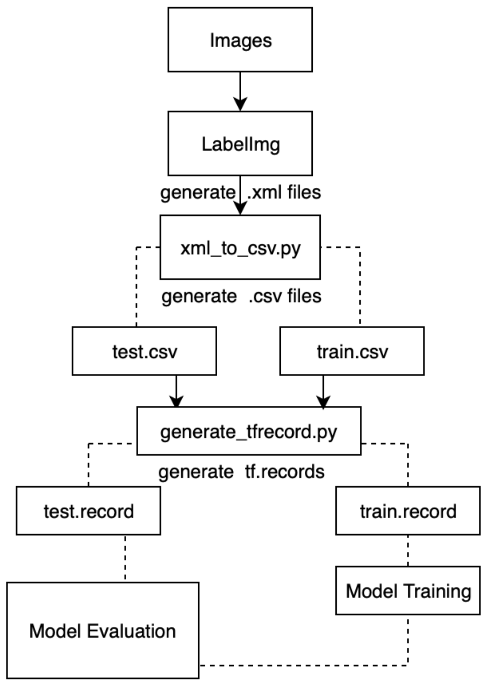


Figure 3.4: Implementation Steps.

3.4.1 Dataset Construction

In order to detect a particular group of objects, a dataset containing only images of guns and tanks was created. Then, it was necessary to indicate/label in these images where the main object was located. This is important since, the model needs to know where to find the specific object that needs to be learned.

To label the objects, a tool called *LabelImg* and provided by [45] was used. Through this tool, it is possible to manually create a bounding box that surrounds the specific object and indicate his type. By doing this, the tool automatically generates a *.xml* file that contains the dimensions and location of the bounding box as well the name of the object. So, in the end, each image should have a *.xml* file associated.

Finally, the dataset was divided into two folders, test and train. Usually, 10% of the images go into the test folder and 90% go into the train folder. Images and *.xml* files in train folder are used to train the model and, the elements of test folder, are used to evaluate the performance of the model, that is, evaluate if the model is predicting the type of the object correctly.

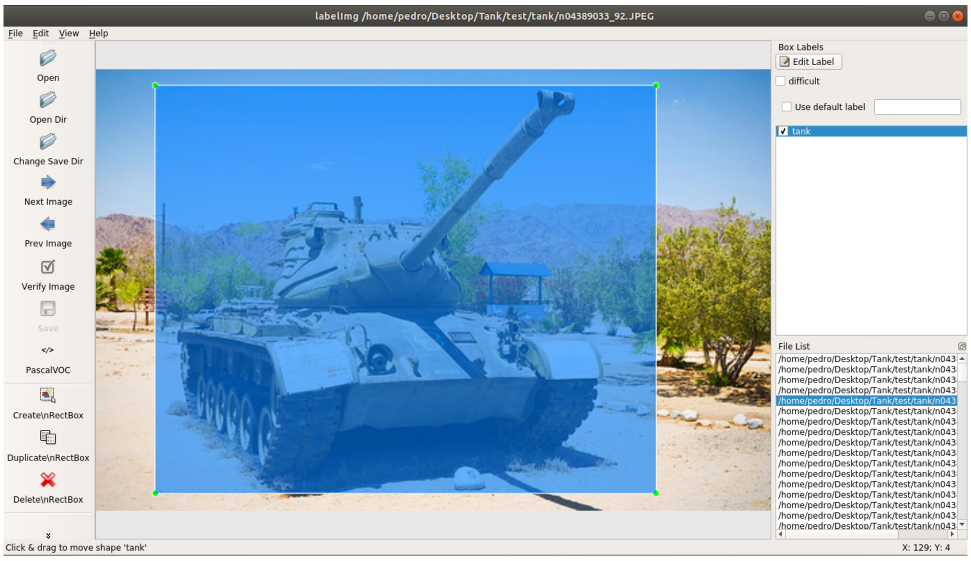


Figure 3.5: LabelImg tool usage to label a tank.

In an initial stage, all the objects that could be detected in an image were labeled, that is, not only tanks and guns were labeled. This introduces errors during training because the same input can give two or more possible outputs, implying that the function that maps images to classes isn't well-defined. So, for the further implementations, only guns and tanks were labeled in the dataset.

3.4.2 Label Map

To map each label to an integer value, TensorFlow uses a label map. This file contains the labels of the objects that model needs to detect and an ID associated to each one. The label map file should have the *.pbtxt* format and is used in training and detection processes.

In this dissertation, two label maps were used. One, only containing the labels "tank" and "gun". Other, containing the same labels as the label map of COCO dataset plus "tank" and "gun" labels. So, the ID should be attributed by respecting the order of the previous labels.

These two label maps were used in two different training sessions, leading to two different sets of results.

3.4.3 TFRecords Generation

TFRecords are TensorFlow own binary storage format. When dealing with a large dataset, binary storage format can be used in order to improve the model performance and training time. Taking less disk space and less time to copy, binary data becomes much more efficient to use. Furthermore, this format easily combine multiple datasets and integrate preprocessing and data import functionality consistently. This is an advantage when there's a dataset too large to be stored in memory since, only the data that is required at the time is loaded from the disk and processed [46].

To generate the TFRecords, there's a couple of steps that were done. First, it is necessary to convert the *.xml* files, that resulted from the object labeling, to *.csv* files. Comma Separated Values

(CSV) files are text files composed by a list of data. These files are used to store complex data from one application and, then, import, this data, to another application. To convert the *.xml* files into *.csv*, the script *xml_to_csv.py* was used, generating *.csv* files containing, in this case, the file name, the bounding box dimensions(normalized coordinates) and localization and the class of the object. In this script, it is necessary to add the path to the test and train image folders and the path to the folder where the *.csv* files should be stored.

Finally, to obtain the TFRecords, *.csv* files are converted by using *generate_tfrecord.py* script. In this script, the label map needs to be updated, that is, for each label there's a row and a return value associated. This return value should be equal to the ID of the respective label in the label map. So, in this case, "tank" and "gun" were added with the returning values of one and two, respectively. The paths for the *.csv* files and for the output folder, are indicated via terminal when running this script.

Both of these scripts were provided by [47] and adapted for this specific case.

3.4.4 Configuration File

Protobuf files are used by TensorFlow Object Detection API to configure the training pipeline. This configuration file is divided into five components [48].

- **Model Configuration** : This defines what model is going to be used as feature extractor, that is, the model that is going to be trained;
- **Train Configuration** : This defines which parameters are going to be used to train the model;
- **Eval Configuration** : This defines the set of metrics that is used in the model evaluation;
- **Train Input Configuration** : This defines the path to the dataset that will be used to train the model on;
- **Eval Input Configuration** : This defines the path to the dataset that will be used to evaluate the model.

Each pre-trained model has a specific configuration file [49]. In this dissertation, the configurations file of the model *ssd_mobilenet_v1_coco* was reused but with some specific changes. An initial setting of this file and further modifications are demonstrated in the next chapter.

3.4.5 Training and Evaluation

Before training the model, all the files should be organized into folders. In Figure 3.6, it is possible to observe the three main folders that were created:

- **/data** - This folder contains the label map, tfrecords files and the labels files;
- **/images** - This folder contains the images to use in test and training procedures;
- **/training** - This folder contains the configuration file and the label map.

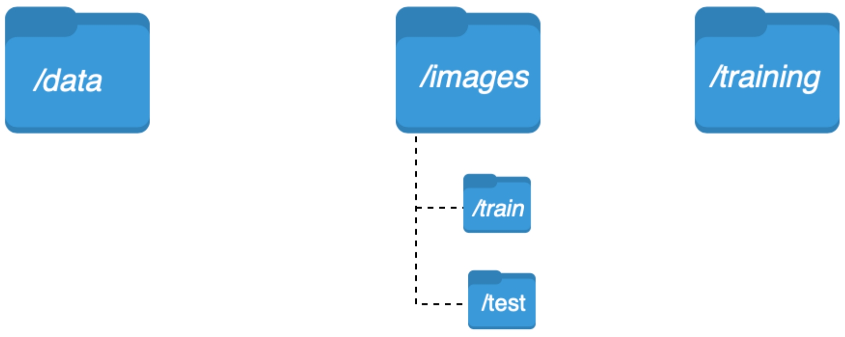


Figure 3.6: Directories Organization.

Then, these folders are moved into */research/object_detection* directory, inside of the models folder that was previous cloned from TensorFlow *Github*. Also, the folder that contains the pre-trained model, that is going to be used, was moved into this directory too.

The training and evaluation processes, can be performed at the same time by using the *model_main.py* script [50], created by TensorFlow researchers. This script is executed by taking as input, the path to an output folder, where it stores the model training checkpoints, and the path to the configuration file. During the training session, the number of the training step and the respective loss are displayed in the terminal.

The model is evaluated while the training takes place. This evaluation measures the accuracy of the model when predicting the class of an object on the images from the test folder.

For better control of this procedures, the TensorFlow visualization toolkit, TensorBoard [51], is used. Through this tool, it is possible to track the loss and the accuracy of the model that is being trained. Also, it displays the evolution of the detection bounding box in test images, during the training. The tool is initiated via *terminal* and it is necessary to indicate the directory of the training checkpoints.

3.4.6 Inference Graph

Finally, to perform object detection with the newly trained model, it is necessary to export the frozen inference graph. This graph cannot be trained anymore, it defines the computational graph of the model at that point. Also, it contains all the training weights. To export the inference graph, TensorFlow provides the script *export_inference_graph.py* [52]. By taking as input the path to the configuration file and to the last model checkpoint during training, this script outputs a folder with the newly trained model and the respective inference graph.

3.4.7 Testing the Trained Model

For the newly model testing, the object detection tutorial jupyter notebook [53], available at TensorFlow *GitHub*, was converted to a *python* file. Then, some adjustments were made to the code in order to adapt it to video feed, that is, real-time video capture or video sequences. So, all the fields in this script that deal with single images are no longer needed. To capture video or read video files, the library *cv2* is imported and the class *VideoCapture* is used. Through this class, frames are grabbed, decoded and returned in an array. Then, this array, is processed by a function that runs the inference and gives the outputs. This function was created by separating the detection boxes definition from the execution of the TensorFlow session. Basically, instead of defining the bounding boxes for each inference, they are defined once. Through this modification, the inference frame rate of the output video gets improved. To visualize the results of the detection, the function *imshow()* is used.

Chapter 4

Results Analysis

In this chapter, each section is composed by the configuration file setting and by the respective outcome. Also, each different outcome is analyzed and each modification done to the configuration file is explained.

4.1 Configuration File: Initial Settings

Before initiating the training and evaluation procedures, some parameters in the configuration file should be defined.

First, the number of classes of objects to detect is defined. Since the goal is to detect tanks and guns, this number was defined as 2. Then, the batch size needs to be specified in the train configuration field. This parameter defines the number of samples that are propagated through the network. When defining this number it should be accountable the fact that, it should not be higher than the number of samples inside the train folder and that a large batch size can degrade the quality of the model. So, in this dissertation, the batch size was defined as 32. In the evaluation configuration field, the number of samples was defined as 55, since this is the number of images inside the test folder. Also, the detection metrics were defined as the ones used in *COCO* dataset. Finally, the paths to the tfrecords, to the label map and to the pre-trained model that are going to be used, were indicated.

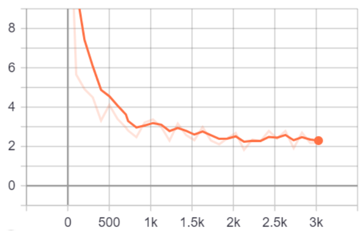
These are the main settings of the configuration file and, in further sections, are maintained.

4.1.1 Initial Results

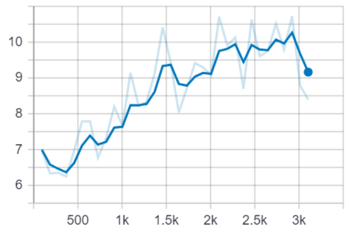
In order to control the training and evaluation of the pre-trained model, the training and validation loss are analyzed. These losses are the sum of errors made, by the training model, when dealing with the train and test datasets. So, it indicates how well the model is doing for these two datasets.

In Figure 4.1, it is possible to observe the validation and training loss corresponding to the previous configuration file settings. During the training steps, the validation loss increases while the training loss decreases. This represents a typical case of overfitting.

Overfitting means that the model is unable to generalize well. It is caused when the model learns noise and random fluctuations in training data as features. These features cannot be applied to new data so, the performance of the model on the validation set decreases. However, the performance on the training set improves.



(a) Training Loss



(b) Validation Loss

Figure 4.1: Validation and Training Loss Graphs

In order to analyze the model accuracy when detecting objects, the detector is evaluated by measuring two different tasks.

- **Classification** - Determining if an object exists in an image;
- **Localization** - Determining the location of the object.

It is important to control the risk of misclassifications so, each bounding box needs to be associated with a model score or a "confidence score". This can be conducted by using Average Precision (AP) that deals with the precision and recall of a classifier. Precision is the fraction of relevant items among the retrieved instances and recall is the fraction of relevant items, that have been retrieved, over the total amount of relevant instances. To calculate the AP score, the average value of precision is taken across all the recall values.

In terms of calculating the model score on the task of object localization, the intersection over union (IoU) is computed. The IoU is given by the ratio of the area of intersection and the area of union between the predicted and the ground-truth bounding box. Intersection over union is a number between zero and one and larger the better. Ideally, the IoU between the predicted and the ground-truth bounding box should be 100% but, anything above 50% is considered a correct prediction.

Finally, the mean average precision (mAP) score combines these two metrics. It is calculated by taking the AP score over all the classes and all the IoU scores.

In Figure 4.2, it is possible to observe that the model bounding box prediction accuracy (mAP score) decreases. Since the model cannot generalize well due to overfitting, it causes a decreasing accuracy during the training. So, there is the possibility of occurring misclassifications.

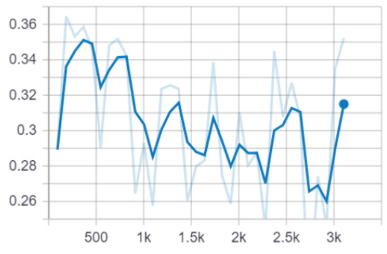
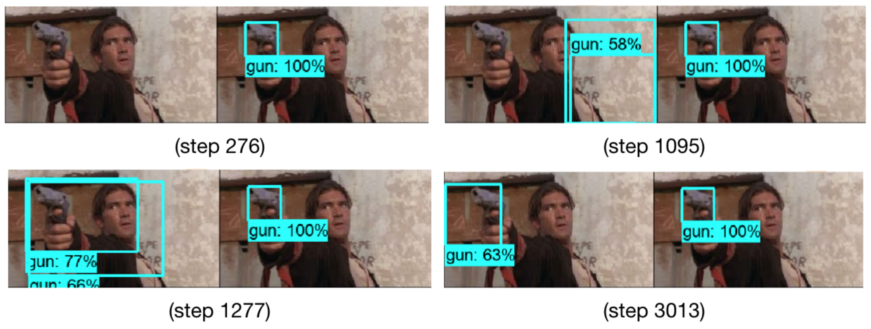
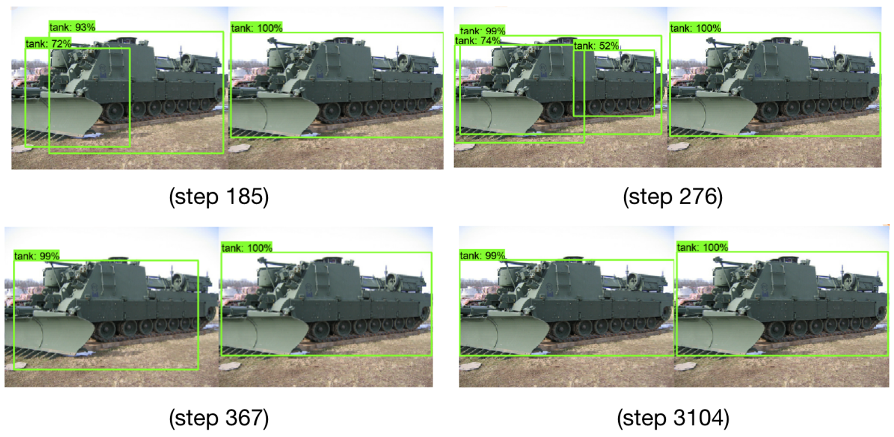


Figure 4.2: Bounding Boxes Precision (mAP).

During training, checkpoints that summarize the training status are made. In each checkpoint, an evaluation step occurs by predicting bounding boxes for possible objects in test images. Figure 4.3, displays the predicted bounding box by the model on the image of the left and the ground-truth on the right. It is possible to observe that, in the initial training steps, the bounding boxes are not in the correct location. However, in the latest steps, the bounding box converges to the correct location of the tank but regarding the gun image it does not converge accurately.



(a) Gun



(b) Tank

Figure 4.3: Bounding boxes predictions through training.

In the next sections, some techniques will be applied in order to eliminate overfitting and to improve the training model accuracy.

4.2 Configuration File: Dropout

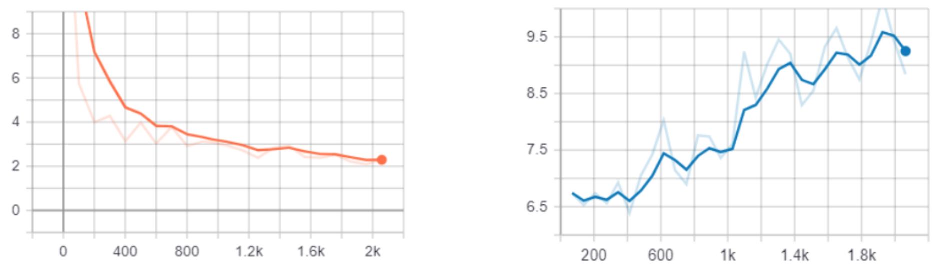
In order to eliminate overfitting, there are regularization techniques that can be applied. Regularization techniques produce slight modifications to the learning algorithm to improve the models ability to generalize. In this section, the regularization technique used is Dropout.

Dropout is a regularization technique that, at every iteration/step, randomly selects and eliminates nodes and their connections from the neural network. So, each step will have a different set of nodes and a different set of outputs. This technique reduces the complexity of the NN and improves the models ability to generalize, when dealing with data that haven't been seen.

To use Dropout in the training procedure, in the field of the box predictor in the configuration file, the flag *use_dropout* was set to true, enabling this technique. Also, the dropout keep probability, that is, the probability of keeping a node, was defined as 0.8. So, the probability of dropping a node is 20%. It is advisable to have a probability of dropout between 20% and 50%. This was the only modification, in this set, made to the configuration file.

4.2.1 Result Set 1

In Figure 4.4, it is possible to observe that the overfitting persists. Even after introducing dropout to the training algorithm, the validation loss keeps increasing while the training loss decreases. Once more, the model is learning noise and random fluctuations as features. It seems that using dropout it is not sufficient to eliminate the overfitting of the model. So, dropout it is not the best approach to this problem.



(a) Training Loss (b) Validation Loss

Figure 4.4: Validation and Training Loss Graphs

Figure 4.5, displays the model predicting accuracy decreasing during the training. Again, overfitting is comproved, since the model cannot generalize and interpret correctly new data.

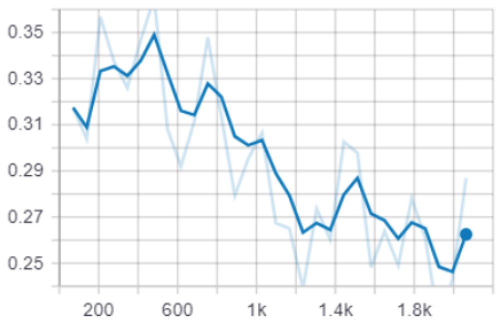


Figure 4.5: Bounding Boxes Precision (mAP).

Figure 4.6 reflects the model accuracy. It is possible to observe a classification error made by this model since, it has confused the position of the men holding the gun with the structure of a tank.

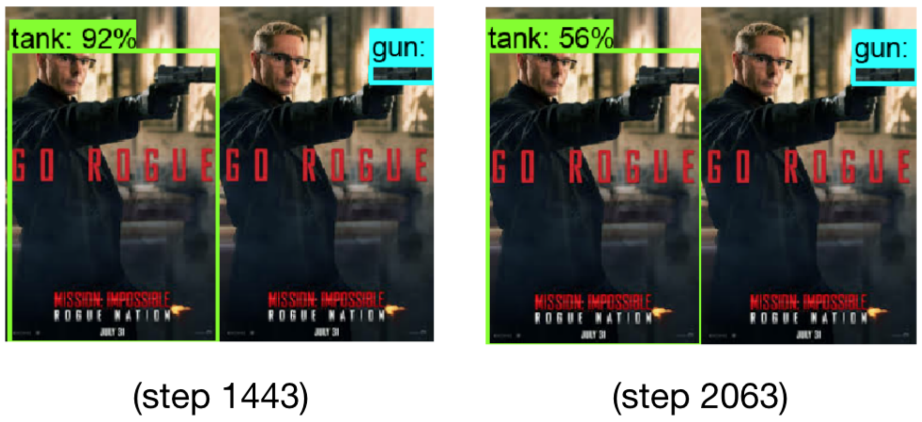


Figure 4.6: Bounding boxes predictions through training.

4.3 Configuration File: Data Augmentation

Another regularization technique that can be used is data augmentation. Data augmentation is a simple way of reducing overfitting by increasing the size of the training data. The training data is increased by perfoming transformations to images in training dataset. Basically, images are rotated, scaled or flipped, depending on the transformations that are chosen. The list of possible transformations can be found at [54].

For this case, the transformations chosen were :

- Horizontal and vertical Flip;

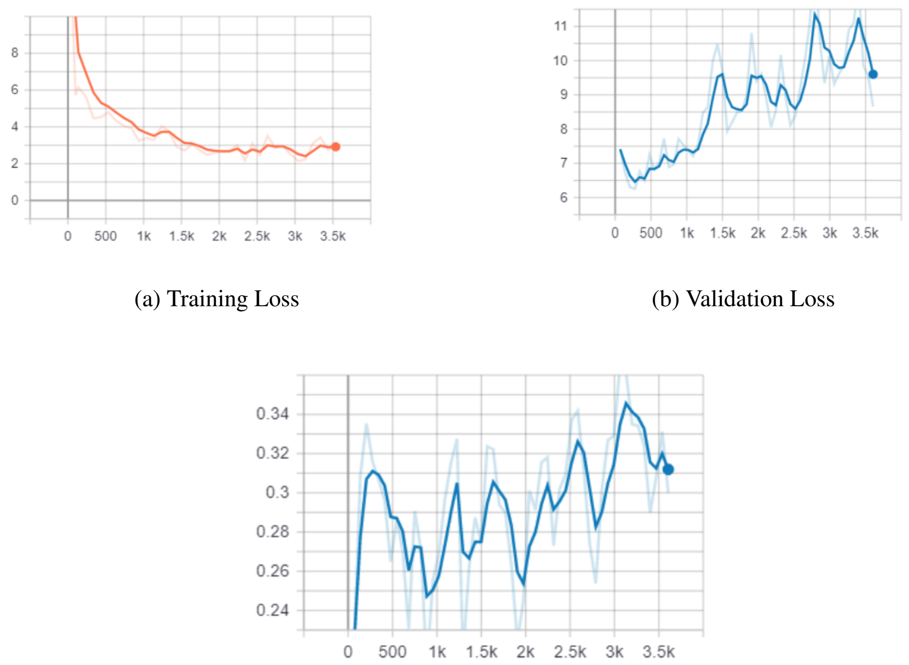
- Image scaling;
- Conversion of RGB to grayscale;
- SSD random crop;

So, images are flipped, enlarged or shrinked, converted to grayscale and randomly cropped according to [15].

It is possible to used this transformations by introducing them into the training configuration field in the configuration file. Also, the probability of converting a RGB image into grayscale needs to be defined. In this case, it was set as 0.5.

4.3.1 Result Set 2

Even with the increase of the training data, Figure 4.7 shows that overfitting steal persists during the training of the model. The validation loss still increases during the training procedure, however, as it is possible to observe in Figure 4.7c, the models accuracy improves. This is unusual, since the increasing of the validation loss should lead to a decrease in models accuracy. This can be caused by examples with an incorrect prediction that keep getting worse in each training step. Also, when examples with a correct prediction lose accuracy during the training steps, can cause this situation. So, introducing data augmentation options is still not sufficient to reduce overfitting.



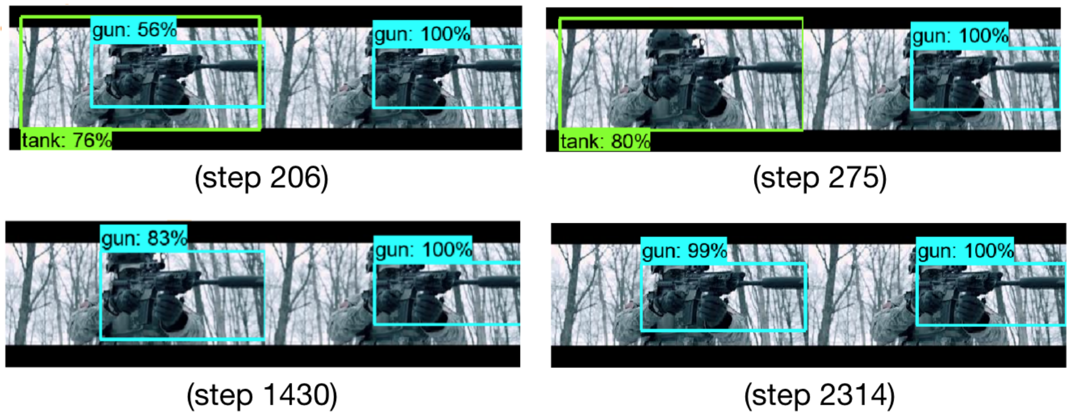
(a) Training Loss

(b) Validation Loss

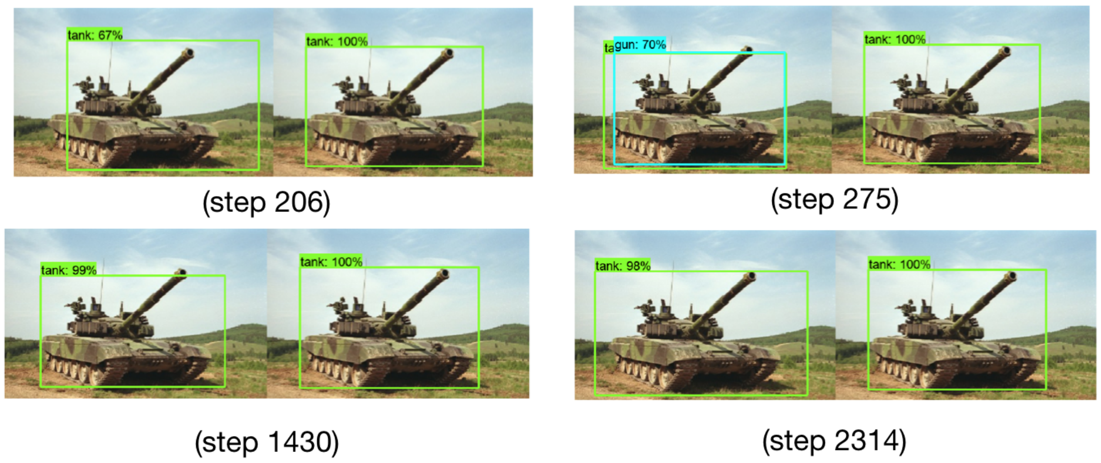
(c) Bounding Boxes Precision (mAP).

Figure 4.7: Validation Loss, Training Loss and Bounding Boxes Precision Graphs

In Figure 4.8, it is possible to observe that in the initial steps of training, a certain misclassification exists. However, in later steps, the object is classified correctly and the bounding boxes converge to right location of the object. So, it justifies the accuracy graph behaviour since, there is high and low peaks along the graph but, during training, it is continually increasing.



(a) Gun



(b) Tank

Figure 4.8: Bounding boxes predictions through training.

4.4 Configuration File: Hybrid Solution

In this section, a hybrid approach to the problem of overfitting was taken. This approach joins fine-tuning and feature extraction techniques in order to perform transfer learning. So, the goal is to maintain the initial layers of the model unchanged and fine-tune the final ones. To achieve this, the weights of the initial layers need to be frozen, that is, attempt to learn new features without forgetting previous ones that had already learned. So, to attempt to freeze the weights, the flag *load_all_detection_checkpoint_vars* in the train configuration field of the configuration file was set to false.

Then, as it was referred at section 3.4.2, the label map of the COCO dataset was used in this training procedure. This label map was used to test whether the model is capable of predicting previous and newly trained classes.

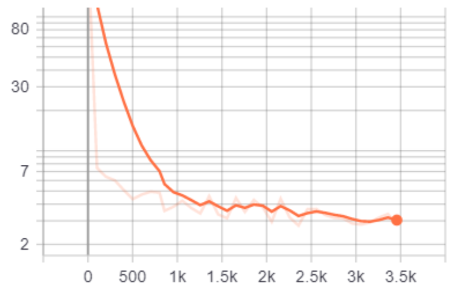
After adding the labels "tank" and "gun" to this label map, the number of classes to detect in the configuration file was defined as 92 since, this was the number of classes that the pre-trained model was previously detecting.

Finally, the previous data augmentation options were maintained in this file, in order to increase the training data.

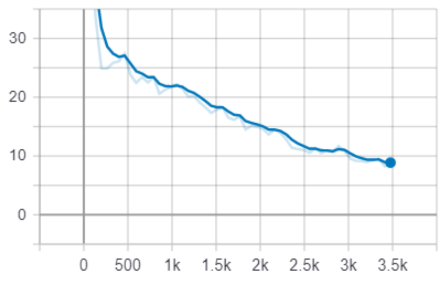
4.4.1 Result Set 3

In Figure 4.9, it is possible to observe that ovefitting was reduced. During the model training, the training and validation loss decrease, meaning that the model generalization was improved. However, the validation loss is still larger than the training loss so, there is still some overfitting. For a perfect fitting, both losses should almost the same and converge into the same value. In this case, it is not a perfect fitting but overfitting was largely reduced.

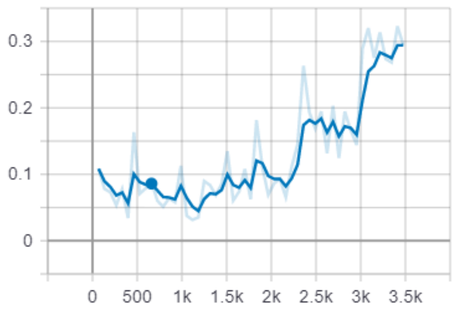
Also, Figure 4.9c, displays the accuracy of the model increasing during the training steps. By analyzing this graph, it is possible to understand that misclassifications are being reduced and the overall accuracy is continuously increasing during training.



(a) Training Loss



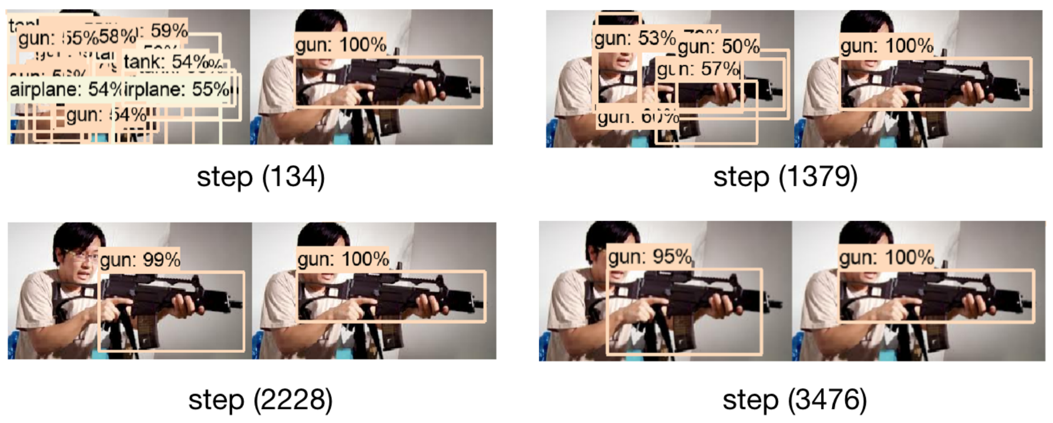
(b) Validation Loss



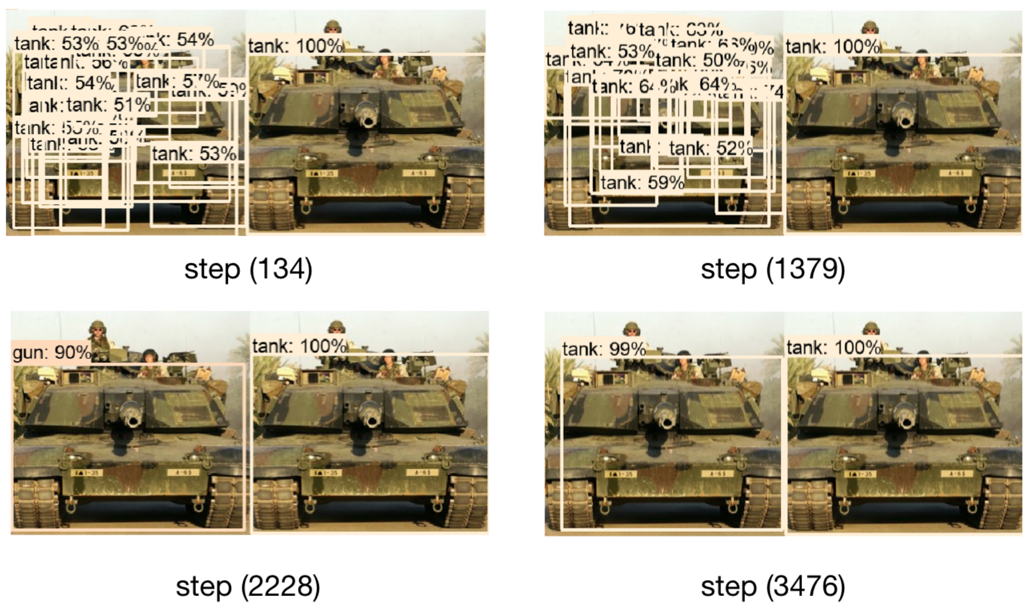
(c) Bounding Boxes Precision (mAP).

Figure 4.9: Validation Loss, Training Loss and Bounding Boxes Precision Graphs

In Figures 4.10a and 4.10b, it is possible to observe that, in the initial steps, some misclassifications occur. However, in the next steps, the classification improves and the bounding boxes are always near the correct location of the object. So, by analyzing the accuracy graph, it is possible to observe that the bounding boxes are progressing according to it.



(a) Gun

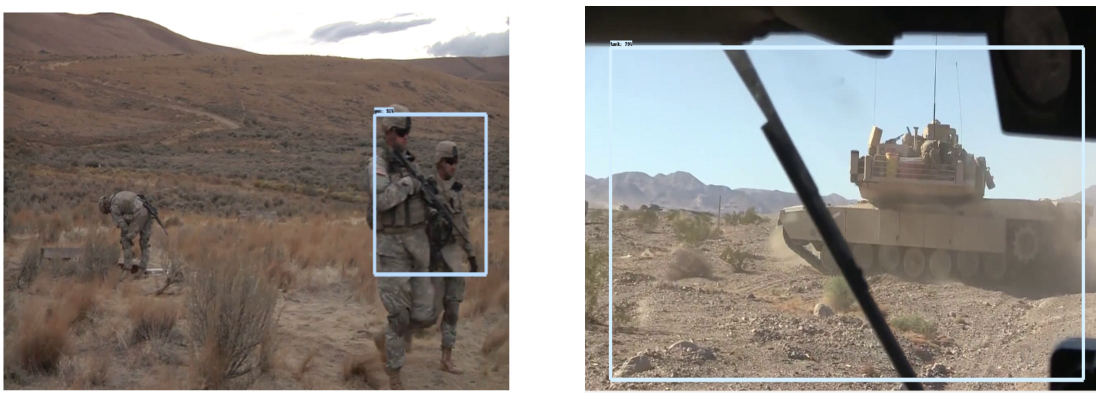


(b) Tank

Figure 4.10: Bounding boxes predictions through training

Since, in this case, it is possible to see an improvement in the model training, is important to analyze the model performance when processing a video sequence. First, the inference graph was extracted and then, by using the script referenced in section 3.4.7, a video sequence was processed by this newly trained model.

Figure 4.11 , displays some of the object detections that occurred. It is possible to observe that it is classifying and locating the object correctly. However, in some cases, it commits incorrect predictions. As it is possible to observe in Figure 4.12, a tank is being classified as a gun. This misclassification problem can be due to the training loss not converging to a lower value. Also, the fact that validation loss is larger than training loss, can cause this problem.



(a) Gun (b) Tank

Figure 4.11: Newly trained model predictions

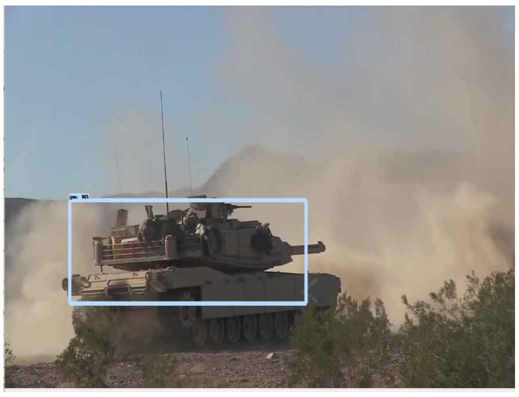


Figure 4.12: Misclassification produced by the newly trained model

4.5 Discussion

To initiate the training procedure, there are numerous parameters in the configuration file, that need to be defined. By using the initial settings, the model displayed a typical case of overfitting. To solute this, different regularization techniques were applied. First, dropout was used but the overfitting persisted. Then, data augmentation options were applied and introduced to the configuration file. In this case, even though the accuracy of the model has improved, the validation loss still increased, that is, the model was overfitting. Finally, the weights of the initial layers were frozen, the label map of the COCO dataset plus the labels "tank" and "gun" and the previous data augmentations were used. This training experiment had the better results among the others. However, training loss needs to decrease to lower values and validation loss too. This can be resolved by passing the model through more training time or increasing the dataset. The dataset can be increased by adding more data to the configuration file, in order to improve the model generalization.

So, it is possible to understand that, when only fine-tuning was performed in sections 4.1, 4.2 and 4.3, it seemed that this approach was deteriorating the original learning of the pre-trained model. Fine-tuning all the layers of the model leaded to overfitting and to an incorrect learning of features. However, when this approach was combined with feature extraction, by freezing the weights of the initial layers, overfitting was reduced and the accuracy improved. This proves that, to perform transfer learning, the fine-tuning approach should be only applied to final layers and initial ones should stay unchanged. Therefore, the approach that combines both fine-tuning and feature extraction techniques in section 4.4, it is the better way to perform transfer learning.

Although in section 4.4 the model training results improved, there are still some problems. Objects that the model was preciously detecting, are no longer being detected. This means that, after training the model to detect guns and tanks, it is only able to detect this type of objects. One possible solution can be building a dataset composed by all the main objects that are necessary to detect. Also, some overfitting is still occurring and can be caused by the complexity of the structure of the model in comparison with the size of the data that is being used. This can be possibly resolved by reducing the number of layers of the network.

Chapter 5

Final Remarks

5.1 Conclusion

Building a software library capable of automatically detecting and classifying objects, in real-time or in video sequences, it is not a trivial problem. Taking into account the large amounts of data that need to be processed and analyzed, it is a process that requires large amounts of time and computational effort. However, after the study here developed, it was possible to find different techniques and tools that implement this type of systems. It was possible to focus on a software library called TensorFlow that already provides an object detection API. Using this API, a training procedure was executed to a pre-trained model with the goal of detecting specific types of objects or classes, namely, guns and tanks.

Throughout this dissertation, different steps have been undertaken to understand how it would be possible to use models that had already been trained to detect various types of classes, with the objective of detecting new objects or to become highly specialized in detecting a limited set of objects. Based on the knowledge and experience acquired, different configuration file settings were defined and used to control the training process towards meeting the desired goal.

The initial general settings that were used, led to overfitting and thus to poor results. Different regularization techniques were applied in order to eliminate this problem. Improvements were achieved when it was used data augmentation options together with a feature extraction approach to freeze layers weights. Then, the model was trained by using the modified COCO dataset label map. The model that resulted from this training procedure achieved the goal of detecting specific objects such as guns and tanks. Even though the solution still led to some misclassification errors, the improvements obtained with this solution provided evidence that this problem can be solved with more training time or dataset augmentation.

The work developed in this dissertation proved that, when dealing with transfer learning implementation problems, it is possible to obtain better results when using the hybrid solution developed and described in section 4.4. Such solution is based on combined use of different existing techniques. The possibility of taking advantage of the feature extraction process of this model, reduces the complexity and the necessity of building a model from scratch. Also, it proves that TensorFlow is a good software tool to be used, since it simplifies this type of processes and offers a wide range of tools to use in model training.

5.2 Future Work

Machine learning systems are at the forefront of the nowadays technologies. Object automatic identification and classification is a growing area, since it can offer a lot of implementing options.

In order for this system to be reliable and usable, the system should identify a larger range of objects. This can possibly be done by adding more object classes to the training dataset. Also, a further analysis should be done to see if, by freezing the weights of some layers, the training procedure and the model performance are improved.

In terms of processing speed, an implementation of this Object Detection API in *C++* should be carried out. Since this is a low-level language, it executes faster than *Python* and, consequently, the real-time performance improves.

References

[1] Michael Blumenstein Matthew Browne, Steve Green and Rodger Tomlinson. The detection and quantification of persons in cluttered beach scenes using neural network-based classification. 2005.

[2] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. 2002.

[3] David G. Lowe. Distinctive image features from scale-invariant keypoints. January 2004.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gooll. Surf: Speeded up robust features. 2006.

[5] Nils J Nilsson. *Introduction to Machine Learning*. 1998.

[6] Y. Bengio P. Vincent, A. Courville. Representation learning: A review and new perspectives. *IEEE Trans. Special issue Learning Deep Architectures.*, pages 1798–1828, 2013.

[7] N. Horning. *Introduction to decision trees and random forests*. 2013.

[8] Assaad MOAWAD. Neural networks and back-propagation explained in a simple way. https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e, 2018.

[9] Dhanoop Karunakaran. Deep learning series 1: Intro to deep learning. https://medium.com/intro-to-artificial-intelligence/deep-learning-series-1-intro-to-deep-learning-abb1780ee20, 2018.

[10] Prabhu. Deep learning understanding of convolutional neural network(cnn). https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148, 2018.

[11] Sumit Saha. A comprehensive guide to convolutional neural networks. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2018.

[12] Ross Girshick Shaoqing Ren, Kaiming He and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. 2016.

[13] Ross Girshick. Fast r-cnn. 2015.

[14] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection.

[15] Dumitru Erhan Christian Szegedy Scott Reed Cheng-Yang Fu Alexander C. Berg Wei Liu, Dragomir Anguelov. Ssd: Single shot multibox detector. *arXiv*, 2016.

[16] Camron Godbout. Tensorflow in a nutshell. https://medium.com/@camrongodbout/tensorflow-in-a-nutshell-part-one-basics-3f4403709c9d, 2016.

[17] Kislay Keshari. Object detection tutorial in tensorflow: Real-time object detection. https://dzone.com/articles/object-detection-tutorial-in-tensorflow-real-time, 2019.

[18] Derek Hoiem Zhizhong Li. Learning without forgetting. *arXiv*, 2017.

[19] TensorFlow. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model _zoo.md.

[20] Ryszard S. Choras. Image feature extraction techniques and their applications for cbir and biometrics systems. *INTERNATIONAL JOURNAL OF BIOLOGY AND BIOMEDICAL ENGINEERING*, 2007.

[21] R. Wilson and M. Spann. A new approach to clustering. *Pattern Recognition*, pages 1413–1425, 1990.

[22] K. Fukunaga and L.D. Hosteler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Trans. Information Theory*, pages 32–40, 1975.

[23] Y. Cheng. Mean shift, mode seeking and clustering. *IEEE Trans. Pattern Analysis and Machine Intelligence*, pages 790–799, 1995.

[24] Andrew P. Witkin. Scale-space filtering. pages 1019–1022, 1983. In *International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany.

[25] Jan J. Koenderink. The structure of images. pages 363–396, 1984. *Biological Cybernetics*.

[26] Tony Lindeberg. Scale-space theory:a basic tool for analyzing structures at different scales. *Journal of Applied Statistics*, pages 224–270, 1994.

[27] David G. Lowe. Object recognition from local scale-invariant features. pages 1150–1157, 1999. In *International Conference on Computer Vision*,Corfu,Greece.

[28] E. Fix and J.L. Hodges. Discriminatory analysis.nonparametric discrimination: Consistency properties. 1951.

[29] Herbert Bay, Tinne Tuytelaars, and Luc Van Gooll. Surf: Speeded up robust features. pages 346–359, 2008. In *Computer Vision and Image Understanding (CVIU)*.

[30] Mathew Brown and David Lowe. Invariant features from interest point groups. 2002. In BMVC.

[31] Ethem Alpaydin. *Introduction to Machine Learning*. 2010.

[32] Ameet Talwalkar Mehryar Mohri, Afshin Rostamizadeh. *Foundations of Machine Learning*. 2012.

[33] Peter Norvig M Stuart J. Russell. *Artificial Intelligence: A Modern Approach*. 2010.

[34] P. Dayan. Unsupervised learning. *The MIT Encyclopedia of the Cognitive Sciences*, 1999.

[35] Andrew W. Moore Leslie P. Kaelbling, Michael L. Littman. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4*, pages 237–285, 1996.

[36] O. Maimon L. Rokach. *Data Mining with Decision Trees: Theory and Applications*. 2008.

[37] Warren S.Sarle. Neural networks and statistical models. *IEEE Trans. Special issue Learning Deep Architectures.*, pages 1798–1828, 2013.

[38] J.G. Makin. Backpropagation. 2006.

[39] Jurgen Schmidhuber. Deep learning in neural networks: An overview. 2014.

[40] Harikrishna B. Deep learning. https://medium.com/datadriveninvestor/deep-learning-2025e8c4a50, 2018.

[41] Jose Manuel Saavedra. Understanding object detection methods. https://medium.com/impresee/understanding-object-detection-methods-f7bb9d245c8c, 2018.

[42] TensorFlow. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md.

[43] TensorFlow. https://github.com/tensorflow/models.

[44] TensorFlow. https://www.tensorflow.org/tutorials/images/transfer_learning.

[45] Darrenl Tzutalin. https://github.com/tzutalin/labelImg.

[46] Thomas Gamauf. Tensorflow records? what they are and how to use them. https://medium.com/mostly-ai/tensorflow-records-what-they-are-and-how-to-use-them-c46bc4bbb564, 2018.

[47] Dat Tran. https://github.com/datitran/raccoon_dataset.

[48] TensorFlow. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/configuring_jobs.md.

[49] TensorFlow. https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs.

[50] TensorFlow. https://github.com/tensorflow/models/blob/master/research/object_detection/model_main.py.

[51] TensorFlow. https://www.tensorflow.org/tensorboard.

[52] TensorFlow. https://github.com/tensorflow/models/blob/master/research/object_detection/export_inference_graph.py.

[53] TensorFlow. https://github.com/tensorflow/models/blob/master/research/object_detection/object_detection_tutorial.ipynb.

[54] TensorFlow. https://github.com/tensorflow/models/blob/master/research/object_detection/protos/preprocesso